IM™ 2003

INTEGRATED NETWORK MANAGEMENT VIII

Managing It All

Edited by

Germán Goldszmidt
Jürgen Schönwälder

IFIP

SPRINGER-SCIENCE
+BUSINESS MEDIA,
B.V.

INTEGRATED NETWORK MANAGEMENT VIII

INTEGRATED NETWORK MANAGEMENT VIII

Managing It All

IFIP / IEEE Eighth International Symposium on Integrated Network Management (IM 2003) March 24–28, 2003, Colorado Springs, USA

Jointly sponsored by IFIP TC6/WG6.6 (Management of Networks and Distributed Systems) and the IEEE Communications Society

Edited by

Germán Goldszmidt
IBM Research
USA

Jürgen Schönwälder
University of Osnabrück
Germany

SPRINGER-SCIENCE+BUSINESS MEDIA, B.V.

Library of Congress Cataloging-in-Publication Data

A C.I.P. Catalogue record for this book is available from the Library of Congress.

INTEGRATED NETWORK MANAGEMENT: Managing It All
Edited by Germán Goldszmidt and Jürgen Schönwälder
ISBN 978-1-4757-5521-3 ISBN 978-0-387-35674-7 (eBook)
DOI 10.1007/978-0-387-35674-7

*The original version of the book frontmatter was revised:
The copyright line was incorrect. The Erratum
to the book frontmatter is available at
DOI: 10.1007/978-0-387-35674-7_66*

Contents

Preface

Welcome to IM 2003, the eighth in a series of the premier international technical conference in this field. As IT management has become mission critical to the economies of the developed world, our technical program has grown in relevance, strength and quality. Over the next few years, leading IT organizations will gradually move from identifying infrastructure problems to providing business services via automated, intelligent management systems. To be successful, these future management systems must provide global scalability, for instance, to support Grid computing and large numbers of pervasive devices. In Grid environments, organizations can pool desktops and servers, dynamically creating a virtual environment with huge processing power, and new management challenges. As the number, type, and criticality of devices connected to the Internet grows, new innovative solutions are required to address this unprecedented scale and management complexity. The growing penetration of technologies, such as WLANs, introduces new management challenges, particularly for performance and security.

Management systems must also support the management of business processes and their supporting technology infrastructure as integrated entities. They will need to significantly reduce the amount of adventitious, bootless data thrown at consoles, delivering instead a cogent view of the system state, while leaving the handling of lower level events to self-managed, multifarious systems and devices. There is a new emphasis on "autonomic" computing, building systems that can perform routine tasks without administrator intervention and take prescient actions to rapidly recover from potential software or hardware failures. Web services, with public interfaces and bindings described using XML, are often proposed as the latest model for interaction between management systems and applications.

These are some of the large background issues that provide a setting for the specific topics that will be covered by an ensemble of 41 papers, 6 panels, 24 posters, 16 tutorials, and 6 invited speakers. Our Technical Program Committee (TPC) consists of 48 members residing in 15 countries in Asia, Europe and the Americas. These members have helped us prepare an excellent technical program, covering areas such as Provisioning, Modeling, Wireless, Quality of Service, Faults, Power, Optical, Configuration, Peer to Peer, Intrusion Detection, Accounting, Policies, and Performance Management. In addition, we have invited six outstanding leaders to present multiple interesting perspectives on different aspects of IT that relate to our field.

We received 149 paper submissions from all over the world, representing research contributions in a wide variety of topics related to network and systems management. The submissions were distributed among TPC members and additional reviewers, resulting in 635 independent expert technical reviews, an average of 4.3 reviews per paper. After a rigorous evaluation process, 24 members of the TPC met on October 23-25, 2002, in Montreal, Canada. At this meeting, we strived to pick the final 41 high-quality papers for technical presentations in IM 2003, the most selective rate of acceptance of any IM conference to date. The topical coverage of the selected papers represents the excitement and diversity present in this field. In addition, the TPC selected 24 short papers that represent the latest ongoing research and development work on network operations and management. These short papers will also be presented in poster sessions, enabling greater interaction directly with the authors.

In closing, we thank the authors of all submitted manuscripts for their effort and creativity, and the reviewers who provided valuable feedback that helped the authors to evince their contributions. Special thanks to the members of the TPC who participated at the meeting in Montreal, and to all the additional reviewers, for their valuable and detailed comments. Our gratitude goes also to the session chairs who ensured that the accepted papers addressed the concerns raised by the reviewers. We warmly thank all those who have participated in and supported this conference, and in particular the active members of the Organizing and Technical Program Committees. Finally, a number of individuals deserve special mention, including Doug Zuckerman, our General Chair, for his inspiring leadership, Peter Kropf for his assistance in providing the local arrangements for the TPC meeting in Montreal, Gayle Weisman of IEEE Communications Society for her vim in managing the many aspects of this symposium, and Yana Lambert of Kluwer, for the production of these proceedings.

Germán Goldszmidt and Jürgen Schönwälder
TPC Co-Chairs
January 2003

Symposium Committees

ORGANIZING COMMITTEE

General Chair
Doug Zuckerman, *Doug Zuckerman Associates, USA*

Technical Program Co-Chairs
Germán Goldszmidt, *IBM Research, USA*
Jürgen Schönwälder, *University of Osnabrück, Germany*

Patron/Exhibits Chair
Shri Goyal, *Pragati-Path, USA*

Tutorial Program Chair
Gabi Dreo Rodosek, *Leibniz Supercomputing Center, Germany*

Panels Co-Chairs
Nelson Fonseca, *University of Campinas, Brazil*
Felix Wu, *University of California at Davis, USA*

Poster/Birds-of-a-Feather Chair
Masum Hasan, *Cisco, USA*

Keynotes/Distinguished Experts Panel Chair
Scott Marcus, *Federal Communications Commission, USA*

Finance Chair
Raouf Boutaba, *University of Waterloo, Canada*

Treasurer
Bruce Worthman, *IEEE Communications Society, USA*

Publicity Chair
Roberta S. Cohen, *IEEE Communications Society, USA*

Cynthia Hood, *Illinois Institute of Technology, USA*
Gabriel Jakobson, *Smart Solutions Consulting , USA*
Alexander Keller, *IBM Research, USA*
Yoshiaki Kiriha, *NEC, Japan*
Peter Kropf, *University of Montreal, Canada*
Lundy Lewis, *University of New Hampshire, USA*
Antonio Liotta, *University of Surrey, UK*
Emil Lupu, *Imperial College London, UK*
Hanan Lutfiyya, *University of Western Ontario, Canada*
Jean-Philippe Martin-Flatin, *CERN, Switzerland*
Subrata Mazumdar, *Avaya Labs Research, USA*
José Marcos Nogueira, *Federal University of Minas Gerais, Brazil*
George Pavlou, *University of Surrey, UK*
Aiko Pras, *University of Twente, The Netherlands*
Danny Raz, *Technion, Israel*
Joan Serrat, *Universitat Politecnica de Catalunya, Spain*
Adarshpal Sethi, *University of Delaware, USA*
Michelle Sibilla, *University Paul Sabatier, France*
Morris Sloman, *Imperial College London, UK*
Rolf Stadler, *KTH Stockholm, Sweden*
Burkhard Stiller, *ETH Zurich/UniBw Munich, Switzerland/Germany*
John Strassner, *Intelliden, USA*
Mehmet Ulema, *Manhattan College, USA*
Robert Weihmayer, *Verizon, USA*
Carlos Becker Westphall, *Federal University of Santa Catarina, Brazil*
Bert Wijnen, *Lucent Technologies*
Felix Wu, *University of California at Davis, USA*
Makoto Yoshida, *The University of Tokyo/NTT-AT, Japan*

REVIEWERS

Masayuki Abe, *NTT Info. Sharing Labs, Japan*
Seongbok Baik, *Illinois Institute of Technology, USA*
Arosha Bandara, *Imperial College London, UK*
Frederico Bastos, *Federal University of Minas Gerais, Brazil*
Michael Bauer, *University of Western Ontario, Canada*
Bela Berde, *Alcatel, France*
Christos Bohoris, *University of Surrey, UK*
Herbert Bos, *University Leiden, The Netherlands*
Seraphin Calo, *IBM Research, USA*
Geoff Carpenter, *FARGOS Development, LLC, USA*
Roberta Cohen, *IEEE Communications Society, USA*

Introduction

Managing It All

Overview

In recent years, the world economy, and especially the telecom sector, has been hard hit by massive 'corrections,' resulting in wide-scale industry redefinition and massive downsizing. The world is looking for solutions that will enable the recovery, with network management positioned to play a key role. IM 2003 will provide the meeting ground for experts and practitioners from around the world to present papers and interact with the goal of helping the world 'manage it all.' Of course, it may not be practical – or even desirable – to 'manage it all.' However, from a 'management-centric' point of view, management (and control) will play a pivotal role as providers of networks and services seek ways to restore business and social value and set a stage for long-term growth. It is in this spirit that IM 2003 was held at the Broadmoor Hotel in Colorado Springs, 24-28 March 2003. It is hoped that this proceedings will have long-lasting value in preserving and disseminating the many high-quality, influential papers presented at the conference.

History

Each IM (previously ISINM) had a general theme, hoping to capture the essence of what was most important in network management at a particular time. Indeed, many of the paper sessions, panels and tutorials addressed different aspects of the same theme issue throughout the course of the conference week and in the proceedings. So, let us look at the themes for a glimpse of IM's – and network management's – past:

1989 (Boston): Improving Global Communication through Network Management
1991 (Crystal City): Worldwide Advances in Integrated Network Management
1993 (San Francisco): Strategies for the Nineties
1995 (Santa Barbara): Rightsizing in the Nineties
1997 (San Diego): Integrated Management in a Virtual World

1999 (Boston): Distributed Management for the Networked Millennium
2001 (Seattle): Integrated Management Strategies for the New Millennium
2003 (Colorado Springs): Managing It All

In the beginnings, the main focus was on enlightening the IT industry and researchers on the importance and necessity of integrating diverse, often incompatible and proprietary, network management systems in a highly complex and costly operating environment. This was followed by development of standards (such as CMIP, SNMP and 'the older' TL-1), sometimes at odds with each other. For the Telco industry, it brought the TeleManagement Forum, whose role was (and continues to be) to pull together diverse approaches into workable solutions that meet service provider business needs. As the decade reached its close, focus turned to the New Millennium, which, if the Y2K bug didn't set us back to the Stone Age (or at least 1900), would be a 'networked' millennium, with an even more prominent role for management, even if more distributed than centralized. However, a year or two into the new millennium, a new reality became apparent: the Internet may not be growing quite as fast as predicted. There was likely a capacity glut, and many of the startups that sprung up like weeds during this phenomenal period of perceived network growth were just as quickly going out of business or being absorbed by larger, more stable firms. But then, many of these larger companies started becoming less stable and had no choice but to have massive force reductions to remain viable. It is hoped that during the IM 2003 year, the 'bottom' will have been reached, and that much of the leading-edge work presented in these proceedings will help provide the technological energy for a rebirth of our industries and a revitalization of future IM and NOMS events.

Future Events

As the management world continues evolving, this ongoing series of international symposia will continue to foster and promote cooperation among individuals of diverse and complementary backgrounds, and to encourage international information exchange on all aspects of network and distributed systems management. To broaden the scope of these symposia, the IEEE Communications Society Technical Committee on Network Operations and Management (CNOM), and the International Federation for Information Processing (IFIP) Working Group 6.6 on Management of Networks and Distributed Systems have been successfully collaborating on the two premier technical conference series in the area of network and systems management, operations and control – the International Symposium on Integrated Network Management (IM) and the Network Operations and Management Symposium (NOMS). IM is held in odd-numbered years, and NOMS is held in even-numbered years. NOMS 2004 will take place in Seoul, Korea on 19-23 April 2004. The next International Symposium on Integrated Network Management (IM 2005) is expected to be held in the Spring of 2005.

Starting in 1990, IEEE CNOM and IFIP WG 6.6 have also worked together to organize the International Workshops on Distributed Systems: Operations and Management (DSOM), which take place in October of every year. DSOM 2003 will take place 20-22 October 2003 in Heidelberg, Germany. For more information on future IM, NOMS, and DSOM events, and other related activities, please get in touch with the conference organizers.

Acknowledgments

Organizing IM 2003 began in April 2000, during NOMS. Organizing a conference like IM is a very big job. I thank the Organizing Committee members for their perseverance and proactive efforts to make this IM one of the best ever. In particular, I would like to thank Raouf Boutaba, our finance chair and representative of the IM/NOMS Steering Committee, for his persistent efforts at assuring a high value experience for the attendees. Also, since the technical program underpins all aspects of the overall program, I thank TPC Co-Chairs Germán Goldszmidt and Jürgen Schönwälder for insisting on the highest quality standards.

Special thanks also go to the organizing committee members: Robbie Cohen, Gabi Dreo Rodosek, Nelson Fonseca, Shri Goyal, Masum Hasan, Scott Marcus, Jerry McCollum, John Strassner, Kim Strassner and Felix Wu. Each of them played a key role on the IM 2003 organizing committee team.

Special thanks also go to several IEEE Communications Society staff members: Gayle Weisman diligently and cheerfully worked with us on almost every aspect of the event planning and implementation, serving as the primary 'point of contact' for almost all issues – thank you, Gayle! Bruce Worthman is to be thanked for helping us keep the finances straight and in line with Communications Society expectations. And, last but not least, thanks to Debora Kingston for leaving 'no clause unturned' in renegotiating our arrangements with the hotel.

I would also like to thank the IFIP WG 6.6 Chair, Raouf Boutaba, and the CNOM chair, Salah Aidarous, for the support they facilitated through these technical committees. Finally, I acknowledge and appreciate the substantive efforts of Branislav Meandzija and Wolfgang Zimmer in the early planning for IM 2003.

Douglas N. Zuckerman
General Chair
January 2003

SESSION 1

Anomaly / Intrusion Detection

Chair: Felix Wu
University of California at Davis, USA

AN SNMP AGENT FOR STATEFUL INTRUSION INSPECTION

Luciano Paschoal Gaspary
Universidade do Vale do Rio dos Sinos – Centro de Ciências Exatas e Tecnológicas
Av. Unisinos 950 – CEP 93.022-000 – São Leopoldo, Brazil
paschoal@exatas.unisinos.br

Edgar Meneghetti
Liane Rockenbach Tarouco
Universidade Federal do Rio Grande do Sul – Instituto de Informática
Av. Bento Gonçalves 9500 – CEP 91.591-970 – Porto Alegre, Brazil
edgar@cesup.ufrgs.br, liane@penta.ufrgs.br

Abstract: Intrusion Detection Systems (IDSs) have been increasingly used in organizations, in addition to other security mechanisms, to detect intrusions to systems and networks. In the recent years several IDSs have been released, but (a) the high number of false alarms generated, (b) the lack of a high-level notation for attack signature specification, and (c) the difficulty to integrate IDSs with existing network management infrastructure hinder their wide-spread and efficient use. In this paper we address these problems by presenting an SNMP agent for stateful intrusion inspection. By using a state machine-based language called PTSL (Protocol Trace Specification Language), the network manager can describe attack signatures that should be monitored. The signatures to be used by the agent are configured by the network manager through the IETF Script MIB. Once programmed, the agent starts monitoring the occurrence of the signatures on the network traffic and stores statistics, according to their occurrence, in an extended RMON2 MIB. These statistics may be retrieved from any SNMP-based management application and can be used to accomplish signature-based analysis. The paper also describes two experiments that have been carried out with the agent to assess its performance and to demonstrate its effectiveness in terms of false alarm generation rates.

Keywords: Network security management, intrusion detection, misuse detection, stateful inspection, intrusion detection SNMP agent, RMON2.

1. INTRODUCTION

Due mainly to the weaknesses of firewall technologies in blocking some malicious incoming network traffic and to the growing number of attacks being initiated from hosts located in the same network as the victim host, Intrusion Detection Systems (IDSs) have been incorporated into the organizations security infrastructure. In order to satisfy the demand for such systems, several IDSs as Snort [1], NFR [2], Bro [3] and Stat [4] (to mention just a few) have been released to the market in the recent years. Despite the intensive research on intrusion detection, (a) reducing the false alarm (false positive) rates generated by IDSs, (b) providing the network manager with a high-level notation for attack signature specification, and (c) integrating security mechanisms with the already existing network management infrastructure are some of the current challenges in the field.

Intrusion Detection Systems employ either *anomaly* or *signature analysis* (*misuse*) to detect attacks. Anomaly-based analysis uses statistical methods to distinguish normal from unexpected behavior, while signature-based analysis tries to match the content of data sources (e.g. network packets and system logs) with the specification of known attacks (attack signatures). Regardless of the technique used, around 90% of the alarms generated by an IDS are false positives [5].

In IDSs that employ anomaly analysis, it is difficult to determine what should be considered normal and abnormal. While attacks such as distributed denial of service, which generate considerable network traffic changes, may be detected at close to zero percent false positive rates (good examples may be found in [6, 7]), the same does not happen to many attacks that do not produce such substantial changes in the network traffic. In the case of these more subtle attacks, depending on the thresholds defined, either they may not be detected or many false alarms may be generated.

With IDSs that use signature analysis, the problem of high false alarm rates also occurs, mainly due to the limited expressive capability of the languages available to model attack signatures and to incomplete representation of signatures. In general, the signature concern is to observe packet fields and not protocol interactions (stateless inspection). An example of the effect of this limitation is the behavior of some IDSs when configured to detect the TCP SYN/TCP RST port scanning technique. The signature used by them consists of the observation of TCP packets with the RST flag on. The problem of this approach is that a TCP RST packet is not generated only by a station that does not have a certain kind of service available (when it receives a connection request). A station also uses this type of packet in order to restart an ongoing connection. As those IDSs are not able to correlate packets or the signature is not precise enough they cannot distinguish between TCP RSTs that represent port scanning from those used during a conventional connection, triggering alarms in both cases. The problem of false alarm generation is also related to the fact that we are not always able to capture the essence of the potential threat. The techniques used by the intruders and the threats posed by them to the system evolve over time and become more sophisticated, while the signatures lag behind. In some cases, there is also a motivation to specify a signature that will generate a large false alarm rate because the intent is to capture and analyse other but similar hypothetical scenarios.

With reference to the integration of security mechanisms and current Network Management Systems (NMSs), there is still a wide gap between security and network management, despite some initiatives (as the ones proposed in [6, 7, 8]). There is no Management Information Base (MIB) related to intrusion detection available. Several Network Management Systems offer an interface to configure Intrusion Detection Systems and are able to receive events generated by them. However, as stated by Qin et al., these systems lack efficient and effective capabilities of analyzing and managing the alarms sent by the IDS.

This paper addresses the false alarm rate, the lack of a high-level notation for attack signature specification and the lack of integration of IDSs and NMSs problems, by presenting an SNMP agent for intrusion detection that makes stateful inspection of data (packets) collected directly from the network. By using a state machine-based language called PTSL (Protocol Trace Specification Language) [9], the network manager can describe attack signatures that should be monitored. The signatures to be used by the agent are configured by the network manager through the IETF Script MIB [10]. Once programmed, the agent starts monitoring the occurrence of the signatures on the network traffic and stores statistics, according to their occurrence, in an extended RMON2 MIB [11]. These statistics may be retrieved from any ordinary SNMP-based management application and can be used to accomplish signature-based analysis.

The main contribution of our work is the development of an agent that is able to do stateful inspection. As it will be shown along the paper, the agent provides accurate detection of both brute force and subtle attacks (concerning network traffic pattern changes). We have also developed a high-level and easy-to-learn language to specify attack signatures. The agent is fully integrated to the SNMP architecture. The configuration of the agent and the retrieval of results may be done using the SNMPv3 protocol.

The paper is organized as follows. Section 2 presents some related work. Section 3 presents PTSL language and some attack signatures described using this language. Section 4 approaches the internal architecture of the agent. In section 5 experiments that have been carried out with the agent are described. Section 6 closes the paper by presenting some final remarks and future work perspectives.

2. RELATED WORK

The problem of false alarms originates from the lack of accuracy in the process of detecting intrusions. As mentioned in the introduction, in the case of signature-based IDSs this imprecision is closely related (a) to the capabilities of the attack signature specification language provided and the respective intrusion detection engine or (b) to inprecise signature representation. Snort [1], for instance, uses a pattern matching model for detection of network attack signatures using identifiers such as TCP fields, IP addresses, TCP/UDP port numbers, ICMP type/code, and strings contained in the packet payload. Filtering rules are applied to each packet and stateful analysis is only partially provided (limited to TCP stream reassembly and inspection, and detection of some portscan e fingerprinting attacks), leading to a high number of false alarms. NFR [2] and Bro [3] suffer from the same problem. Statl [4], on the other hand, is an extensible state/transition-based attack description language used by Stat intrusion detection suite. This language allows one to describe computer penetrations as sequences of actions that an attacker performs to compromise a computer system. The detailed description of the signatures specified in Statl results in a lower number of false alarms (if compared to Snort, NFR and Bro).

Julisch et al propose in [12] some techniques to process alarm logs and filter false positives. This approach is based on the identification of alarm patterns, on the understanding of their root cause and, if non-malicious, on the usage of these alarm patterns for filtering. Finding filtering rules and the risk of filtering out true positives are some of the difficulties to implementing this approach.

None of the Intrusion Detection Systems listed so far offer mechanisms to make their integration with Network Management Systems easier. In the recent years, however, some efforts have been made to bridge this gap [6, 7, 8]. Qin et al. have proposed in [6, 7] the use of MIB II variables for network intrusion detection. This detection technique is clearly anomaly-based and, therefore, tend to be more efficient to detect attacks that generate considerable changes in the network traffic. In [8], Qin et. al extend their previous work by proposing (a) an approach to integrate NMSs and IDSs and (b) a hierarchical correlation architecture for improving the detection accuracy and identifying coordinated intrusions.

Our work should be regarded as a complement to the efforts just mentioned. By proposing an SNMP agent for stateful intrusion inspection we provide an alternate approach, based on signature analysis, that is able to cope with both traffic-based intrusions (e.g DDoS) and slower traffic and stealth attacks, generating few false alarms. In the next section we introduce PTSL (Protocol Trace Specification Language) that is the language to be used by the network manager to specify attack signatures.

3. REPRESENTATION OF ATTACK SIGNATURES USING PTSL

PTSL (Protocol Trace Specification Language) is a language developed to allow the representation of protocol traces based on the concept of finite state machines (FSM). It is part of Trace, an architecture that supports high-layer protocol, service and application management through passive observation of protocol interactions (traces) in the network traffic. A full description of the language can be found in [9]. In this paper, we focus on how PTSL can be used to describe stateful network-based attack signatures.

The language is composed of graphical (Graphical PTSL) and textual (Textual PTSL) notations. These notations are not equivalent. The textual notation allows the complete representation of a trace (attack signature), including the specification of the FSM and the events that trigger transitions. In turn, the graphical notation covers only a subset of the textual notation, offering the possibility of graphically representing the FSM and only labelling the events that trigger transitions.

3.1 Graphical PTSL Notation

Several attack signatures have been modelled using PTSL. Figures 1 and 2 illustrate some of these specifications, described using the graphical notation of the language. Figure 1 shows a signature that can be used to detect the TCP SYN/TCP RST port scanning technique.

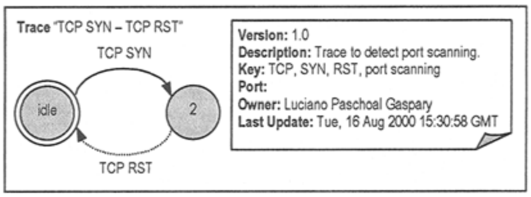

Trace "TCP SYN – TCP RST"
TCP SYN
idle
2
TCP RST
Version: 1.0
Description: Trace to detect port scanning.
Key: TCP, SYN, RST, port scanning
Port:
Owner: Luciano Paschoal Gaspary
Last Update: Tue, 16 Aug 2000 15:30:58 GMT

Figure 1. Signature to detect TCP SYN/TCP RST port scanning

Figure 2 shows other examples. In (a) one can see the signature to detect the `rpcinfo` command (available in Unix environments). This command returns a list of server processes that accept RPCs (Remote Procedure Calls), which is a useful information for the intruder. Similarly, in (b) it is shown the signature to detect the `showmount` command. Although (a) and (b) may appear in legitimate traffic, the occurrence of these signatures during unusual time periods or with high frequency can be regarded as attack evidence (e.g. someone scanning stations running `portmapper`). The signature described in (c) is composed of an HTTP request where the attacker uses the string /scripts/..\%C0\%AF../winnt/system32/cmd.exe?/c+dir+c:\ as argument. An URL like this indicates that he intends to execute some script or CGI at the HTTP server to obtain a list of the files located in the server. The signature to detect the SYN flood attack is depicted in (d). This attack consists of sending a huge number of connection setup packets (TCP packet with the SYN flag on with a fake source address) to a target host. This fake address must be unreachable or non-existent (usually a reserved value). When the target host receives these SYN packets, it creates a new entry on its connection table and sends a SYN/ACK packet back to the possible client. After sending the reply packet, the target host waits for acknowledgement from the client to establish the connection. As the source address is fake, the server will wait a long time for this reply. In a given time, the connection queue of the server will be full and all new connection requests will be discarded, creating a denial of service. Unlike other examples presented, this attack is identified by observing unsuccessful occurrences of the trace.

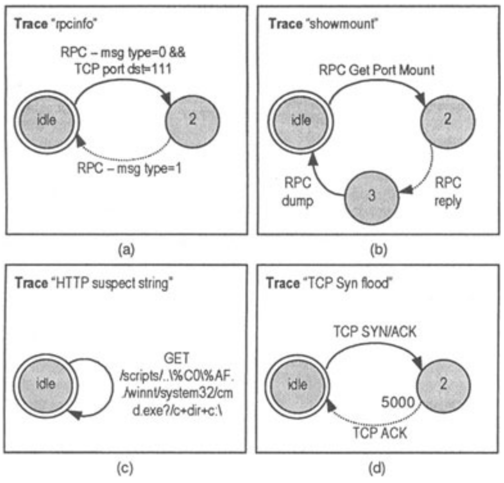


Figure 2. Graphical representation of attack signatures

Representation of states and transitions. As for PTSL graphical notation, one can observe from the previous examples that states are represented by circles. From the initial state (`idle`) other n states can be created, but they must always be reachable. The final state is identified by two concentric circles. In all examples presented, the initial and final states are the same. State transitions are represented by unidirectional arrows. The continuous arrow indicates that the transition is triggered by the client host, while the dotted arrow determines that the transition is triggered by an event coming from the server host. The text associated with a transition is merely a label to the event that triggers it; the full specification can only be made via textual notation.

Representation of timeouts. Transitions, by default, do not have a time limit to be triggered. To associate a timeout with a transition, an explicit value (in milliseconds) must be set. In the example shown in figure 2d, the value 5000 associated to transition TCP ACK indicates that the transition from state 2 to the initial state has up to five seconds to be triggered.

Representation of information for cataloging and version control. The graphical notation also offers a constructor where information about the signature, which are relevant to cataloguing and version control of specifications, are included. The data stored for a signature are: `Version`, `Description`, `Key`, `Owner` and `Last Update`. Besides these data, there is also a `Port` field, used to indicate the TCP or UDP port of the monitored protocol.

3.2 Textual PTSL Notation

Figure 3 presents the textual specification of the signature previously shown in figure 1. All specifications written in Textual PTSL start with the `Trace` keyword and end with the `EndTrace` keyword (lines 1 and 36). Catalog and version control information come right after the `Trace` keyword (lines 2–7). Forthwith, the specification is split into three sections: `MessagesSection` (lines 9–21), `GroupsSection` (not used in this example and not detailed in the paper) and `StatesSection` (lines 23–34). In `MessagesSection` and `GroupsSection` the events that trigger transitions are defined. The FSM that specifies the trace is defined in `StatesSection`.

Representation of messages. Whenever the fields of a captured packet match the ones specified at a `Message` for the current state, a transition is triggered in the FSM. The way those fields are specified depends on the type of protocol to be monitored. In the case of binary protocols (e.g. IP, TCP and UDP), known by their fixed length fields, the identification of a field is determined by a bit offset starting from the beginning of the protocol header; it is also needed to specify the size of the field, in bits (this is the `BitCounter` strategy). On the other hand, in the case of variable-length character-based protocols, where fields are usually split by white space characters (e.g. HTTP), the identification of a field is made by its position inside the message (`FieldCounter` strategy). In GET /scripts/..\%C0\%AF../winnt/system32/cmd.exe?/c+dir+c:\, for instance, GET is at position 0 and /scripts/..\%C0\%AF../winnt/system32/cmd.exe?/c+dir+c:\ is at position 1.

The signature shown in figure 1 is for a binary protocol. The TCP SYN message specification is shown in figure 3 (lines 11–14). In line 12 the message is defined as being of type `client`, meaning that the state transition associated with the message will be triggered by the client host. In line 13 the only field that is supposed to be analyzed is specified. All information necessary to identify it are: fetch strategy (`BitCounter`), protocol encapsulation (`Ethernet/IP`), field position (`110`), field length (`1`), expected value (`1`), comparison operator (`=`), and, optionally, a field description. The reply message TCP RST is shown in lines 16–19. The message type is defined in line 17 as `server`, i.e., the state transition will be triggered by the server host. Finally, the field to be analyzed is defined in line 18. Since the messages of the signatures illustrated in figure 2a, b and d are composed of binary protocol fields, they should be specified in a similar way.

```
Trace "TCP SYN - TCP RST"
Version: 1.0
Description: Trace to detect port scanning.
Key: TCP, SYN, RST, port scanning
Port:
Owner: Luciano Paschoal Gaspary
Last Update: Tue, 16 Aug 2000 15:30:58 GMT

MessagesSection

Message "TCP SYN"
MessageType: client
BitCounter Ethernet/IP 110 1 1 = "Field SYN  - 1 means TCP Connect"
EndMessage

Message "TCP RST"
MessageType: server
BitCounter Ethernet/IP 109 1 1 = "Field RST"
EndMessage

EndMessagesSection

StatesSection
FinalState idle

State idle
"TCP SYN" GotoState 2
EndState

State 2
"TCP RST" GotoState idle
EndState

EndStatesSection

EndTrace
```

Figure 3. Representation of a signature using Textual PTSL

As opposed to the example mentioned above, the signature specified in figure 2c is for a character-based protocol (HTTP). The attack is composed of a single transition and is recognized whenever an HTTP GET request packet with the argument /scripts/..\%C0\%AF../winnt/ system32/cmd.exe?/c+dir+c:\ is observed. Figure 4 presents part (the `MessagesSection`) of the textual specification for the trace shown in figure 2c. Lines 3–8 describe the HTTP request. In line 5 the GET field is defined. The information needed to identify a character-based protocol field are: fetch strategy (`FieldCounter`), protocol encapsulation (`Ethernet/IP/TCP`), field position (0), expected value (GET), comparison operator (=), and, optionally, a field description.

```
1  MessagesSection
2
3  Message "GET /scripts/..\%C0\%AF../winnt/system32/cmd.exe?/c+dir+c:\"
4  MessageType: client
5  FieldCounter Ethernet/IP/TCP 0 GET =
6  FieldCounter Ethernet/IP/TCP 1 /scripts/..\%C0\%AF../winnt/system32/cmd.exe?
7  /c+dir+c:\ =
8  EndMessage
9
10 EndMessagesSection
```

Figure 4. Field identification in character-based protocols

Representation of the FSM. Lines 23–34 in figure 3 define the textual specification of the state machine shown in figure 1. The final state is identified just after `StatesSection` (line 24). The states `idle` and 2 are defined in lines 26–28 and 30–32, respectively. The state specification only lists the events (messages and groups) that may trigger transitions, indicating, for each of them, which is the next state (lines 27 and 31).

4. THE INTRUSION DETECTION SNMP AGENT

The intrusion detection agent requires as input attack signatures specified in PTSL. The configuration of which signatures should be monitored at a given moment is made by the network manager through the Script MIB. Once programmed, the agent starts monitoring the occurrence of the signatures on the network traffic and stores statistics, according to their occurrence, in an extended RMON2 MIB. These statistics may be retrieved from any SNMP-based management application by periodically polling the agent. In order to reduce management traffic, it is possible to use the `alarm` and `event` RMON MIB groups instead. In this case, the network manager must configure thresholds to certain RMON2 MIB variables and define notifications that will be sent to the management station when these thresholds are reached. Figure 5a and b illustrates the communication flow between manager and agent considering both approaches. Expression [13] and Event [14] MIBs could also be used. The former provides the manager with a flexible mechanism to define thresholds (based on expressions), while the latter extends the capabilities of the RMON `alarm` and `event` groups by allowing alarms to be generated for MIB objects that are on another network element.

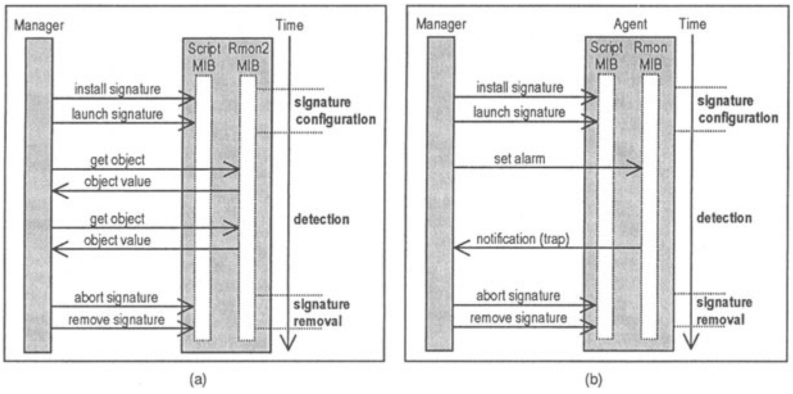


Figure 5. Communication flow between manager and agent

4.1 Architecture

The intrusion detection SNMP agent runs on Linux stations and was implemented using the C language, the POSIX thread library, the NET-SNMP framework [15] and Jasmin [16]. Figure 6 shows the agent architecture. The PTSL manager thread is responsible for the integration between the Script MIB and the PTSL core. It updates both the data structures used by the PTSL core and the RMON2 `protocolDir` table whenever a new signature is configured to be monitored or an existing signature is requested to be removed from the agent. Three more threads – queue, PTSL engine and RMON2 – operate in a producer/consumer fashion. The first thread captures all the packets arriving at the network interface card using the libpcap library and inserts them in a circular queue. The second thread processes every packet in the queue, without removing them from it, to identify if they have the characteristics expected to allow one or more signatures to evolve in the state machine. If so, special marks are attached to the packets. Finally, the RMON2 thread removes every packet from the queue and, according to the markings, updates the RMON2 tables in the mySQL database.

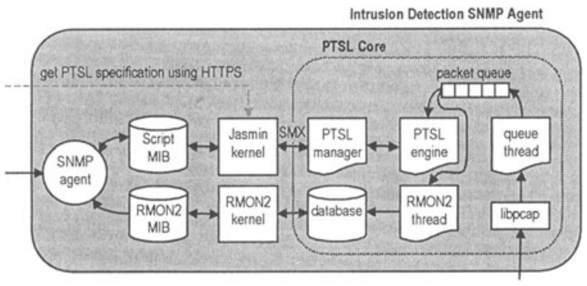


Figure 6. Internal organization of the agent

4.2 The Management Information Base

Every time that a signature is observed between any pair of hosts, data is stored in the mySQL database. This database is source of information for the SNMP sub-agent that implements an extended version of the RMON2 MIB [11]. One of the differences between our MIB and RMON2 is that the `protocolDir` group, which indicates the protocol encapsulations that the agent is able to monitor, now allows protocol traces (attack signatures) to be indexed. Therefore, monitoring granularity is considerably increased. Besides gathering statistics about the traffic generated by hosts using certain protocols, the agent also stores information related to the occurrence of attack signatures. Table 1 shows a set of entries that could appear in an agent protocol directory.

Table 1. RMON2 protocolDir table

ID	*LocalIndex*	*Description*
00-00-00-01-00-00-08-00	1	ether2.ip
00-00-00-01-00-00-08-00-00-00-00-17	2	ether2.ip.tcp
00-00-00-01-00-00-08-00-00-00-00-17-00-00-00-50	3	ether2.ip.tcp.http
00-00-00-01-00-00-08-00-00-00-00-17-00-01-00-04	4	ether2.ip.tcp.rpcinfo
00-00-00-01-00-00-08-00-00-00-00-17-00-01-00-05	5	ether2.ip.tcp.showmount
00-00-00-01-00-00-08-00-00-00-00-17-00-01-00-06	6	ether2.ip.tcp.http suspect string

As it was mentioned in the previous sub-section, the `protocolDir` table is automatically updated when a new signature is configured to be monitored or an existing signature is requested to be removed from the agent. The ID (`protocolDirID`) is composed of $n \times 4$ bytes, where n is the number of protocols that comprise the encapsulation [17]. The number used to identify ethernet (00-00-00-01), IP (00-00-08-00) and high-layer protocols is never longer than 16 bits (2 bytes). Hence, to avoid conflicts with the identification of existing protocols, IDs above 65.535 are assigned to attack signatures (see table 1 above).

We have implemented most of the RMON2 groups, including `nlHost`, `alHost`, `nlMatrix`, and `alMatrix`. The later stores statistical data about the signature when it is observed between each pair of hosts at the granularity of attack signatures. Table 2 illustrates the contents of the `alMatrixSD` table. The semantic of the MIB has not been changed, since it still stores packet and octet rates. To determine the number of occurrences of a signature between two hosts it is necessary to divide the number of packets (stored in the MIB) by the number of transitions that form the signature. From the third line of the table, for example, one can infer that the signature TCP SYN – TCP RST has been observed 127 times (254 packets divided by 2 transitions).

Table 2. Information from the alMatrixSD table

Source Address	*Destination Address*	*Protocol*	*Packets*	*Octets*
172.16.108.1	172.16.108.2	ether2.ip.tcp.http suspect string	4	4.350
172.16.108.32	172.16.108.2	ether2.ip.tcp.rpcinfo	8	7.300
172.16.108.1	172.16.108.254	ether2.ip.tcp syn-tcp rst	254	1.202.126
125.120.10.100	172.16.108.254	ether2.ip.tcp.showmount	20	3.204

4.3 Signature-based Intrusion Detection

The agent can be used to accomplish signature-based intrusion detection. Figures 1 and 2c and d show examples of signatures that, regardless of when they are observed, indicate the occurrence of a scanning (figure 1) or attack (figure 2c and d). To accurately detect them it is necessary to define how many occurrences of the signature should be observed within a time interval in order to be considered an attack. Figure 7 presents a sample Tcl script that could be used to install and monitor the occurrence of these signatures.

The programming of the Script MIB on the agent is made with the aid of a specially developed package (line 2). In lines 10–13 the PTSL signature is installed. In line 14 the agent is asked to start observing the network for the occurrence of the signature just installed. Then, the script polls the agent every 120 second (line 26) to get information (line 18) and checks whether the signature has been counted or not (line 20). If the signature has been observed three times within an interval an alarm is generated.

The examples presented in figure 2a and b, on the other hand, can be part of legitimate traffic. Therefore, the identification of these network patterns as part of an attack is not straightforward. In this case, the network manager must have a precise characterization of the network (baseline) to be able to create rules to efficiently detect when such traffic can be regarded an attack evidence.

5. EVALUATION OF THE AGENT

This section describes two experiments that were accomplished to assess the performance of the agent and its behavior regarding false positive generation rates.

```
package require Tnm 3.0
package require ScriptMib 1.0

set oid "protocolDist.protocolDistStatsEntry.protocolDistStatsPkts.1.10"
set prev 0

if { [catch {::Tnm::snmp generator -address $agent} s] } {
   ::Tnm::log exit -code runtimeError "Error creating SNMP session: $s"
   }
if { [catch {ScriptMib::InstallScript $ma $m_owner $m_name $m_lang $m_src \
   $m_descr $m_args $m_ltime $m_etime $m_mrun $m_mcomp} e]} {
   ::Tnm::log exit -code runtimeError "Error installing script: $e"
   }
::ScriptMib::RunScript $ma $m_owner $m_name 0

proc monitor {} {
   global s oid prev
   set val [$s get $oid]
   set val [lindex [lindex $val 0] 2];
   if {[expr $val - $prev] > 3} {
      ::Tnm::log "Intrusion alarm: $m_name, $m_descr"
   }
   set prev $val
}
::Tnm::job create \
   -interval 120000 -error {::Tnm::log exit -code runtimeError $errorInfo} \
   -exit {::Tnm::log exit} -command {monitor}
vwait forever
```

Figure 7. Sample script to install and monitor the occurrence of an attack

5.1 Performance Analysis

In order to identify the network load supported by the agent (without dropping packets) some experiments have been carried out. The test environment was composed of three hosts connected through a 10 Mbps IEEE 802.3 network segment. The first host was used to generate network traffic, while the second one was supposed to receive it. The third host, a 450 MHz K6II PC with 64 MB RAM, was used to run the agent. The results obtained were the following:

- The sustained agent capacity (without packet loss) is around 235 datagrams per second when one signature is monitored. This rate was obtained by the consecutive generation of UDP datagram sequences that matched exactly the signature configured;
- The increase in the number of signatures monitored causes performance degradation of the agent. The agent capacity was reduced to 172 datagrams/second when it was configured to monitor five signatures simultaneously and traffic that matched exactly these signatures was generated;
- When generating traffic at 10 Mbps (around 5000 datagrams/second), with all datagrams being part of the signature, the agent discards packets.

We have also run the agent on a small production network, characterized as follows: (a) IEEE 802.3 network running at 10 Mbps, (b) 10 hosts (connected to a hub) running Windows operating system and configured to share files and printers, (c) average traffic rate of 150 packets/second during prime hours, (d) 75% of the packets was between 65 and 256 bytes long and 21% of the packets was longer than 1024 bytes, and (e) application protocols composed of HTTP (45%), NetBIOS (27%) and SMTP (15%). In this scenario, packet discards have not been observed.

5.2 Alarm Generation Analysis

To demonstrate that our approach tend to generate few false positives we have carried out an experiment based on previous work done at Lincoln Laboratory [18]. The main idea is to create background traffic that is similar to the traffic observed on the production network. In the next step packets corresponding to attacks are merged to the background traffic. By reproducing the resulting traffic, one or more IDSs can be evaluated regarding false positive generation rates. In this experiment we have assessed our agent.

We have used the packet trace file available at Lincoln Laboratory named DDos 1.0. It is a distributed denial of service attack that explores a buffer overflow technique in a `sadmind` server running on Solaris operating system. The attack has five phases:

- *Phase 1 (host scanning)*: the purpose of this phase is to identify which hosts are up and running by sending ICMP echo request packets to all hosts of the network. This phase was cut from the original packet trace and not considered in the evaluation, because ICMP echos and replies appear a lot in legitimate traffic (would generate many false positives);
- *Phase 2 (look for sadmind running on a target)*: in this phase the `portmapper` of the hosts (that replied the ICMP echo request packets in the previous phase) are queried in order to identify which port the `sadmind` daemon is listening. Next, a TCP connection to this port is opened;
- *Phase 3 (try to compromise the target)*: once the TCP connection is established, a buffer overflow technique is used to edit the password file and create a new user account;
- *Phase 4 (identify which target has been compromised)*: in this phase one must open a telnet connection using the user account created in the previous step;
- *Phase 5 (launch a distributed denial of service attack)*: this phase was also cut from the original packet trace and not considered in our evaluation. We have focused on the detection of the earlier stages of the attack, because we believe that detecting the distributed denial of service when it is occurring is not very useful, since very little can be done against it.

The packet trace just described was merged to background traffic, collected from an IEEE 802.3 network segment composed of 10 personal computers. The traffic is characterized as follows: the application protocols used were NetBios (48%), HTTP (22%), mail (11%), FTP (8%), SSH (3%) and other (8%). The packet size distribution was: 38% (<64 bytes), 51% ($\geq$ 65 and <128 bytes), 1% ($\geq$ 128 and <256 bytes), 1% ($\geq$ 256 and <512 bytes), 8% ($\geq$ 512 and <1024 bytes) and 1% ($\geq$ 1024 bytes). It is important to highlight that the background traffic has some legitimate `sadmind` traffic.

The test platform consisted of three hosts connected to a hub. The first host was used as the traffic generator. The second host ran the agent prototype. The agent was configured to monitor several attack signatures (including the ones presented in the paper and the signatures to detect phases 2, 3 and 4 of the attack). The third host executed a MIB browser, which was configured to poll the agent once a second (to get the value associated to the `protocolDistStatsPkts` variable).

We have then used `tcpreplay` [19] to reproduce the traffic. As we did not want the agent to discard packets (to be able to focus the evaluation on the accuracy of the detection process), we have replayed the traffic at low speed.

Our agent has triggered three alarms related, respectively, to phases 2, 3 and 4 of the attack. No false positives have been generated. We believe this has occurred due to the mechanism adopted by the agent, which is able to analyze packet sequences (stateful analysis). Packet correlation (intrinsic characteristic of the agent) helps on distinguishing legitim and attack traffic.

6. CONCLUSIONS AND FUTURE WORK

This paper presented an SNMP agent for stateful intrusion inspection. We have also presented PTSL, a language for the representation of protocol traces that, in this paper, was used to model attack signatures. Then we have presented some experiment results related to the agent performance and alarm generation rates. Providing the network manager with a high-level notation for attack signature specification, reducing the false positive rates generated, and integrating security mechanisms with the already existing network management infrastructure were the general objectives of the work. It is important to highlight that the absence of false alarms in our experiment is due to the programming of the signatures, and not due to the network management portion of our agent.

PTSL language has shown to be very adequate for the specification of attack signatures because of its simplicity. The expression power of PTSL is another point to be highlighted. The possibility of correlating packets, whether from the same flow or not, enables the identification of attacks with a low error rate, considerably reducing the number of false alarms. Besides, while most IDSs allow the selection of packets based on a few predetermined header fields only up to the transport layer, PTSL goes beyond, allowing the use of filters based on any protocol, all the way up to the application layer.

The use of an extended RMON2 MIB to store information related to the occurrence of attack signatures is a significant step towards integration of security mechanisms with the current existing SNMP-based management applications and platforms. Our agent should not be used isolated from other approaches. While the work published by Qin et al. in [6, 7, 8] tend to be more efficient to detect attacks that generate considerable changes in the network traffic (anomaly-based detection), our agent is able to cope with both traffic-based intrusions and slower traffic and stealth attacks (since it is signature-based).

Regarding security, the agent supports all facilities provided by SNMPv3, including the User-based Security Model (USM) [20] and the View-based Access Control Model (VACM) [21]. Using these facilities it is assured that the agent cannot be "reprogrammed" by a person who is not allowed to do this.

According to the results presented in sub-section 5.1 one can observe that the passive network traffic monitoring technique is computationally onerous. To achieve better results and not to compromise the intrusion detection process we have considered the following alternatives: use of hosts with more than one processor, distribution of signatures to more than one host, filtering out packets that are not useful for the signatures being monitored (using BPF filters), and replacement of the mySQL database by a more efficient alternative.

References

[1] Snort The Open Source Network Intrusion Detection System. `http://www.snort.org/`.

[2] NFR Security. `http://www.nfr.net/`.

[3] V. Paxson. Bro: A System for Detecting Network Intruders in Real-Time. *Computer Networks*, 31(23–24), Dec. 1999, p. 2435–2463.

[4] G. Vigna, S. T. Eckmann, and R. A. Kemmerer. The STAT Tool Suite. In *Proceedings of DARPA Information Survivability Conference & Exposition (DISCEX 2000)*, 2000.

[5] D. Alessandri. Using Rule-Based Activity Descriptions to Evaluate Intrusion-Detection Systems. In *Proceedings of International Workshop on the Recent Advances on Intrusion Detection (RAID 2000)*, 2000.

[6] J. B. D. Cabrera, L. Lewis, X. Qin, W. Lee, R. K. Prasanth, B. Ravichandran, and R. K. Mehra. Proactive Detection of Distributed Denial of Service Attacks using MIB Traffic Variables – a Feasibility Study. In *Proceedings of IFIP/IEEE International Symposium on Integrated Management (IM 2001)*, 2001.

[7] X. Qin, W. Lee, L. Lewis, and J. B. D. Cabrera. Using MIB II Variables for Network Intrusion Detection. *Data Mining for Security Applications, Advances in Computer Security*. Kluwer Academic Press, March 2002.

[8] X. Qin, W. Lee, L. Lewis, and J. B. D. Cabrera. Integrating Intrusion Detection and Network Management. In *Proceedings of IFIP/IEEE Network Operations and Management Symposium (NOMS 2002)*, 2002, p. 329–344.

[9] L. P. Gaspary, L. F. Balbinot, and L. R. Tarouco. Monitoring High-Layer Protocol Behavior Using the Trace Architecture. In *Proceedings of Latin American Network Operation and Management Symposium (LANOMS 2001)*, 2001, p. 99–110.

[10] D. Levi and J. Schönwälder. Definitions of Managed Objects for the Delegation of Management Scripts. *RFC 3165*, Aug. 2001.

[11] S. Waldbusser. Remote Network Monitoring Management Information Base Version 2 using SMIv2. *RFC 2021*, Jan. 1997.

[12] K. Julisch. Dealing with False Positives in Intrusion Detection. In *Proceedings of International Workshop on the Recent Advances on Intrusion Detection (RAID 2000)*, 2000.

[13] R. Kavasseri and B. Stewart. Distributed Management Expression MIB. *RFC 2982*, Oct. 2000.

[14] R. Kavasseri and B. Stewart. Event MIB. *RFC 2981*, Oct. 2000.

[15] NET-SNMP. `http://net-snmp.sourceforge.net/`.

[16] Jasmin - A Script MIB Implementation. `http://www.ibr.cu.tu-bs.de/projects/jasmin`.

[17] A. Bierman, C. Bucci, and R. Iddon. Remote Network Monitoring MIB Protocol Identifier Reference. *RFC 2895*, Aug. 2000.

[18] R. Lippmann et al. Evaluating Intrusion Detection Systems: the 1998 DARPA Off-line Intrusion Detection Evaluation. In *Proceedings of DARPA Information Survivability Conference & Exposition (DISCEX 2000)*, 2000.

[19] Tcpreplay. `http://sourceforge.net/projects/tcpreplay/`.

[20] U. Blumenthal and B. Wijnen. User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3). *RFC 2574*, Apr. 1999.

[21] B. Wijnen, R. Presuhn, and K. McCloghrie. View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP). *RFC 2575*, Apr. 1999.

FIREWALL POLICY ADVISOR FOR ANOMALY DISCOVERY AND RULE EDITING

Ehab S. Al-Shaer and Hazem H. Hamed
Multimedia Networking Research Laboratory
School of Computer Science, Telecommunications and Information Systems
DePaul University, Chicago, USA
{ehab,hhamed}@cs.depaul.edu

Abstract: Firewalls are core elements in network security. However, managing firewall rules, especially for enterprize networks, has become complex and error-prone. Firewall filtering rules have to be carefully written and organized in order to correctly implement the security policy. In addition, inserting or modifying a filtering rule requires thorough analysis of the relationship between this rule and other rules in order to determine the proper order of this rule and commit the updates. In this paper, we present a set of techniques and algorithms that provide (1) automatic discovery of firewall policy anomalies to reveal rule conflicts and potential problems in legacy firewalls, and (2) anomaly-free policy editing for rule insertion, removal and modification. This is implemented in a user-friendly tool called "Firewall Policy Advisor." The Firewall Policy Advisor significantly simplifies the management of any generic firewall policy written as filtering rules, while minimizing network vulnerability due to firewall rule misconfiguration.

Keywords: Firewall, security management, security policy, policy conflict.

1. Introduction

With the global Internet connection, network security has gained significant attention in the research and industrial communities. Due to the increasing threat of network attacks, firewalls have become important integrated elements not only in enterprize networks but also in small-size and home networks. Firewalls have been the frontier defense for secure networks against attacks and unauthorized traffic by filtering out unwanted network traffic coming into or going from the secured network. The filtering decision is taken according to a set of ordered filtering rules defined based on predefined security policy requirements [4].

Although deployment of firewall technology is an important step toward securing our networks, the complexity of managing firewall policy might limit the effectiveness of firewall security. A firewall policy may include *anomalies*, where a packet may match with two or more different filtering rules. When the filtering rules are defined, serious attention has to be given to rule relations and interactions in order to determine the proper rule ordering and guarantee correct security policy semantics. As the number of filtering rules increases, the difficulty of writing a new rule or modifying an existing one also increases. It is very likely, in this case, to introduce conflicting rules such as one general rule shadowing another specific rule, or correlated rules whose relative ordering determines different actions for the same packet. In addition, a typical large-scale enterprize network might involve hundreds of rules that might be written by different administrators in various times. This significantly increases the potential of anomaly occurrence in the firewall policy, jeopardizing the security of the protected network.

Therefore, the effectiveness of firewall security is dependent on providing policy management techniques and tools that enable network administrators to analyze, purify and verify the correctness of written firewall legacy rules. In this paper, we define a formal model for firewall rule relations and their filtering representation. The proposed model is simple and visually comprehensible. We use this model to develop an anomaly discovery algorithm to report any anomaly that may exist among the filtering rules. We finally develop an anomaly-free firewall rule editor, which greatly simplifies adding, removing and modifying rules into firewall policy. We used the Java programming language to implement these algorithms in one graphical user-interface tool called the "Firewall Policy Advisor."

Although firewall security has been given strong attention in the research community, the emphasis was mostly on the filtering performance and hardware support issues [5, 8, 10, 11, 17]. On the other hand, few related work [6, 10] present a resolution for the correlation conflict problem only. Other approaches [2, 9, 12, 14, 18] propose using a high-level policy language to define and analyze firewall policies and then map this language to filtering rules. Firewall query-based languages based on filtering rules are also proposed in [7, 11]. So in general, we consider our work a new progress in this area because it offers new techniques for complete anomaly discovery and rule editing that can be applied on legacy firewall policies of low-level filtering rule representation.

This paper is organized as follows. In Section 2, we give an introduction to firewall operation and filtering rule format. In Section 3, we formally define filtering rule relations, and we present our proposed model of filtering rule relations and the policy tree representation. In Section 4, we classify and define firewall policy anomalies, and then we describe the anomaly discovery algorithm and implementation. In Section 5, we present the design and implementation of anomaly-free firewall rule editor. In Section 6, we give a summary of related work. Finally, in Section 7, we show our conclusions and our future work plan.

2. Firewall Background

A firewall is a network element that controls the traversal of packets across the boundaries of a secured network based on a specific security policy. A firewall security policy is a list of ordered filtering rules that define the actions performed on matching packets. A rule is composed of filtering fields (also called network fields) such as protocol type, source IP address, destination IP address, source port and destination port, and an action field. Each network field could be a single value or range of values. Filtering actions are either to *accept*, which passes the packet into or from the secure network, or to *deny*, which causes the packet to be discarded. The packet is accepted or denied by a specific rule if the packet header information matches all the network fields of this rule. Otherwise, the following rule is examined and the process is repeated until a matching rule is found or the default policy action is performed [3]. In this paper, we assume a "deny" default policy action.

order	protocol	src_ip	src_port	dst_ip	dst_port	action
1:	tcp,	140.192.37.20,	any,	*.*.*.*,	80,	deny
2:	tcp,	140.192.37.*,	any,	*.*.*.*,	80,	accept
3:	tcp,	*.*.*.*,	any,	161.120.33.40,	80,	accept
4:	tcp,	140.192.37.*,	any,	161.120.33.40,	80,	deny
5:	tcp,	140.192.37.30,	any,	*.*.*.*,	21,	deny
6:	tcp,	140.192.37.*,	any,	*.*.*.*,	21,	accept
7:	tcp,	140.192.37.*,	any,	161.120.33.40,	21,	accept
8:	tcp,	*.*.*.*,	any,	*.*.*.*,	any,	deny
9:	udp,	140.192.37.*,	any,	161.120.33.40,	53,	accept
10:	udp,	*.*.*.*,	any,	161.120.33.40,	53,	accept
11:	udp,	*.*.*.*,	any,	*.*.*.*,	any,	deny

Figure 1. A firewall policy example.

Filtering Rule Format It is possible to use any field in IP, UDP or TCP headers in the rule filtering part, however, practical experience shows that the most commonly used matching fields are: protocol type, source IP address, source port, destination IP address and destination port. Some other fields, like TTL and TCP flags, are occasionally used for specific filtering purposes [5]. The following is the common format of packet filtering rules in a firewall policy:

```
<order> <protocol><src_ip><src_port><dst_ip><dst_port> <action>
```

In this paper, we refer to the network fields as the "5-tuple filter." The order of the rule determines its position relative to other filtering rules. IP addresses can be a host (e.g. 140.192.37.120), or a network address (e.g. 140.192.37.*). Ports can be either a single specific port number, or any port number indicated by "any." Some firewall implementations allow the usage of non-wildcard ranges in specifying source and destination addresses or ports. However, it is always possible to split a filtering rule with a multi-value field into several rules each with a single-value field [15]. An example of typical firewall rules is shown in Figure 1.

3. Firewall Policy Modelling

As a basic requirement for any firewall policy management solution, we first model the relations between the rules in a firewall policy. Rule relation modelling is necessary for analyzing the firewall policy and designing management techniques such as anomaly discovery and policy editing. In this section, we formally describe our model of firewall rule relations.

3.1 Formalization of Firewall Rule Relations

To be able to build a useful model for filtering rules, we need to determine all the relations that may relate two or more packet filters. In this section we define all the possible relations that may exist between filtering rules, and we show that there is no other relation exists. We determine the relations based on comparing the network fields of filtering rules as follows.

DEFINITION 1 *Rules R_x and R_y are completely disjoint if every field in R_x is not a subset and not a superset and not equal to the corresponding field in R_y.*

Formally, R_x and R_y are completely disjoint iff
$\forall i : R_x[i] \not\bowtie R_y[i]$
where $\bowtie \in \{\subset, \supset, =\}, i \in \{\texttt{protocol}, \texttt{src\_ip}, \texttt{src\_port}, \texttt{dst\_ip}, \texttt{dst\_port}\}$

DEFINITION 2 *Rules R_x and R_y are exactly matched if every field in R_x is equal to the corresponding field in R_y.*

Formally, R_x exactly matches R_y iff
$\forall i : R_x[i] = R_y[i]$ where $i \in \{\texttt{protocol}, \texttt{src\_ip}, \texttt{src\_port}, \texttt{dst\_ip}, \texttt{dst\_port}\}$

DEFINITION 3 *Rules R_x and R_y are inclusively matched if they do not exactly match and if every field in R_x is a subset or equal to the corresponding field in R_y. R_x is called the subset match while R_y is called the superset match.*

Formally, R_x inclusively matches R_y iff
$\forall i : R_x[i] \subseteq R_y[i]$ and $\exists j$ such that : $R_x[j] \neq R_y[j]$
where $i, j \in \{\texttt{protocol}, \texttt{src\_ip}, \texttt{src\_port}, \texttt{dst\_ip}, \texttt{dst\_port}\}$

For example, rule 1 inclusively matches rule 2 in Figure 1. Rule 1 is the subset match of the relation while rule 2 is the superset match.

DEFINITION 4 *Rules R_x and R_y are partially disjoint (or partially matched) if there is at least one field in R_x that is a subset or a superset or equal to the corresponding field in R_y, and there is at least one field in R_x that is not a subset and not a superset and not equal to the corresponding field in R_y.*

Formally, R_x and R_y are partially disjoint (or partially matched) iff
$\exists i, j$ such that $R_x[i] \bowtie R_y[i]$ and $R_x[j] \not\bowtie R_y[j]$ and $i \neq j$
where $\bowtie \in \{\subset, \supset, =\}, i, j \in \{\texttt{protocol}, \texttt{src\_ip}, \texttt{src\_port}, \texttt{dst\_ip}, \texttt{dst\_port}\}$

For example, rule 2 and rule 6 in Figure 1 are partially disjoint (or partially matched).

DEFINITION 5 *Rules R_x and R_y are correlated if some fields in R_x are subsets or equal to the corresponding fields in R_y, and the rest of the fields in R_x are supersets of the corresponding fields in R_y.*

Formally, R_x and R_y are correlated iff
$\forall i : R_x[i] \bowtie R_y[i]$ and
$\exists i, j$ such that : $R_x[i] \subset R_y[i]$ and $R_x[j] \supset R_y[j]$ and $i \neq j$
where $\bowtie \in \{\subset, \supset, =\}, i, j \in \{\texttt{protocol}, \texttt{src\_ip}, \texttt{src\_port}, \texttt{dst\_ip}, \texttt{dst\_port}\}$

For example, rule 1 and rule 3 in Figure 1 are correlated.

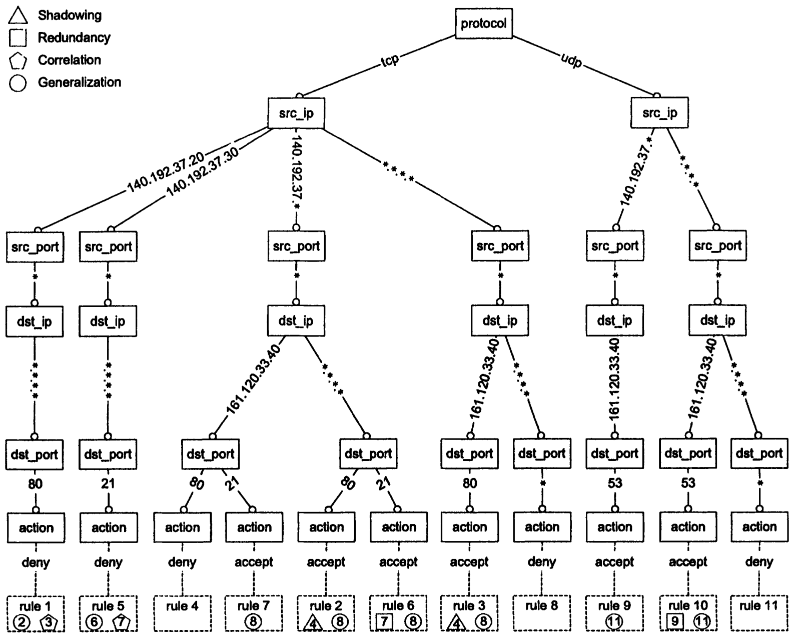


Figure 2. The policy tree for the firewall policy in Figure 1.

The following theorems show that these relations are distinct, i.e. only one relation can relate R_x and R_y, and complete, i.e. there is no other relation between R_x and R_y could exist. A complete proof of the theorems is presented in [1].

THEOREM 1 *The relations defined above are distinct; i.e. any two k-tuple filters in a firewall policy are related by only one of the defined relations.*

THEOREM 2 *The union of these relations represents the universal set of relations between any two k-tuple filters in a firewall policy.*

3.2 Firewall Policy Representation

We represent the firewall policy by a single rooted tree that we name the *policy tree*. The tree model provides a simple and apprehensible representation of the filtering rules and at the same time allows for easy discovery of relations and anomalies among the rules. Each node in a policy tree represents a field of the filtering rule, and each branch at this node represents a possible value of the associated field. The root node of the policy tree represents the protocol field, and the leaf nodes represent the action field, intermediate nodes represent other 5-tuple filter fields in order. Every tree path starting at the root and ending at a leaf represents a rule in the policy and vice versa. Rules that have the same field value at a specific node, will share the same branch representing that value.

Figure 2 illustrates the policy tree model of the security policy in Figure 1. Notice that every rule should have an action leaf in the tree. The dotted box below each leaf indicates the rule represented by that branch in addition to other rules that are in anomaly with it as described in the following section. The tree shows that rules 1 and 5 each has a separate source address branch as they have different field values, whereas rules 2, 4, 6 and 7 share the same source address branch as they all have the same field value. Also notice that rule 8 has a separate branch and also appears on other rule branches of which it is a superset, while rule 4 has a separate branch and also appears on other rule branches of which it is a subset.

The basic idea for building the policy tree is to insert the filtering rule in the correct tree path. When a rule field is inserted at any tree node, the rule branch is determined based on matching the field value with the existing branches. If a branch exactly matches the field value, the rule is inserted in this branch, otherwise a new branch is created. The rule also propagates in superset or superset branches to preserve the relations between the policy rules.

4. Firewall Policy Anomaly Discovery

The ordering of filtering rules in a firewall policy is very crucial in determining the security policy because the firewall packet filtering process is performed by sequentially matching the packet against filtering rules until a match is found. If filtering rules are completely disjoint, the ordering of the rules is insignificant. However, it is very common to have filtering rules that are inter-related. In this case, if the relative rule ordering is not carefully assigned, some rules may be always screened by other rules producing an incorrect security policy. Moreover, when a security policy contains a large number of filtering rules, the possibility of writing conflicting or redundant rules is relatively high. A firewall policy anomaly is defined as the existence of two or more different filtering rules that match the same packet. In this section, we classify different anomalies that may exist among filtering rules and then describe a technique for discovering these anomalies.

4.1 Firewall Policy Anomaly Classification

Here, we describe and then define a number of possible firewall policy anomalies. These include errors for definite conflicts that cause some rules to be always suppressed by other rules, or warnings for potential conflicts that may be implied in related rules.

1. **Shadowing anomaly** A rule is shadowed when a previous rule matches all the packets that match this rule, such that the shadowed rule will never be activated. Rule R_y is shadowed by rule R_x if R_y follows R_x in the order, and R_y is a subset match of R_x, and the actions of R_x and R_y are different. As illustrated in the rules in Figure 1, rule 4 is a subset match of rule 3 with a different action. We say that rule 4 is shadowed by rule 3 as rule 4 will never get activated.

   Shadowing is a critical error in the policy, as the shadowed rule never takes effect. This might cause a permitted traffic to be blocked and vice versa. It is important to discover shadowed rules and alert the administrator who might correct this error by reordering or removing the shadowed rule.

2. **Correlation anomaly** Two rules are correlated if the first rule in order matches some packets that match the second rule and the second rule matches some packets that match the first rule. Rule R_x and rule R_y have a correlation anomaly if R_x and R_y are correlated, and the actions of R_x and R_y are different. As illustrated in the rules in Figure 1, rule 1 is in correlation with rule 3; if the order of the two rules is reversed, the effect of the resulting policy will be different.

   Correlation is considered an anomaly warning because the correlated rules imply an action that is not explicitly handled by the filtering rules. Consider rules 1 and 3 in Figure 1. The two rules with this ordering imply that all HTTP traffic coming from address 140.192.37.20 and going to address 161.120.33.40 is denied. However, if their order is reversed, the same traffic will be accepted. Therefore, in order to resolve this conflict, we point out the correlation between the rules and prompt the user to choose the proper order that complies with the security policy requirements.

3. **Generalization anomaly** A rule is a generalization of another rule if this general rule can match all the packets that match a specific rule that precedes it. Rule R_y is a generalization of rule R_x if R_y follows R_x in the order, and R_y is a superset match of R_x, and the actions of R_y and R_x are different. As illustrated in the rules in Figure 1, rule 2 is a generalization of rule 1; if the order of the two rules is reversed, the effect of the resulting policy will be changed, and rule 1 will not be effective anymore, as it will be shadowed by rule 2. Therefore, as a general guideline, if there is an inclusive match relationship between two rules, the superset (or general) rule should come after the subset (or specific) rule.

   Generalization is considered only an anomaly warning because the specific rule makes an exception of the general rule, and thus it is important to highlight its action to the administrator for confirmation.

4. **Redundancy anomaly** A redundant rule performs the same action on the same packets as another rule such that if the redundant rule is removed, the security policy will not be affected. Rule R_y is redundant to rule R_x if R_x precedes R_y in the order, and R_y is a subset or exact match of R_x, and the actions of R_x and R_y are similar. If R_x precedes R_y in the order, and R_x is a subset match of R_y, and the actions of R_x and R_y are similar, then Rule R_x is redundant to rule R_y provided that R_x is not involved in any generalization or correlation anomalies with other rules preceding R_y. As illustrated in the rules in Figure 1, rule 7 is redundant to rule 6, and rule 9 is redundant to rule 10, so if rule 7 and rule 9 are removed, the effect of the resulting policy will not be changed.

   Redundancy is considered an error. A redundant rule may not contribute in making the filtering decision, however, it adds to the size of the filtering rule table, and might increase the search time and space requirements. It is important to discover redundant rules so that the administrator may modify its filtering action or remove it altogether.

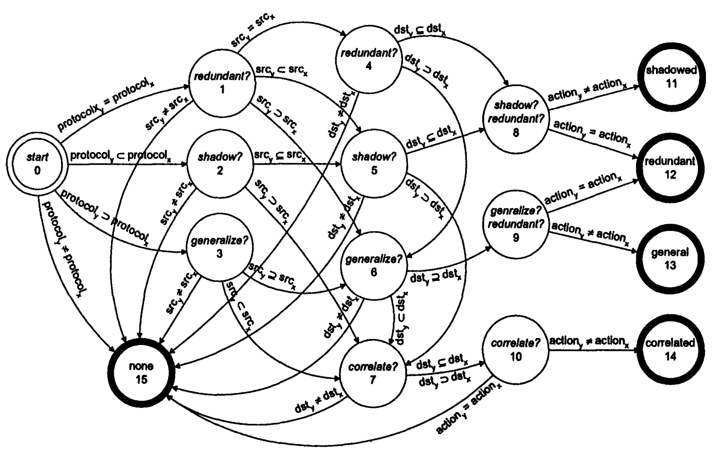


Figure 3. State diagram for detecting anomalies for rules R_x and R_y, R_y comes after R_x.

4.2 Anomaly Discovery Algorithm

The state diagram shown in Figure 3 summarizes anomaly discovery for any two rules, R_x and R_y where R_y comes after R_x in the order. For simplicity, the source address and source port and integrated into one field, and the same with the destination address and port. This simplification reduces the number of states and simplifies the explanation of the diagram. A similar state diagram can be produced for the real case of five fields with a substantially larger number of states involved.

Initially no relationship is assumed. Each field in R_y is compared to the corresponding field in R_x starting with the protocol then source address and port, and finally destination address and port. The relationship between the two rules is determined based on the result of subsequent comparisons. If every field of R_y is a subset or equal to the corresponding field in R_x and both rules have the same action, R_y is redundant to R_x, while if the actions are different, R_y is shadowed by R_x. If every field of R_y is a superset or equal to the corresponding field in R_x and both rules have the same action, R_x is potentially redundant to R_y, while if the actions are different, R_y is a generalization of R_x. If some fields of R_x are subsets or equal to the corresponding fields in R_y, and some fields of R_x are supersets to the corresponding fields in R_y, and their actions are different, then R_x is in correlation with R_y. If none of the preceding cases occur, the two rules do not involve any anomalies.

The basic idea for discovering anomalies is by determining if two rules coincide in their policy tree paths. If the tree path of a rule coincides with the tree path of another rule, there is a potential anomaly that can be determined based on the previous definitions of anomalies. If rule paths do not coincide, these rules are disjoint and they have no anomalies. The algorithm for building the policy tree and determining the anomalies among the filtering rules is shown in Figures 4 and 5. The algorithm is divided into two main parts: an anomaly discovery routine, `DiscoverAnomaly`, which represents the transition states in the state diagram, and an anomaly decision routine, `DecideAnomaly`, which represents the termination states.

```
function DiscoverAnomaly(rule, field, node, anomaly_state)
  if field = ACTION then
    value_found = FALSE
    for each branch in node.branch_list do
      if branch.value = rule.field.value then
        value_found = TRUE
        if anomaly_state = NOANOMALY then anomaly_state = REDUNDANT
        DiscoverAnomaly(rule, field.next, branch.node, anomaly_state)
      else if rule.field.value is superset of branch.value then
        if anomaly_state = GENERALIZATION then
          DiscoverAnomaly(rule, field.next, branch.node, CORRELATION)
        else
          DiscoverAnomaly(rule, field.next, branch.node, SHADOWING)
      else if rule.field.value is subset of branch.value then
        if anomaly_state = SHADOWING then
          DiscoverAnomaly(rule, field.next, branch.node, CORRELATION)
        else
          DiscoverAnomaly(rule, field.next, branch.node, GENERALIZATION)
      end if
    end for
    if value_found = FALSE then
      new_branch = new TreeBranch(rule, rule.field, rule.field.value)
      node.branch_list.add(new_branch)
      DiscoverAnomaly(rule, field.next, new_branch.node, NOANOMALY)
    end if
  else /* action field reached */
    call DecideAnomaly(rule, field, node, anomaly_state)
  end if
end function
```

Figure 4. Algorithm for building the policy tree with anomaly discovery.

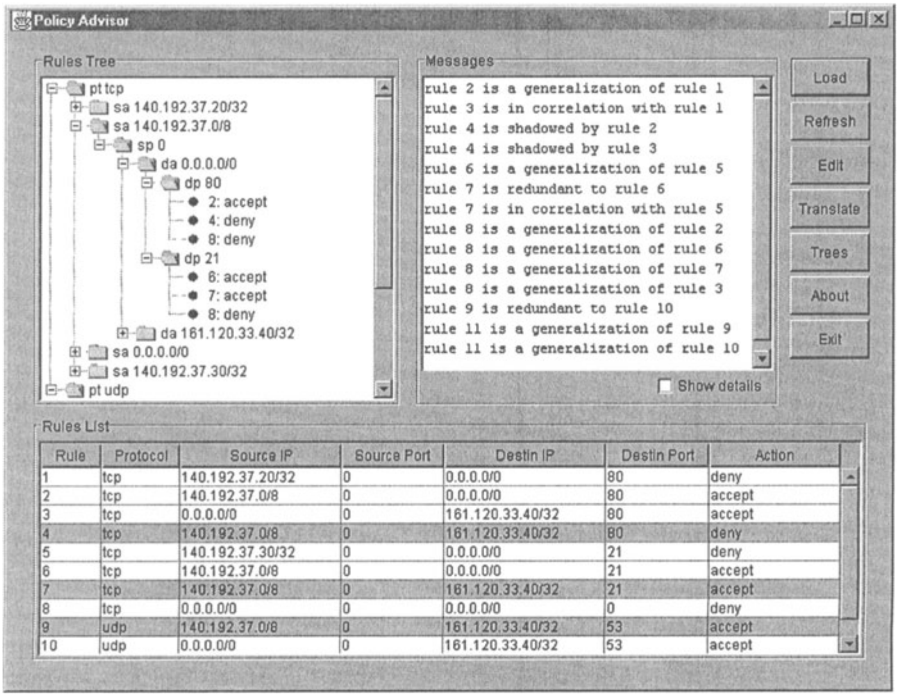
In the discovery routine, the previous anomaly state is checked if there is a value match between the field of the new rule and the already existing field branch. The next anomaly state is determined based on the shown state diagram and the algorithm is executed recursively to let the rule propagate in existing branches and check the remaining fields. As the rule propagates, the anomaly state is updated until the final state is reached. If there is no exact match for the value of a field, a new branch is created at the current node to represent the inserted field value, and the anomaly state is initialized to no anomaly. The decision routine is activated once all the rule fields have been inserted in the tree and the action field is reached. If the rule action coincides with the action of another rule, an anomaly is discovered. At that point the final anomaly state is determined and reported. If an anomaly is discovered and decided, the user is reported with the type of anomaly and the rules involved.

Applying the algorithm on the rules in Figure 1, the discovered anomalies are marked in the dotted boxes at the bottom of the policy tree in Figure 2. Shadowed rules are marked with a triangle, redundant rules with a square, correlated rules with a pentagon and generalization rules with a circle.

Figure 6 shows the graphical user interface for the Firewall Policy Advisor. The bottom panel shows a tabular list of filtering rules. The top-left panel displays the policy tree showing aggregated rules. The top-right panel displays the anomalies discovered along with highlighting redundant and shadowed rules in a different color.

```
function DecideAnomaly(rule, field, node, anomaly)
  if node has branch_list then
    branch = node.branch_list.first()
    if anomaly = CORRELATION then
      if not rule.action = branch.value then
        branch.rule.anomaly = CORRELATION
        report rule rule.id is in correlation with rule branch.rule.id
      else anomaly = NONE
    else if anomaly = GENERALIZATION and not rule.action = branch.value then
      branch.rule.anomaly = SPECIALIZATION
      report rule rule.id is a generalization of rule branch.rule.id
    else if anomaly = GENERALIZATION and rule.action = branch.value then
      if branch.rule.anomaly = NONE then
        anomaly = NONE; branch.rule.anomaly = REDUNDANCY
        report rule branch.rule.id is redundant to rule rule.id
      end if
    else if rule.action = branch.value then
      anomaly = REDUNDANCY
      report rule rule.id is redundant to rule branch.rule.id
    else if not rule.action = branch.value then
      anomaly = SHADOWING
      report rule rule.id is shadowed by rule branch.rule.id
    end if
  end if
  rule.anomaly = anomaly
end function
```

Figure 5. Algorithm for making the anomaly decision.



```
pt tcp
  sa 140.192.37.20/32
  sa 140.192.37.0/8
    sp 0
      da 0.0.0.0/0
        dp 80
          2: accept
          4: deny
          8: deny
        dp 21
          6: accept
          7: accept
          8: deny
      da 161.120.33.40/32
  sa 0.0.0.0/0
  sa 140.192.37.30/32
pt udp
```

```
rule 2 is a generalization of rule 1
rule 3 is in correlation with rule 1
rule 4 is shadowed by rule 2
rule 4 is shadowed by rule 3
rule 6 is a generalization of rule 5
rule 7 is redundant to rule 6
rule 7 is in correlation with rule 5
rule 8 is a generalization of rule 2
rule 8 is a generalization of rule 6
rule 8 is a generalization of rule 7
rule 8 is a generalization of rule 3
rule 9 is redundant to rule 10
rule 11 is a generalization of rule 9
rule 11 is a generalization of rule 10
```

Rule	Protocol	Source IP	Source Port	Destin IP	Destin Port	Action
1	tcp	140.192.37.20/32	0	0.0.0.0/0	80	deny
2	tcp	140.192.37.0/8	0	0.0.0.0/0	80	accept
3	tcp	0.0.0.0/0	0	161.120.33.40/32	80	accept
4	tcp	140.192.37.0/8	0	161.120.33.40/32	80	deny
5	tcp	140.192.37.30/32	0	0.0.0.0/0	21	deny
6	tcp	140.192.37.0/8	0	0.0.0.0/0	21	accept
7	tcp	140.192.37.0/8	0	161.120.33.40/32	21	accept
8	tcp	0.0.0.0/0	0	0.0.0.0/0	0	deny
9	udp	140.192.37.0/8	0	161.120.33.40/32	53	accept
10	udp	0.0.0.0/0	0	161.120.33.40/32	53	accept

Figure 6. Policy Advisor anomaly discovery user interface.

5. Firewall Policy Editor

Firewall policies are often written by different network administrators and occasionally updated (by inserting, modifying or removing rules) to accommodate new security requirements and network topology changes. Editing a security policy can be far more difficult than creating a new one. As rules in firewall policy are ordered, a new rule must be inserted in a particular order to avoid creating anomalies. The same applies if any network field in a rule is modified. In this section, we present a policy editor tool that simplifies the rule editing task significantly, and avoids introducing anomalies due to policy updates. The policy editor (1) prompts the user with the proper position(s) for a new or modified rule, (2) shows the changes in the security policy semantic before and after removing a rule, and (3) provides visual aids for users to track and verify policy changes. Using the policy editor, administrators require no prior knowledge or understating of the firewall policy in order to insert, modify or remove a rule.

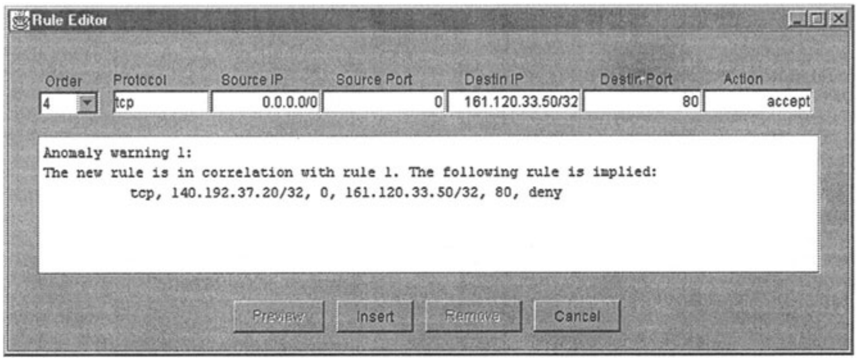
5.1 Rule Insertion

Since the ordering of rules in the filtering rule list directly impacts the semantics of the firewall security policy, a new rule must be inserted in the proper order in the policy such that no shadowing or redundancy is created. The policy editor helps the user to determine the correct position(s) of the new rule to be inserted. It also identifies anomalies that may occur due to improper insertion of the new rule.

The general idea is that the order of a new rule is determined based on its relation with other existing rules in the firewall policy. In general, a new rule should be inserted before any rule that is its superset match, and after any rule that is its subset match. The policy tree is used to keep track of the correct order of the new rule, and discover any potential anomalies. The algorithm implementing the mechanism to insert a new rule is fully described in [1].

The algorithm is organized into two phases: the browsing phase and the insertion phase. In the browsing phase, the fields of the new rule are compared with the corresponding tree branch values one at a time. If the field value of the new rule is a subset of an existing branch, then the new rule must be inserted before the minimum order of all the rules in this branch. If the field value is a superset of an existing branch, the rule must be inserted after the maximum order of all the rules in this branch. In addition, if the field value is an exact match or a subset match of a branch, evaluating the next field continues recursively by browsing through the branch sub-tree until correct position of the rule within the sub-tree is determined. Otherwise, if disjoint or superset match occurs, a branch is created for the new rule.

The algorithm enters into the insertion phase when the action field of a new rule is to be inserted. If an action branch is created for the new rule, then the rule will be inserted and assigned the order determined in the browsing phase. If there is more than one possible order for this rule, the user is asked to select an order from within a valid range of orders as determined in the browsing phase. However, if the order state of the new rule remains undetermined then policy editor rejects this new rule and prompts the user with the appropriate message. If the rule is inserted, the anomaly discovery algorithm is invoked to alert the administrators with any generalization or correlation cases as a possible source of anomalies in the firewall policy.



Order	Protocol	Source IP	Source Port	Destin IP	Destin Port	Action
4	tcp	0.0.0.0/0	0	161.120.33.50/32	80	accept

```
Anomaly warning 1:
The new rule is in correlation with rule 1. The following rule is implied:
        tcp, 140.192.37.20/32, 0, 161.120.33.50/32, 80, deny
```

Figure 7. Rule editor user interface.

5.2 Rule Removal and Modification

In generɐl, removing a rule has much less impact on the firewall policy than insertion. A removed rule does not introduce an anomaly but it might change the policy semantics and this change should be highlighted and confirmed. To remove a rule, the user enters the rule number to retrieve the rule from the rule list and selects to remove it. To preview the effect of rule removal, the policy editor gives a textual translation of the affected portion of the policy before and after the rule is removed. The user is able to compare and inspect the policy semantics before and after removal, and re-assure correctness of the policy changes. Modifying a rule in a firewall policy is also a critical operation. However, this editing action can be easily managed as rule removal and insertion as described before.

Figure 7 shows the graphical user interface for the rule editor tool. The figure shows the final step in inserting a rule in the filtering rule table. The tool alerts the user for any anomalies that may be introduced by inserting the new rule.

6. Related Work

A significant amount of work has been reported in the area of firewall and policy-based security management. In this section, we focus our study on related work that intersects with our work in three areas: packet filter modelling, conflict discovery and rule analysis.

Several models have been proposed for filtering rules. Ordered binary decision diagram is used as a model for optimizing packet classification in [11]. Another model using tuple space is developed in [16], which combines a set of filters in one tuple and stored in a hash table. The model in [17] uses bucket filters indexed by search trees. Multi-dimensional binary tries are also used to model filters [15]. In [6] a geometric model is used to represent 2-tuple filtering rules. Because these models were designed particularly to optimize packet classification in high-speed networks, we found them too complex to use for firewall policy analysis. We can confirm from experience that the tree-based model is simple and powerful enough for this purpose.

Research in policy conflict analysis has been actively growing for many years. However, most of the work in this area addresses general management policies rather than firewall-specific policies. For example, authors in [13] classify possible policy conflicts in role-based management frameworks, and develop techniques to discover them. A policy conflict scheme for IPSec is presented in [8]. Although this work is very useful as a general background, it is not directly applicable in firewall anomaly discovery. On the other hand, few research projects address the conflict problem in filtering rules. Both [6] and [10] provide algorithms for detecting and resolving conflicts among general packet filters. However, they only detect what we defined as correlation anomaly because it causes ambiguity in packet classifiers. In conclusion, we could not find any published research work that uses low-level filtering rules to perform a complete anomaly analysis and guided editing of firewall policies.

7. Conclusions and Future Work

Firewall security, like any other technology, requires proper management to provide the proper security service. Thus, just having a firewall on the boundary of a network may not necessarily make the network any secure. One reason of this is the complexity of managing firewall rules and the potential network vulnerability due to rule conflicts. The Firewall Policy Advisor presented in this paper provides a number of user-friendly tools for purifying and protecting the firewall policy from anomalies. The administrator can use the firewall policy advisor to manage a general firewall security policy without prior analysis of filtering rules. In this work, we formally defined all possible firewall rule relations and we used this to classify firewall policy anomalies. We then model the firewall rule information and relations in a tree-based representation. Based on this model and formalization, the firewall policy advisor implements two management tools:

- **Policy Anomaly Detector** for identifying conflicting, shadowing, correlated and redundant rules. When a rule anomaly is detected, users are prompted with proper corrective actions. We intentionally made the tool not to automatically correct the discovered anomaly but rather alarm the user because we believe that the administrator is the one who should do the policy changes.

- **Policy Editor** for facilitating rules insertion, modification and deletion. The policy editor automatically determines the proper order for any inserted or modified rule. It also gives a preview of the changed parts of the policy whenever a rule is removed to show the affect on the policy before and after the removal.

The firewall policy advisor is shown to be very useful and effective when used on real firewall rules in different academic and industrial environments [1]. However, we believe that there is more to do in firewall policy management area. Our future research plan includes extending the proposed techniques to handle distributed firewall policies with centralized or distributed repositories, classifying different semantics in firewall policies and extracting them from the filtering rules, translating low-level filtering rules into high-level textual description, providing a query-based policy analysis algorithms to enhance our visualization of the underlying firewall security policy.

Acknowledgments

We gratefully thank Iyad Kanj for his feedback on the theory work in this paper. We would also like to thank Lopamudra Roychoudhuri and Yongning Tang for their useful comments on an earlier version of this paper.

References

[1] E. Al-Shaer and H. Hamed. "Design and Implementation of Firewall Policy Advisor Tools." *Technical Report CTI-techrep0801*, School of Computer Science Telecommunications and Information Systems, DePaul University, August 2002.

[2] Y. Bartal., A. Mayer, K. Nissim and A. Wool. "Firmato: A Novel Firewall Management Toolkit." *Proceedings of 1999 IEEE Symposium on Security and Privacy*, May 1999.

[3] D. Chapman and E. Zwicky. *Building Internet Firewalls, Second Edition*, Orielly & Associates Inc., 2000.

[4] W. Cheswick and S. Belovin. *Firewalls and Internet Security*, Addison-Wesley, 1995.

[5] S. Cobb. "ICSA Firewall Policy Guide v2.0." NCSA Security White Paper Series, 1997.

[6] D. Eppstein and S. Muthukrishnan. "Internet Packet Filter Management and Rectangle Geometry." *Proceedings of 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, January 2001.

[7] P. Eronen and J. Zitting. "An Expert System for Analyzing Firewall Rules." *Proceedings of 6th Nordic Workshop on Secure IT-Systems (NordSec 2001)*, November 2001.

[8] Z. Fu, F. Wu, H. Huang, K. Loh, F. Gong, I. Baldine and C. Xu. "IPSec/VPN Security Policy: Correctness, Conflict Detection and Resolution." *Proceedings of Policy'2001 Workshop*, January 2001.

[9] J. Guttman. "Filtering Posture: Local Enforcement for Global Policies." *Proceedings of 1997 IEEE Symposium on security and Privacy*, May 1997.

[10] B. Hari, S. Suri and G. Parulkar. "Detecting and Resolving Packet Filter Conflicts." *Proceedings of IEEE INFOCOM'00*, March 2000.

[11] S. Hazelhusrt. "Algorithms for Analyzing Firewall and Router Access Lists." *Technical Report TR-WitsCS-1999*, Department of Computer Science, University of the Witwatersrand, South Africa, July 1999.

[12] S. Hinrichs. "Policy-Based Management: Bridging the Gap." *Proceedings of 15th Annual Computer Security Applications Conference (ACSAC'99)*, December 1999.

[13] E. Lupu and M. Sloman. "Conflict Analysis for Management Policies." *In Proceedings of IFIP/IEEE International Symposium on Integrated Network Management (IM'1997)*, May 1997.

[14] A. Mayer, A. Wool and E. Ziskind. "Fang: A Firewall Analysis Engine." *Proceedings of 2000 IEEE Symposium on Security and Privacy*, May 2000.

[15] L. Qiu, G. Varghese, and S. Suri. "Fast Firewall Implementations for Software and Hardware-based Routers." *Proceedings of 9th International Conference on Network Protocols (ICNP'2001)*, November 2001.

[16] V. Srinivasan, S. Suri and G. Varghese. "Packet Classification Using Tuple Space Search." *Computer ACM SIGCOMM Communication Review*, October 1999.

[17] T. Woo. "A Modular Approach to Packet Classification: Algorithms and Results." *Proceedings of IEEE INFOCOM'00*, March 2000.

[18] A. Wool. "Architecting the Lumeta Firewall Analyzer." *Proceedings of 10th USENIX Security Symposium*, August 2001.

[19] "Cisco Secure Policy Manager 2.3 Data Sheet."
http://www.cisco.com/warp/public/cc/pd/sqsw/sqppmn/prodlit/spmgr_ds.pdf

[20] "Check Point Visual Policy Editor Data Sheet."
http://www.checkpoint.com/products/downloads/vpe_datasheet.pdf

A SCALED, IMMUNOLOGICAL APPROACH TO ANOMALY COUNTERMEASURES

Combining pH with cfengine

Kyrre M. Begnum
Faculty of Engineering, Oslo University College, Norway
Kyrre.Begnum@iu.hio.no

Mark Burgess
Faculty of Engineering, Oslo University College, Norway
Mark.Burgess@iu.hio.no

Abstract: We discuss the combination of two anomaly detection models, the Linux kernel module pH and cfengine, in order to create a multi-scaled approach to computer anomaly detection with automated response. By examining the time-average data from pH, we find the two systems to be conceptually complementary and to have compatible data models. Based on these findings, we build a simple prototype system and comment on how the same model could be extended to include other anomaly detection mechanisms.

1. Introduction

Computer Immunology is an approach to integrity management, based on the notion that computer systems are healthy when their behaviour is free of anomalous occurrences[16, 3]. The onus falls on researchers to define what 'anomaly free' means, or conversely what is normal for a system. This can be done in several ways.

Commonly one supposes that systems are normal when they exhibit medium term stability, i.e. stability on a time scale at which users experience the system[4]. Health or stability is thus related to ones idea of *policy.* Long term changes, such as policy revisions, can occur and short term changes are occurring all the time. Normality is a statistical concept, which accrues over time, and computer immunology is a form of computer learning[6, 12]. Unlike many other methods of artificial intelligence, computer immunology is about purely mechanistic regulation of behaviour, rather devoid of 'intelligence' in the normal sense of the word. It concerns prescriptions for recognition of change with computer systems. In summary, this medium term stability is achieved with the following strategy:

- The system should be tolerant of anything determined by policy.
- Policy is partly specified and partly learned from experience.
- The system should react to abnormal or non-policy-conformant events in order to restore normality.

Two approaches have emerged for addressing these issues at different scales.

- At the University of New Mexico, the Computer Immunology group has examined strategies for detecting signatures of abnormal computer behaviour at kernel level. Their pH system[10, 14] learns new signatures over time, but is resistant to doing so. The primary motivation here has been in deflecting network intrusions, though the method is equally effective in detecting abnormal local usage, such as attempts to exploit buffer overflows. The response provoked by anomalies has been in the form of scheduling delays in processes with unknown call sequences, in order to urge attackers to lose interest.
- At Oslo University College we have focused on the configuration management aspect of policy, using a system of agents (cfengine) that detect and use their environmental conditions and current configuration to detect anomalous changes[1]. Again, the policy is partly specified and partly learned from patterns of usage, and the response to different events is specified itself as a matter of policy, and the agents ensure that the system tends to maintain the same state over time.

This paper describes the process of combining cfengine, a high level configuration engine with pH, a kernel patch which enables anomaly detection and reaction on a per process basis. The project has two independent goals: to provide a better anomaly detection capability for cfengine, and a better response engine for pH; to create a versatile framework for the collection of system related data for further research into anomaly detection. There is thus a security motivation and a research motivation. The 'science' of anomaly detection is still in its infancy, thus the latter should not be neglected for the sake of building a quick fix.

The plan for this paper is as follows. We begin by discussing the requirements for compatibility between pH and cfengine, as well as what we hope to achieve by combining them. In sections 3 and 4 we provide some details about these two systems in order to contextualize their marriage. Finally, we provide a cooperative model for these systems and discuss further extensions for future work.

2. Compatibility

On the surface, it would seem that pH and cfengine are two very different systems, with somewhat different goals. How then are they to be meaningfully combined?

The common thread between the systems is their long term goal: that of system regulation, or *homeostasis* (state regulation)[2, 3, 15, 6]. When a change occurs in a system, there are two general ways that it can respond. With negative feedback, the system responds in such a way as to reverse the direction of change; this tends to keep things constant and allows us to maintain a regular state; positive feedback would tend to amplify the change. This has a de-stabilizing effect, so it does not result in homeostasis. While it can be useful for rapid responses, it is a dangerous strategy, since the change will tend to dominate and eventually consume a system. Regulation, or homeostasis, is thus a self-regulating mechanism that allows a system to avoid paying detailed attention to its most basic functions thereby helping keep it in a steady state

The University of New Mexico's pH kernel modification stands for process homeostasis. Its goal is to seek a 'steady state' list of tasks that are undertaken by a computer system (i.e. a normalized list). It detects previously unknown tasks and offers resistance to their execution. If the new tasks persist, they are eventually tolerated by the system.

Cfengine, on the other hand, seeks to maintain a 'steady state' *configuration* of a system, where configuration means the state of the file system, process table and service ports. It detects and opposes changes by two strategies: i) by referring to a descriptive policy about what is considered acceptable, and ii) by monitoring key system resource usage over days and weeks, and responding to statistical irregularities. Thus, both pH and cfengine have a policy of maintaining a 'normal' or 'regular' state, and both are able to learn about long term changes by adapting their reference state. Their key difference is the scale at which they operate: pH works at the microscopic, short-term level of system calls, while cfengine works at medium term user time-scales.

How can these be combined? As we have already stated, an adaptive, learning system is necessarily a statistical system; we should therefore ask: i) Do they have compatible data models? ii) What are the tolerances of the systems? i.e. with what accuracy can they make claims about system normality; hence, when is it appropriate to activate countermeasures?

iii) Resource utilization is known to be a strongly social phenomenon, with a marked variation over the working week. Cfengine uses the working week as a model for measuring its medium-term state. Is the same time reference appropriate for pH, which deals with short term events?

Combining these seemly disparate mechanisms is thus a scaled approach to system regulation. PH detects events on short time scale, responds simply and propagates the data forward as medium term statistics which it uses privately for future reference. Cfengine measures medium term events and activates medium to long term response strategies. Our aim here is to see whether medium term data from pH can be read and utilized by cfengine in order to bring the knowledge of short term behaviour to bear on longer term strategy.

3. Short introduction to pH

Ph is a patch for the Linux 2.2 kernel. It addresses the anomaly countermeasure problem at the level of system call sequences. A lot of security software today is designed to detect attacks (e.g. Intrusion Detection Systems (IDS)) or to find vulnerabilities in the system; only a few tries to stop attacks as they occur (e.g. some can delete viruses and even repair damaged files). Although a response capability exists in some programs, most software only issues a warning, and waits for a system administrator to react.

The key is the ability to tell the difference between "benign" and "hostile". For pH, like most other IDS, *normal* is benign, and *abnormal* is hostile. By analysing the pattern of system-calls made by each process using pattern-matching algorithms, pH gains knowledge about what it perceives as normal behaviour. It also maintains a profile of each binary as to see if each process produces the expected patterns of system-calls. As long as a process keeps it's number of strange patterns under a certain level, it is considered normal. If the number rises above a threshold, pH starts to sabotage the process by delaying all the system-calls made by it.

Ph keeps one profile for each binary, but it reacts to individual processes. Every process has a sequence of system calls, known as a *trace*. The profile of each binary is updated and adjusted to the behaviour of the processes. This means that instances of a specific behaviour will in time be considered normal. Not all anomalies are real threats to the system, but earlier research by UNM suggested that "To date, all of the intrusions we have studied produce anomalous sequences in temporally local clusters[10]; pH is therefore designed to react regarding the density of anomalous system call patterns.

Strategy

The algorithm used by pH is called "time-delay embedding"; it looks at the trace of each process' system calls. For each system call, pH notes the calls preceding this one within a window. This gives a number of system call pairs for each position in the window. For instance, suppose we have the following trace of an imagined process:

```
getpriority, open, write, write,
close, open, pipe, close, exit
```

We read the trace from the left. While reading, we note which calls come behind the current one. Just like sliding a window over the trace. The default window-size for pH is 6 calls. pH does not consider each sequence it encounters as part of its process profile right away. There is a distinction between the current profile (called *test*) for a given binary and the temporary profile of the running process (called *train*). The *train*-array is continuously updated with new pairs. Should a pair occur, that is *not* part of the *test*-array, then it is considered an anomaly. The test-array can only be updated by replacing it with the current train-array. This replacement occurs under one of three conditions (from the documentation): i) The user explicitly signals via special system call (`sys_pH`) that a profile's training data is valid. ii) The profile anomaly count exceeds the parameter `anomaly_limit`. iii) A specialized training formula is satisfied.

Should an anomaly occur, then a number of the following system-calls will be delayed. After a certain number of anomalies, the train-profile will switch to the test-profile. This is called *acquired tolerance*, meaning the profile adapts to the behaviour of the process. But should the anomalies occur too close to each other, then pH will react in the opposite manner and reset the train-profile.

Implementation

It is possible to control pH at runtime with a system call interface: `sys_pH()`, and `pH-admin` which is basically a front-end to the system call. This tool can, amongst other things:

- Turn the monitoring on/off
- Write profiles to disk
- Adjust pH-variables (i.e `delay_factor`)
- Force the train profile to be copied to the test profile.

pH saves information about each running process from the /proc directory. Each folder belonging to a process has a file called pH, which contains information about delay, system call count, if the profile is considered normal and if the process is currently frozen.

All the messages created by pH are logged in the log file /var/log/syslog. The profiles for all the binaries are located in the folder /var/lib/pH/profiles where they are sorted in a hierarchy which mirrors the actual file-system. Each binary is therefore identifiable by it's path. For example, the program less, which has the path /usr/bin/less, will have it's profile at
/var/lib/pH/profiles/usr/bin/less.

4. Short introduction to cfengine

Cfengine is a well-known policy based configuration management system written at Oslo University College[1], which is comprised of a number of components (see fig 1). An agent component is responsible for enforcing specified policy by comparing a description of the permissable states of a host's configuration with the host's actual state. There are also file-server and scheduler components for deploying cooperative management schemes. The cfenvd environment daemon is a component that measures system resource usage, independently of the other parts and records it in a database[5], which becomes the definition of 'normal'. This tool is intended both for regulative feedback and for gathering research data. It classifies the current state of resource usage

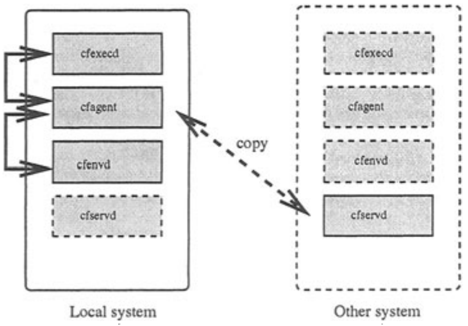


Figure 1 A schematic representation of cfengine components. The environment daemon communicates with the agent on each host, by providing it with classified state information.

in relation to what has been learned previously, using units of the statistical uncertainty (standard deviation) for each time of week. It then publishes its results for other programs to use, notably cfagent. Cfagent receives the data as a 'classified event' which can be used to predicate countermeasures or follow-up responses for the state concerned. Some examples of classes which can become active in the cfagent:

```
RootProcs_low_dev2
netbiosssn_in_low_dev2
smtp_out_high_anomalous
www_in_high_dev3
```

The first of these classes tells us that the number of root processes is two standard deviations below the average for past behaviour. This might be fortuitous, or might signify a problem, such as a crashed server; we do not know the reason, only that an anomaly has occurred. The WWW item tells us that the number of incoming connections is three standard deviations above average. The SMTP item tells us that the number of outgoing SMTP connections is more than three standard deviations (this is the defined meaning of anomalous) above average, perhaps signifying a mail flood. The setting of these classes is transparent to the user, but the additional information is only visible to the privileged owner of the cfengine work-directory, where the data are cached.

Countermeasures or follow-up actions can be attached to events in order to automate a policy decision to the occurrence. For example, one might decide to shut down an offending service temporarily, and then follow up with a file audit:

```
processes:

 smtp_out_high_anomalous::

   ''sendmail'' signal=kill

files:

 smtp_out_high_anomalous::

   /usr recurse=inf checksum=md5
```

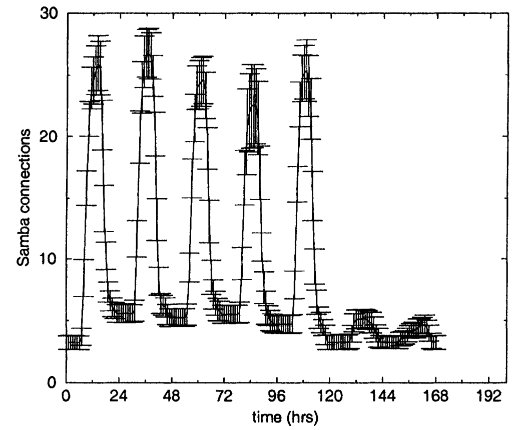


Figure 2 Cfengine measures patterns of resource usage over the working week. This example shows how measurements of Samba file sharing lead to an average picture of behaviour at different times of the week. The solid line is the average value over many weeks and error bars indicate the standard deviation.

5. A cooperative model

We wish to combine these two systems in order to create a better and comprehensible high level system that can react to the systems state. This is a topic which was not clear from the available research; therefore wish to find a framework for collecting, storing and analyzing data on their properties. A combined system has to be both reliable and secure if it is to be used on systems that do actual work. Creating a isolated system for testing makes sense for keeping the system clear from uncontrollable noise (users, network traffic and so on). But if noise is normal, and normal is what you're looking for, then the only way to test it, will be real-life.

Various models might be used for establishing a connection between cfengine and pH. The first alternative is a plug-in architecture, where pH is considered to be a cfengine plug-in module. This would facilitate a close working relationship, but it requires permanent structural modifications to both.

A second alternative, would be a model where a higher level system invokes and controls smaller components. The process monitoring would be done by a detached participant. The higher level system would act on the information delivered by the component. This information could be gathered via a spacial interface or by parsing log files.

The model we have chosen is a feedback model that preserves the domain of each software system, but allows a passive communications channel between them (see fig. 4), using files and databases. Thus pH will be able to adjust it's monitoring level depending on instruction from cfengine, and cfengine can adjust its behaviour based on results from pH. pH has it's own engine for data-analysis and cfengine analyzes the data further.

pH already has an interface that cfengine can use to control it in the form of shell commands. We could also go directly to the system call `sys_pH` instead of going via the `pH-admin` command. pH stores its information in several places: `/proc`, `/var/log/syslog` and `/var/lib/pH/profiles`. The profiles are in a self-defined binary format and will be printed to the terminal by the command `pH-print-profile`. The same holds for the sequence files, with the corresponding command `pH-print-sequences`.

In order to collect the data from pH, we use cfengine's `cfenvd` daemon and database, which in turn provides information to the agent when it activates.

In fig. 3, a number of identical trials was performed in order to simulate a long term variation of the form measured by `cfenvd` (see fig. 2). An apache web server was used as the pH monitored process. It was loaded by a number of clients in an identical pattern of variation. Repeating the same changing load five times, a pH process counted the total number of system calls. The average of the five identical trials with standard deviation shown as error-bars is plotted in the figure. Each trial measured 120 values, recorded each 30 seconds over the space of an hour. This figure is sufficient to make two points:

- The statistical model of average behaviour with certain tolerance is compatible with that currently used by `cfenvd`.
- The error bars are not zero, thus there is a natural uncertainty in the results, even with close to identical trials.

The latter point is interesting, since these additional system calls cannot be explained by other processes. Ph measures only system calls related to a specific binary. No other binaries could be responsible for this error.

The fact that there is a statistical uncertainty is very important. It means that the purely digital approach to anomaly detection is not sufficient to yield exact repeatability. Thus if one is looking for repeat-ably identifiable signatures, one must allow a margin for error. This is clearly significant for intrusion detection systems, which normally recognize only exactly learned patterns. The source of the uncertainty could lie both on the side of the server, or on the side of the clients loading the server. It could be a result of scheduling differences, since measurements are cumulative values over a 30 second period. Differences due to network traffic load can be ruled out, since the trials were performed in isolation.

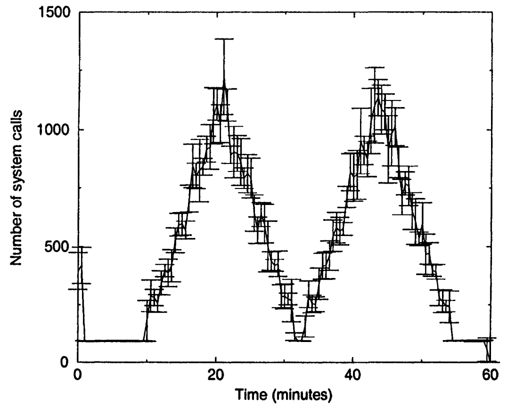


Figure 3 Repeated trials on a simulated load, showing how the average number of system calls varies in proportion to applied load, within measured tolerances. This shows that the basic cfengine statistical model applies to pH also.

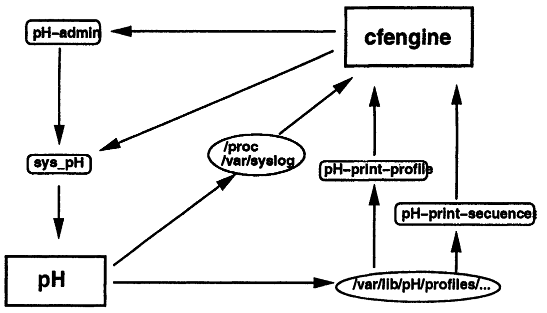


Figure 4 Information Flow Diagram: how cfengine and pH exchange information and management intructions. Communication makes use of existing operating system abstractions, like the /proc filesystem for kernel tables. Similarly, cfengine uses the standard pH API, maintaining the independence of the two systems.

Using pH to measure process load shows us one other thing, that is interesting for future work: the simple measurements show a clear pattern, i.e. the input pattern is reflected linearly, up to a standard error, in the output graph. Monitoring the number of system calls for a process over time, we can determine when it has been used, and how much. We could also build up a statistic here to gather a trend of how much a program is used, and how much we can expect it to be used. By measuring individual sequences separately, it would be possible to perform a code analysis of software, indicating how much users used different parts of the software. This is very interesting for future research.

Modifications to pH

The most important modification to pH, is having the ability to specify what processes to delay. The monitoring will still be done on all profiles, but a variable describing if this process is subject to delaying must exist for every binary. In addition, we must be able to choose if this variable should be set to *delay* or *ignore* by default on the creation of a new profile. If the default value is *delay*, then pH will work as before. The administration of these variables can be handled from cfengine. This enables us to achieve the following: i)Delay all but these binaries (Default on). ii) Ignore all but these binaries (Default off). Note, that the default value could be changed in runtime too.

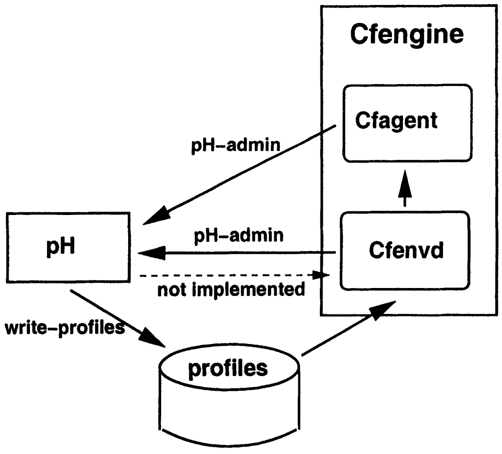


Figure 5 Architectural view of cfengine and pH's collaborative scheme. Both systems are independent of one another and communicate only minimally through profiles held in the kernel.

Data storage

The new pH-related data need to be stored, e.g. the number of abnormal processes, number of system calls for selected processes. The size of the database will vary depending on the number of profiles we wish to monitor and how long we wish to keep the data. `Cfenvd` stores only one week's worth of data, and merges the data together with a average from all other preceding weeks, by a geometric series. This approach would also be useful for data like the number of system calls for a process. It would give us enough to generate the expected usage throughout the week for a given application.

For other data, like the sum of anomalies at any given moment, this variable is a bit more tricky. This variable will be influenced by the use of new applications and has to be monitored over a longer period. Clearly, not all anomalies are genuine and the system must learn to tolerate those that are not dangerous. A one-week local average can be useful as soon as the variable is stable or else the first encounter with all applications will influence the average and deviation so much that small and potentially interesting deviations later on will be unrecognizable.

Cfengine is designed to work independently. An anomaly in the data will trigger an event in cfengine, but we are not always interested in anomalies. We need an option for getting the datasets so that we can view them in plots or analyze them statistically (see fig 5).

6. Example regulation strategy

We envisage automated responses to anomalous behaviour. Such responses have been considered before in other contexts (see refs [11, 9]). A simple example of a `cfagent` response helps in visualizing the interplay between the two anomaly systems. A special `cfagent` class is made to activate on the presence of a recent anomaly. This class persists until it has been expedited by an agent.

Note that pH does not try to start `cfagent` immediately. For one thing, pH is in kernel space, and the agent must run in user space. However, it leaves a semaphore to the cfengine scheduler to activate the agent with a special class, on its next scheduled run.

If the agent were started immediately as a direct result of the anomaly, it would be trivial to use this in a denial of service attack. Our strategy here is a scaled approach: using cfengine with its normal 'policy' level of statistical uncertainty, and leave pH itself to deter potential attacks with its delaying tactics.

Two classes can become active: a sequence anomaly semaphore, indicating that a potentially dangerous sequence of system calls was identified, and a load anomaly, indicating that `cfenvd` has found the total load being processed by pH is anomalous. We therefore cover qualitative and quantitative anomalies.

```
control:

 actionsequence = ( files processes )

files: // ph_sequence_anomaly::

 bin_bash_sequence_anomaly::

   # Do MD5 integrity check on system files, in case of intrusion
   /usr owner=root,bin checksum=md5 recurse=inf action=warnall

processes:  // ph_load_anomaly::

  bin_bash_high_dev1::

    # Kill the processes causing anomalous load, if it still exists
    ''*'' signal=kill filter=ph_load_filter
```

pH communicates its variables (the list of offending processes) to `cfagent` using one of cfengine's filter interfaces for selecting processes. pH has no functionality for killing a process itself, so this is a natural task for `cfagent` to perform, assuming the offending process is persistent over the `cfagent` scheduling interval.

7. Conclusions

We have measured the behaviour of pH and cfengine and found that they have compatible goals and data models. Cfengine's statistical tools for state analysis complement the powerful pH data-microscope. We have implemented a pilot scheme for combining them into a multi-scaled approach to anomaly detection. Our interest has been two-fold: we were keen to devise a fully autonomic response to anomalous behaviour in computer systems, and were driven to learn more about the meaning of 'normal' and 'anomalous' in the context of the human-computer interaction. We feel that we have made headway towards both of these goals in part, by showing how two such disparate mechanisms can cooperate in a scaled information model. In future work, we hope to study the behaviour of the combined system in a production setting.

How many processes have anomalies? This number would be an indicator of the stability and predictability of a host.

Analysing the behaviour of a binary over time. A comparison of profiles across different hosts could also indicate how similar the different applications are being used on the different hosts. This has Human-Computer-Interaction ramifications, and is especially interesting for complex programs, such as computer games or office applications, where perhaps only a small part of the program is actually ever used. The relevance here is thus not only system administration, but also software engineering. We could also use these data in a work-routine experiment. When are certain applications being used? Do people use more complex programs at the end of their work-day? The benefit of having the monitoring system separated from the application, is that we can gather these data for every program on the host. Eventually such data can also lead to better management policies.

On the issue of intrusion detection, there are numerous possibilities to explore. Should a host experience a high anomaly on one process it could try to warn other machines on the network about it, by sending inter-host semaphores. In addition, different hosts could interchange profiles. This, off course, implies a secure channel and a protocol for the communication. Today, cfengine offers a framework for the communication, and there is also research going on to define a standard format for intrusion detection (Intrusion Detection Message Exchange Format - IDMEF)[7].

Cfengine and pH alone might not be able to document intrusions on all fronts of the system. They should therefore be able to spawn other intrusion detection and forensic systems on demand if they are present, e.g. packet based detectors like SNORT[13], or Network Flight Recorder[8], that are perhaps too demanding to run all the time. In that way, the combination of pH and cfengine could act as a front-line defense against network intrusion, and vcooperate to switch on forensic capture software and perform backup checks on system integrity. Alternatively they could simply collaborate to identify the appropriate forensic data for human examination. We hope to return to some of these problems in future work.

8. Availability

GNU Cfengine may be obtained from http://www.cfengine.org. pH may be obtained from http://www.cs.unm.edu/s̃oma/pH

References

[1] M. Burgess. A site configuration engine. *Computing systems (MIT Press: Cambridge MA)*, 8:309, 1995.

[2] M. Burgess. Automated system administration with feedback regulation. *Software practice and experience*, 28:1519, 1998.

[3] M. Burgess. Computer immunology. *Proceedings of the Twelth Systems Administration Conference (LISA XII) (USENIX Association: Berkeley, CA)*, page 283, 1998.

[4] M. Burgess. On the theory of system administration. *Submitted to J. ACM.*, 2000.

[5] M. Burgess. Two dimensional time-series for anomaly detection and regulation in adaptive systems. *13th International Workshop on Distributed Systems: Operations and Management (DSOM 2002)*, page 293, 2001.

[6] M. Burgess, H. Haugerud, T. Reitan, and S. Straumsnes. Measuring host normality. *ACM Transactions on Computing Systems*, 20:125–160, 2001.

[7] J. Arvidsson et al. Terena's incident object description and exchange format requirements. *RFC3067*, 2001.

[8] M.J. Ranum et al. Implementing a generalized tool for network monitoring. *Proceedings of the Eleventh Systems Administration Conference (LISA XI) (USENIX Association: Berkeley, CA)*, page 1, 1997.

[9] J.L. Hellerstein, F. Zhang, and P. Shahabuddin. An approach to predictive detection for service management. *Proceedings of IFIP/IEEE INM VI*, page 309, 1999.

[10] S. A. Hofmeyr, A. Somayaji, and S.Forrest. Intrusion detection using sequences of system calls. *Journal of Computer Security*, 6:151–180, 1998.

[11] P. Hoogenboom and J. Lepreau. Computer system performance problem detection using time series models. *Proceedings of the USENIX Technical Conference, (USENIX Association: Berkeley, CA)*, page 15, 1993.

[12] P.D'haeseleer, Forrest, and P. Helman. An immunological approach to change detection: algorithms, analysis, and implications. *In Proceedings of the 1996 IEEE Symposium on Computer Security and Privacy (1996).*

[13] Snort. Intrusion detection system. *http://www.snort.org.*

[14] A. Somayaji and S. Forrest. Automated reponse using system-call delays. *Proceedings of the 9th USENIX Security Symposium*, page 185, 2000.

[15] A. Somayaji and S. Forrest. Automated response using system-call delays. *Proceedings of the 9th USENIX Security Symposium (USENIX Association; Berkeley, CA)*, page 185, 2000.

[16] A. Somayaji, S. Hofmeyr, and S. Forrest. Principles of a computer immune system. *New Security Paradigms Workshop, ACM*, September 1997:75–82.

SESSION 2

Internet Accounting

Chair: Burkhard Stiller
ETH Zurich / UniBw Munich, Switzerland / Germany

A HIGHLY DISTRIBUTED DYNAMIC IP MULTICAST ACCOUNTING AND MANAGEMENT FRAMEWORK

Hassen Sallay, Olivier Festor
The Madynes Research Team
LORIA-INRIA Lorraine
615, rue de Jardin Botanique
54602 Villers-lęs-Nancy, France
Tel: +33 (0) 383.592.000, Fax: +33 (0) 383.278.319
Hassen.Sallay@loria.fr, Olivier.Festor@loria.fr

Abstract: We present a highly distributed management architecture dedicated to IP multicast services. This architecture relies on a three level hierarchical model over which both management data and functions are distributed. We show how the architecture can be used to support an extended multicast accounting algorithm which adapts itself to the dynamics of a multicast tree and detail the implementation of the proposed framework using active network technology.

Keywords: Dynamic Accounting, Multicast Management, IP Accounting, Active Networks.

1 Introduction

As pred'cted in the last years [17], IP multicast is slowly moving from an optional feature in some networks to a primary service in the Internet. Multicast services gain more and more importance and become increasingly attractive. Multicast protocols are now deployed in almost all router products available on the market and they support a wide variety of applications (cooperative work, video-conferencing, tele-teaching, CDN update, ...).

With the advent of protocols like PIM-SSM [1] or EXPRESS[2] [26, 13] which are scalable and which cover a wide number of applications, multicast deployment and usage will grow even faster. Unfortunately, this success curve is currently slowed by several factors. One of these limiting factors, is the lack of integrated management solutions able to cope with all components entering in the multicast service delivery chain [19].

Efficient management of multicast services is both a crucial requirement and a major challenge for multicast services. Building management solutions which can address dedicated security, accounting and fault management for multicast is a key to their successful deployment. But, due to its differences with unicast communications, namely the potentially huge number of participants together with its tree-based connectivity which dynamically evolves over time, the design and implementation of the

[1]Protocol Independent Multicast - Single Source Mode
[2]EXPlicitly Requested Single-Source Multicast

afore mentioned management services are much more complex to achieve. To be efficient in the multicast context, four factors need to be considered :

1 the multicast dynamics,

2 the scale factor,

3 the degree of specialisation for the management functions, and,

4 the ease of deployment and integration with the legacy.

Multicast services are dynamic by nature. This dynamic behaviour is mainly generated by join/leave operations of group members. The diffusion tree follows this dynamic behaviour, either expanding or reducing itself over time. In such an evolving environment, the signalling and management plane loops sometime share very close timing requirements, management being forced to follow near real-time constraints found in the control plane.

Multicast is in theory the solution which scales best. The problem of scalability is in itself not bound to the multicast technology used but rather to the implementation and the associated management solutions. For management solutions dedicated to multicast services, scalability is essentially a gradient of :

- the number of multicast groups to manage,
- the number of nodes of each managed multicast tree,
- the frequency of change in groups,
- the overhead of management and signalling.

Designing efficient management solutions that can scale up to thousands of groups and millions of group members, while maintaining a limited management overhead remains a very challenging task.

Management functions such as accounting and security have to integrate the specific nature of multicast in the definition of the corresponding management applications and related metrics. For example, it is not an easy task to maintain an up-to-date central knowledge of the number of participants together with their distribution in a multicast tree. Thus, if one of the accounting approach relies on considering the exact number of participants, these dynamic operations (join/leave) must be traced in a very precise way. Moreover new parameters like the number of active groups, the overall number of groups and members in a given network, the tree topology and the number of links, traffic volume and memory usage within routers must be considered in the cost allocation process[3] for multicast service.

For the security function, if a group has strong security requirements, cryptographic keys have to be generated and distributed each time a member joins or leaves a group to guarantee that members who left do no longer have access to the group's exchanged data and that members who join do not have access to data that was exchanged before they joined.

To be well integrated, any management solution must provide gateways and interfaces to standard protocols and frameworks. To be efficient, they also need to take

[3]i.e. the application of a set of strategies which enables a cost to be associated to each participant including the savings generated by the use of multicast.

advantage of the most recent technologies to facilitate their deployment and configuration. In the context of IP multicast management, this means that gateways with the SNMP world must exists and that the management platform must be sufficiently open and dynamic to adapt itself to changes and to be capable of supporting new types of management algorithms.

The goal of the work undertaken in our group, is to build such a framework to manage IP multicast services. Based on the above requirements and on an in-depth study of multicast management, we propose a management framework that is highly distributed and extensible to fit both very flexible service level constraints and very dynamic network conditions. In addition to describing the core concepts of the architecture, we show how it was implemented using an active network framework. To illustrate the applicability of the framework, we propose an extension of a recently described cost allocation algorithm and show how the framework can cope with the added dynamics and show how it can be used in the context of fault management.

The paper is organised as follow. Section 2 provides a description of work related to multicast accounting and security management. The proposed architecture is presented in section 3. The technology choices made to host this architecture are described in section 4. Section 5 is dedicated to the description of the application of the architecture for cost allocation and accounting purpose followed by the implementation details (section 6). Some conclusions together with an outlook for future work are given in section 7.

2 Related work

Several approaches have been designed and proposed so far for the management of multicast communications. Each of them targets a set of specific management functions.

Within IETF, the AAA[4] model [23, 7] has been designed. This architecture interacts with the various services it applies to through specific modules. Diameter [6] is the communication protocol used among different entities of the AAA architecture. This protocol can be extended to meet the requirements of specific target applications. So far, no extension was proposed to embrace multicast services. Thus, using the architecture for multicast management is not feasible as is and its scalability has not been investigated in this context.

HERZOG et al. [12, 11] propose different strategies to allocate the cost of a multicast tree over its members. A cost allocation mechanism and a LPM[5] architecture defining components in charge of access control and accounting have also been proposed in this work. An example of the proposed strategies is ELSD[6]. This strategy divides equally the cost of a link over all members in the downstream of the multicast tree. This strategy, which is the most equal for a tree/source schema, has been implemented in the MultiCost prototype.
Unfortunately this strategy, together with the LPM architecture, do not take into account the dynamics of a multicast tree and consider the costs associated to a link to be static. LPM also implements access control but does not support any key distribu-

[4]Authentication Authorisation Accounting architecture
[5]Local Policy Modules
[6]Equal Link Split Downstream

tion mechanism nor does it support any fault management function. The use of a tree metric for charging multicast communications is described in [8].

Designed by HOLBROOK et al., EXPRESS[7] [26], is a multicast communication model dedicated to the single source multicast schema. Within EXPRESS, a protocol named ECMP[8] offers support for accounting tasks and integrates options for security features in addition to routing and membership management. ECMP assumes the accounting architecture is based on a centralised architecture (the source being the center) from where accounting campaigns on a well known attribute are initiated on demand. Data collected through this process are global data related to the multicast tree like number of members, number of branches, ...
Collecting more fine grained data for more precise accounting strategies, e.g. number of members beyond a given router, generates a huge wave of requests over all nodes of the multicast tree. Centralised management at the source is known for its bottleneck, its intolerance to any fault, and the overload generated on the network with often unnecessary management traffic. ECMP does not consider the dynamics of the multicast tree, nor does it implement any cost allocation strategy. The protocol remains dedicated to source specific multicast trees and no implementation of this part is known to us so far.

Solutions for securing multicast can be found in [24, 10, 16, 4]. Centralised key generation by a KDC[9] in charge of manually distributing these keys is proposed in [24]. This solution has clear scalability problems due to the off-line coordination effort that needs to be provided between the KDC and the members of the multicast service. Several decentralised and automated solutions are proposed in [10, 16, 4]. These architectures distribute key management functions among the involved entities and have a much better scalability.

In [1], an architecture for fault and quality management of a multicast link based on SNMP is proposed. Another architecture, based on the integration of existing management tools like MTrace and MRM[10] [2] is also proposed in a previous work of our group [18]. While these architectures offer a good level of integration and interesting functions like limited fault management, topology discovery and test generation and processing, they face scalability issues and do not address security nor accounting. A dynamic topology discovery approach for IP multicast is proposed in [20]. An alternative is proposed is [15]. A more complete monitoring environment called MCPM [14] exists but does not address accounting.

3 A highy distributed architecture

In this section we present the basics of the management architecture we designed to overcome the limitations of existing approaches and fit the requirements identified in the context of IP multicast.

3.1 Design choices

The main concept behind our architecture is to distribute as much as possible both management data and processing units to maintain a high degree of dynamics and

[7]EXPlicity REquested Single-Source
[8]Express Count Management Protocol
[9]Key Distribution Center
[10]Multicast Reachability Monitoring

ensure scalability. This distribution follows a pattern that enables composition and coupling of operations. Our architecture embraces this principle to meet the requirements identified in the introduction.

We consider the multicast dynamics as a two facets entity : one at the micro-dynamics level and another at the macro-dynamics one. Through the micro-dynamics level, the evolution of the group members (join/leave operations to a multicast group) can be monitored. The macro-dynamic level represents the evolution of the multicast tree through expansion/reduction operations (i.e. routers leaving/joining the multicast tree). As we will see below, differenciating these two facets, leads to the design of a scalable management solution especially in terms of the number of group members supported.

The overhead generated by the transport of management data related to a multicast service can also be reduced by setting the granularity of the data to be exchanged. Thus, only data that is mandatory for a given management task is sent over the network. Management data storage becomes necessary at the places where these data are produced and tasks can be delegated to the various network nodes that participate in the multicast service delivery chain. In this case, the processing distribution follows the data distribution providing the most efficient solution for large groups.

3.2 Global architecture

The proposed architecture provides a 3-level hierarchy : the source level, intermediate nodes level and the edge nodes level. At each level of the hierarchy a dedicated management agent is operational (see Figure 1).

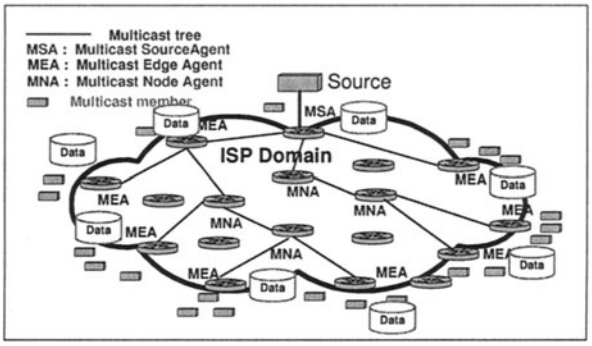

Multicast tree
MSA : Multicast SourceAgent
MEA : Multicast Edge Agent
MNA : Multicast Node Agent
Multicast member
Source
MSA
Data
MEA
ISP Domain
MNA

Figure 1. Global management architecture

At the multicast source level, a Multicast Source Agent (MSA) is in charge of the management activities. The instantiation and content of this agent can be part of the SLA/SLS that have been defined for the service between the service provided and the multicast service customer. The agent itself can be instantiated within the source which delivers traffic if a single administrative domain is considered. When multiple sources exist in one multicast session, the MSA agent is placed in the rendez-vous point node. In a multi-domain environment, we can extend our architecture by considering one administrative domain as a peer. The multicast service will be established by the cooperation of the different peer representing different domains. The source agent can play the role of the peer and negociate the service SLA setup among different ISPs concerned by the service. In this paper we consider only the single domain scenario, and leave the multi-domain scenario for a future work.

The MSA hosts a data storage facility that has the entire data related to the service. The collection algorithm initiated to feed the database follows the dynamics of the multicast tree. The database is updated each time an edge node (router) joins or leaves the tree.

At the edge node level, Multicast Edge Agents (MEA) are deployed. These agents manage the micro-dynamic facet of the multicast management. They maintain a local view of session join and leave operations performed by end users. To this end, these agents interact with the membership management protocols like IGMPv3 [5] or MLDv2 [9] through dedicated interfaces and build a local database which holds very detailed data about each member. These agents push the data forward to the concerned source agents (MSAs) only after all members of a session have left.

Nodes of the intermediate level (not the source, nor the edges) hold specific agents called Multicast Node Agent (MNA). Deployment or activation of these agents is done dynamically according to the expansion/reduction of the multicast tree of the managed service. This dynamic deployment implements the macro-dynamics facet related to topology changes in the multicast tree. These agents interact with the local multicast routing protocol entities through a well defined interface. Through this interface, data related to the service can be collected (e.g. number of sent/lost packets, number of links per multicast node, ...). Each agent has a local view of the tree topology, maintains a link with the underlying agents (MNA or MEA) as well as a link upward towards the source MSA.

This model is more scalable than a full source driven polling approach. Moreover it enables source agents to build service level statistics which are specific to the service level management process in use for the service (e.g. checking the conformity of the delivered service to the agreed level).

4 AMAM : an active network-based support of the architecture

The previously presented architecture can be implemented in several ways. This can be done with feature and protocol extensions of specific multicast approaches or through a standard management framework like SNMP with dedicated Management Information Models and associated MIBs together with usage scenarios. We chose an alternative to the above solutions, namely to exploit the benefit of active network technology in terms of flexibility and extensibility, to host the various components of the architecture [21, 22, 3]. We use this technology to dynamically download and operate both edge and node agents (MEAs and MNAs), enabling a seamless evolution that follows the topology changes of the managed multicast tree. The availability of a dynamic code distribution facility and the presence of execution environments on all nodes enables rapid and online cost calculation strategies change, update of security components, activation of tests for fault management purpose or more generally to push management functions where they are needed.

The resulting implementation called AMAM (Active-based Management Architecture for IP Multicast) is illustrated in Figure 2. Within AMAM, multicast management functions are designed as a set of dedicated active applications called plug-ins (accounting plug-in, security plug-in, test sender plug-in, ...). Plug-ins are stored in a repository within the source agent context. Once installed on the source agent, these components can be downloaded by either MEA or MNA agents where they are executed. Plug-ins uninstall themselves when they are no longer in use.

The FLAME active network framework is the execution environment we used to build AMAM. This execution environment enables both dynamic installation and removal of active applications but also APIs and associated libraries enabling the execution environment to dynamically extend the interface it offers to active applications (e.g. providing a new packet capture service). All AMAM plug-gins are defined as FLAME applications.

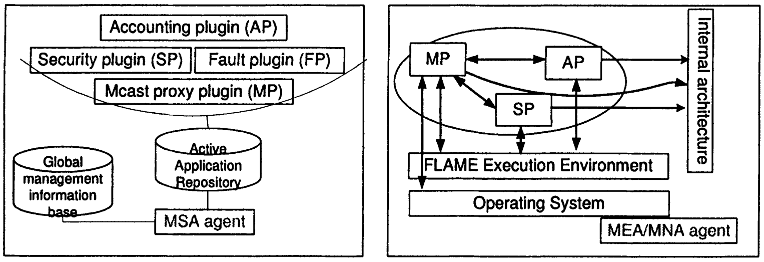


Figure 2. AMAM architecture

The repository contains by default four plug-ins, each of them having its own internal architecture. These plug-ins are :

- the **Mcast proxy plug-in (MP)**. The role of this plug-in is to provide transparency to the underlying multicast technology used. The plug-in interacts with the routing as well as with the membership management protocols through specific interfaces to collect information concerning members and delivered traffic. Providing such a protocol independence facilitates deployment and integration into legacy approaches. This plug-in is loaded in all agents.

  In the edge agents, MP builds a multicast table that contains information for each member on this edge. This generic table is built using proxy-lets which provide the link with the two currently supported group membership protocols namely IGMP for IPv4 and MLD for IPv6 (see Figure 3).

  In the node agents (MNAs), proxy-lets that communicate with the routing protocol are used. Other plug-ins use these proxy-lets and the data they collected to perform their management task (see Figure 2);
- the **Accounting plug-in (AP)** specialised in accounting tasks. It owns an interface with the local MP, with external APs deployed in the other nodes as well as with the group members that joined a session. Using the routing proxy-lets, this agent collects information related to the multicast traffic and feeds the local database. Based on this information and the member table built by the MP agent, this plug-in allocates the cost and computes the amount that will be assigned to each member. This information can then be used directly in the charging process and published directly towards the users through the *Client publishing* interface. The agent also maintains a copy of charging units in the local database (see Figure 3).
- the ***Security plug-in (SP)*** executes the security functions. Combined with a policy server which can be located at any place in the network, even in the

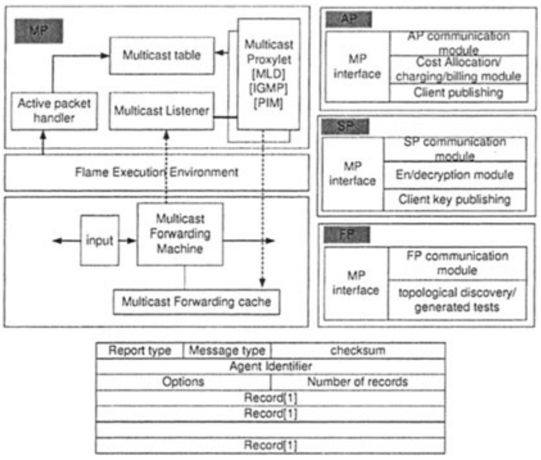


Figure 3. Internal architecture of the plug-ins

multicast source domain, it ensures access control and user authentication. The SP generates local keys and ensures their distribution to all connected members through its *key publishing* interface. This plug-in also executes the algorithms that encrypt/decrypt the multicast traffic that needs to be delivered (see figure 3).

- the ***Fault plug-in (FP)*** notifies the source of changes in the topology (i.e. add/remove of a branch in the multicast tree) that were recorded locally by either MNAs or MEAs. Based on the knowledge acquired through the reception of these notifications, the source agent (MSA) builds a topological view of the distribution tree. The plug-in configures itself through the source agent to generate test cases based on the information model defined in [18]. The routing proxy-let is also used through the MP interface to measure link quality over the distribution tree.

In addition to the plug-ins, a management data exchange protocol has been defined to enable communication among plug-ins either in different entities or in the same system. The protocol defines a generic data format (see Figure 3). This message format can be specialised for each plug-in. For example, the AP plugin defines a message to transport the data necessary to allocate the costs and another to transport the cost vector to be allocated. Each message contains the following elements :

- report plug-in type : an octet that specifies the type of the plug-in that sends the report. For example, if the *report type* of an MP agent is set to 1, then all messages received by FLAME whose type is 1 are forwarded to the local MP plug-in;
- message type : one octet describing the message type. Each plug-in defines its own message name-space;
- checksum : checksum over the entire packet. It is checked at each node which processes the message;

- agent identifier : a two octet identifier uniquely identifying the agent that initiated the message;
- options : used to specify a set of options required by plug-ins;
- number of records : number of data records contained in the message;
- record : contains the data which is specific to the message type.

The sequencing of messages is defined by each plugin.

5 Using AMAM for cost allocation and accounting management

Cost allocation is done through the application of a set of strategies which enable the assignment of a given cost to every participant in the downstream. In the multicast case, gain can be obtained by sharing costs. For example, the ELSD (Equal Link Split Downstream) strategy divides equally the cost of a link to all participants behind the link. The cost is computed over several parameters like the volume of traffic, the used bandwidth, congestion state of the link, ... To this transport cost, one has to add the content cost. The ELSD strategy has been implemented in [12] but without taking into account the dynamic nature of multicast trees.

Our architecture enables the support for an extended ELSD approach that we propose. This extension, called D-ELSD explicitly considers the dynamics of a multicast tree as a fundamental parameter of the strategy as opposed to ELSD.

Lets consider following definitions:

- An allocation session allocates costs among leaving participants. This session is started each time an edge node looses its last participant. The node that initiates the allocation session is represented as *init* in the remainder;
- Δ_t denotes the duration between the arrival of the first member on an MEA and the departure of the last member from the same MEA. Over this period, the MEA records all arrivals and departures of its local members;
- $\Delta_t = \sum_{m=0}^{k} \delta_m$ and $N_{loc}(i) = Vect_{m=0}^{k}[n_m]$ where n_m is the number of active members in δ_m at node i;
- for each intermediate node, the MNA agent maintains the evolution of the number of branches towards downstream nodes. Let $\Delta_t = \sum_{h=0}^{r} \theta_h$ and $B_{loc}(i) = Vect_{h=0}^{r}[b_h]$ such that b_h is the number of branches in θ_h[11];
- let $\Delta_t = \sum_{j=0}^{p} \tau_j$ and $N_{downstream}(i) = Vect_{j=0}^{p}[n_j]$ such that n_j is the number of members downstream of node i within τ_j;
- let $N_{merge}(i)$ be the resulting vector of the following classification (δ_m, τ_j) from $N_{downsteam}(i)$ out of every downstream branch and N_{loc} over Δ_t for node i. Thus, if we have only one branch downstream : $N_{merge}(i) = Vect_{s=0}^{s=k+p}[n_s]$ over $\Delta_t = \sum_{s=0}^{k+p} \alpha_s$ such that $n_s = n_i + nj$ for $\alpha_s = \min(\delta_m, \tau_j)$. Thus, we also have $N_{merge}^{1/N}(i)$: the notation of the vector whose elements are $1/n_s$;

[11]Note that one MEA can serve its members locally and have downstream branches to other MEAs or MNAs. Thus, an MEA can maintain both vectors of numbers for members and branches.

- let $chem(i, j)$ be the path from a node i to a downstream node j belonging to the multicast tree and finally, let $cost(i, i+1)$ be the cost of a link between a node i and its first downstream node $chem(i, j)$ towards node j;

D-ELSD cost allocation is done in two phases : (1) a preparation phase during which the data necessary for the application of a strategy is collected. The necessary data is the number of members downstream (in the $chem(source, init)$ path)(2) a processing phase where the cost for all members which receive their traffic through a path that crosses $chem(source, init)$. Thus :

- for the duration Δ_t, the node that initiates the cost allocation session (node i), builds his $N_{merge}(i)$ vector. Once built, this vector includes the entire dynamics of the downstream multicast subtree. The initiating node sends this vector to the first upstream node ($i-1$) on the path to the source ($chem(source, i)$).

  This vector will then be used as the $N_{downstream}(i-1)$ vector. Node $i-1$ then builds its fusion vector and sends it upwards. This process continues until all vectors have reached the source;

- once the MSA received the vector $Vect[n_i]_{i=0}^{i=l}$ for the duration $\Delta_t = \sum_{i=0}^{l} \beta_i$, it builds the following allocation vector :

  $$Vect_{toSend}(source) = cost(source, downstream\_node) * [N_{merge}^{1/N}(source)] \tag{5.1}$$

  and sends it to the first node downstream. l represents all join/leave events that have been registered for the Δ_t duration for the subtree containing *chem-(source,init)*;

- every node i in $chem(source, init)$ that did receive an allocation vector from the upstream node, builds the cost vector for its members according to the following equation :

  $$Vect_{cost}(i) = Vect_{toSend}(i-1) * I(l, l) * (M(l, k) * N_{loc}) \tag{5.2}$$

  where $I(l, l)$ is the identity matrix of size l and $M(l, k)$ the transformation matrix of the vector N_{loc} from size k to a size l vector.

  $$M(l,k) = \begin{pmatrix} 1 & 0 & \ldots & 0 \\ \vdots & \vdots & \ldots & \vdots \\ 1 & 0 & \ldots & \vdots \\ 0 & 1 & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ \vdots & 1 & \vdots & 1 \\ \vdots & 0 & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \ldots & 0 & 1 \end{pmatrix} \tag{5.3}$$

  where the number of 1's per column is the number of δ_l in δ_k

- Node i sends a notification to its upstream nodes that it assigns the following cost vector :

$$Vect_{toSend}(i) = Vect_{toSend}(i-1) + cost(i, i+1) * N_{merge}^{1/N}(i) \tag{5.4}$$

  Every node on the path $chem(source, init)$ does the same processing until the vector reaches the node that initiated the session.

As soon as the cost allocation vector has been received (see equation 5.2), each MEA agent that is in the subtree on the $chem(source, init)$ path ensures allocation for all its local members. This is done, based on the detailed knowledge of the behaviour of each member, behaviour that has been recorded in the local database. A member m_i being connected during a $\subset \Delta_t$ period of time will get a cost equal to the sum of elements from $Vect_{cost}$ for this duration. These costs can be communicated to the members which can thus estimate their cost in near real time.

Note that the link cost is computed while respecting the strategies of load computation. Load computation can rely on several parameters like the volume of received data, the duration of the connection to the service, the used bandwidth, ...). The cost vector depends only on the cost allocation strategy. In our framework, we used an extended version of ELSD but other strategies can be implemented as well.

6 Implementation issues

In this section we will discuss some implementation issues in the AMAM architecture. Figure 4 represents the deployment senario of the management agents. We assume that in each edge router, a MEA is installed and that the MP plugin is downloaded by default. The MP plugin serves to interact with the multicast membership and routing protocols as mentionned before. When MP detects the arrival of the first member joining the multicast session, the MEA initiates the deployment of the different MNAs in all the routers which are in the route to the first MNA/MEA belonging to the multicast distribution tree. These MNAs take their proper deployment configuration from a policy sever provided by the FLAME environment. Once installed, the MNAs records the routing information that will be used in the accounting process (like number of branches...). The MEA records in its turn all the arrival and departure of its local members. When The MEA detects the departure of the last member, it initiates a cost allocation session according to the D-ELSD strategy. The source computes the cost allocation vector and sends it to the concerned downstream routers. Each MEA, based on this cost allocation vector and its local information computes the charge of its local members and sends them a real time invoice. Finally, before desinstalling itself, the MNA notifies the policy server. This interaction with the policy server is also used to construct and update the topologic view of the multicast tree distribution at the source. The MEA, before desinstalling its downloaded plugins (except MP plugin), sends the local management data to the source for a final storage purposes (see Figure 4).

Note that D-ELSD is computed only when a topology change in the multicast distribution tree occured and not as long as there is a join/leave of members. Consquently, the scalability of management functions in the core is the same as the one of the multicast routing protocol since these management nodes only communicate and process data on a tree topology change only.

The open interfaces with the membership and routing protocols can be obtained by extending the major open source IGMP, PIM-SM and MLDv2 implementations.

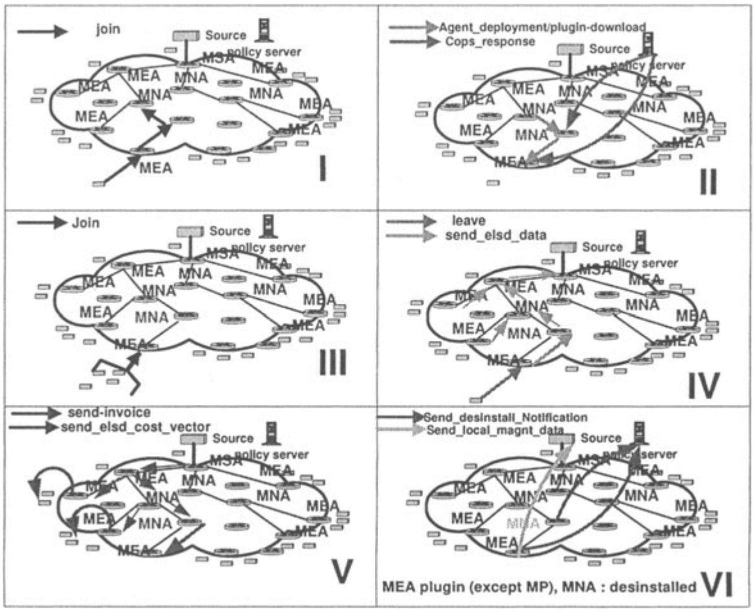


Figure 4. AMAM Agents deployment scenario

In our research group, we have extended the IGMPv3 interface for some accounting purposes. For instance, the connection duration for each member is computed with some timer variable used by the IGMP proxylet to fixe the time of arrival and departure of each member.

To evaluate how the performance could be affected when extra functions are deployed in the agents, FLAME provides some integrated functionnalities for perfermance evaluation that we will use for our real implementation of AMAM. To improve the scalability gain in the case of large scale of group members a simulation work will be done for more performance analysis.

7 Conclusion and future work

In this paper, the need for developing management architectures dedicated to multicast services was addressed. The dynamics of these services, the very specific requirements towards the standard management functions especially the needs for scalability and ease of deployment and maintenance have been identified as the major points, a management solution for multicast services should master.

Based on these requirements, a management architecture was proposed. This architecture is based on a three level hierarchy over which both management data and functions are distributed. It enables the fusion of both signalling and management planes for several multicast management functions like security and accounting. The FLAME execution environment has been selected to host the architecture. The resulting environment is called AMAM. Within AMAM, management functions are defined in terms of active applications which can be downloaded on demand and which use a dedicated protocol to exchange management messages. Integration with the control planes (membership management and routing) is done through proxy-lets. New management functions can be dynamically added.

Two management functions and their implementation within AMAM have been presented. Through the proposal of an extended ELSD strategy, we have shown that the architecture can support accounting functions that support the dynamics of multicast group members. Other functions can be implemented in the framework in a similar way. For example, in the security management process and especially in the key management function, AMAM implements efficiently the solution based on the principle of a single global key per source and several local keys managed by the edge routers. The source agent generates one global key and ensures its distribution to all nodes of the multicast tree. This key is used to encrypt the traffic that will be delivered over the secured group. Each edge agent decrypts the traffic with the global key and generates a local key, ensures its distribution to its local members and encrypts the traffic with this local key. Local update of cryptographic keys makes the solution more scalable avoiding a complete update over the entire tree each time a member joins or leaves. Furthermore, the architecture can deploy approaches like MRM through configuration of MNAs and MEAs. Session control can be done based on the information model proposed in [18]. Another multicast monitoring service was also implemented in this architecture. This is the Hierarchical Passive Multicast Monitoring (HPMM) framework [25] which does fault correlation over the multicast tree.

Pushing forward the implementation of the architecture within the FLAME active network constitutes our first goal in the near future. This mainly requires some refinement in the specification of the management plug-ins. A second goal is to complete the study of the behaviour of the framework in the context of SSM multicast services. Finally, the architecture need to be extended to be deployed in a multi-domain (multi-ISP) environment.

References

[1] E. Al-Shaer and Y. Tang. Smrm: Snmp-based multicast reachability monitoring. Proc. NOMS'2002, p. 467-482, 8th IEEE/IFIP Network Operations and Management Symposium : Management Solutions for the New Communications World, R. Stadler and M. Ulema, Editors, ISBN 0-7803-7382-0, April 2002.

[2] K. Almeroth, L. Wei, and D. Frainacci. Multicast reachability monitor (mrm), April 1999.

[3] L. Andrey, I. Chrisment, O. Festor, and E. Fleury. *Les réseaux multi-média*, chapter Les réseaux actifs. collection IC2 chez Hermès, 2000.

[4] A. Ballardie. Scalable multicast key distribution, May 1996. IETF RFC 1949, Experimental.

[5] B. Cain, S. Deering, B. Fenner, I. Kouvelas, and A. Thyagarajan. Internet group management protocol, version 3, March 2001. RFC 1075.

[6] P. Calhoun, J. Arkko, E. Guttman, G. Zorn, and J. Loughney. Diameter base protocol, june 2002. <draft-ietf-aaa-diameter-11.txt>.

[7] C. de Laat, G. Gross, L. Gommans, J. Vollbrecht, and D. Spence. Generic AAA Architecture, August 2000. RFC 2903.

[8] H.J. Einsiedler, P. Hurley, B. Stiller, and T. Braun. Charging multicast communications based on a tree metric, May 1999. 1st Workshop on Multicast Protokolle und Anwendungen, Braunschweig, Germany.

[9] B. Haberman and R. Worzella. Ip version 6 management information base for the multicast listener discovery protocol, January 2001. RFC 3019, Standards Track.

[10] H. Harney and C. Muckenhirn. Group key management protocol (gkmp) architecture, July 1997. IETF RFC 2094, Experimental.

[11] S. Herzog, S. Shenker, and D. Estrin. Sharing the "cost" of multicast trees: an axiomatic analysis. *IEEE/ACM Transactions on Networking*, 5(6):847–860, 1997.

[12] Shai Herzog. *Accounting and Access Control for Multicast Distributions : Models and Mechanisms*. PhD thesis, USC, august 1996.

[13] H. Holbrook and B. Cain. Source-specific multicast for ip. november 2000.

[14] A. Kanwar, K. Almeroth, S. Bhattacharya, and M. Davy. Enabling end-user network monitoring via the multicast consolidated proxy monitor. In *SPIE ITCom Conference on Scalability and Traffic Control in IP Networks, Denver, Colorado, USA*, 2001.

[15] Jangwon Lee and Gustavo de Veciana. Resource and topology discovery for IP multicast using a fan-out decrement mechanism. In *INFOCOM*, pages 1627–1635, 2001.

[16] R. Oppliger and A. Albanese. Distributed registration and key distribution (dirk), May 1996.

[17] Bob Quinn and Kevin Almeroth. Ip multicast applications : Challenges and solutions, septembre 2001. RFC 3170, Informational.

[18] H. Sallay, R. State, and O. Festor. A distributed management platform for integrated multicast monitoring. Proc. NOMS'2002, p. 483-496, 8th IEEE/IFIP Network Operations and Management Symposium : Management Solutions for the New Communications World, R. Stadler and M. Ulema, Editors, ISBN 0-7803-7382-0, April 2002.

[19] K. Sarac and K. Almeroth. Supporting multicast deployment efforts: A survey of tools for multicast monitoring.

[20] K. Sarac and K. Almeroth. Scalable techniques for discovering multicast tree topology, 2001.

[21] J. Schoenwaelder. Emerging internet management technologies. IEEE IM'99 (Tutorial), October 1999.

[22] D.L. Tennenhouse, J.M. Smith, W.D. Sincoskie, D.J. Wetherall, , and G.J. Minden. A survey of active network research. *IEEE Communications Magazine, Vol. 35, No. 1, pp80-86*, January 1997.

[23] J. Vollbrecht, P. Calhoun, S. Farrell, L. Gommans, G. Gross, B. de Bruijn, C. de Laat, M. Holdrege, and D. Spence. AAA Authorization Framework, August 2000. RFC 2904 Informational.

[24] D. Wallner, E. Harder, and R. Agee. Key management for multicast : Issues and architecture. Internet RFC 2627, june 1999.

[25] J. Walz. Multicast Monitoring - Current Usage and a New Hierarchical Pr otocol . Master's thesis, Dept. of Computer Science, University of Massachusetts, February 2001.

[26] Hugh W.Holbrook and David R. Cheriton. Ip multicast channels : Express support for large-scale single-source applications. 29(4), october 1999.

USER ORIENTED IP ACCOUNTING IN MULTI-USER SYSTEMS

Ge Zhang, Bernd Reuther, Paul Mueller
Department of Computer Science, University of Kaiserslautern
Postfach 3049, 67653 Kaiserslautern, Germany
Tel: ++49 631 2054520, ++49 631 2052161, ++49 631 2052263, Fax: ++49 631 2053056
gezhang@informatik.uni-kl.de, reuther@informatik.uni-kl.de, pmueller@rhrk.uni-kl.de

Abstract: The traditional IP accounting method is IP address oriented, that means one IP address corresponds to one user, but it can not meet the finer granularity accounting requirement in multi-user systems, in which many users share one or more IP address at the same time. In the multi-user systems the user oriented IP accounting can distinguish the producers of the IP traffics, which come from the same IP address. Hence it is a more accurate accounting method than traditional IP address oriented accounting method. In this paper, we present the technology of the user oriented IP accounting, and describe the principle of this method, and the realization considerations.

Keywords: IP Accounting, IP Billing, Multi-user System

1. INTRODUCTION

With the rapid development of the Internet, more and more services are provided by the Internet, more and more users enjoy these Internet services, and consequently more and more traffic are produced. For example, during the last three years the IP data volume of the University of Kaiserslautern doubled every year. In this situation, it is very important to control and measure the Internet usage. IP traffic accounting provides information about the usage of a network and therefore helps to manage it. The accounting data may be used also for billing. Today billing is commonly used by Internet service providers (ISP). But billing systems might be used even within LAN. This enables to allocate the costs, which are produced by the network traffic of a single-user or an institution using a campus network. But billing has also an influence on the behavior of the users. Users will not use the network resource responsible if the usage is free of charge. Because of the rising costs it is reasonable to present bills to the end users also. This aspect becomes very important when a network offers different Classes of Service (CoS).

It does not depend if billing is used in a LAN to allocate costs or to motivate reasonable network usage. In both cases it makes sense to be able to correlate the network traffic with the users, which are responsible for it. Today several IP accounting and billing solutions exist. But these solutions correlate IP addresses and traffic only. But there are several scenarios where an IP address is not associated with one user. For example in the computer center of University of Kaiserslautern the multi-user computers or PC pools play a very important role. The traditional IP accounting solutions are not able to distinguish different users of those systems.

This paper describes method that enables the distinction of users even on multi-users systems.

2. TRADITIONAL IP ACCOUNTING METHOD

2.1 IP accounting and IP billing system

Accounting is "The collection of resource consumption data for the purposes of capacity and trend analysis, cost allocation, auditing, and billing."[1]. Whereby Billing is the process of utilizing the processed Accounting Records on a per user basis to generate the invoice. The architecture of a general IP billing system is illustrated in Figure 1[2].

An IP Billing system consists of three layers: Traffic Flow Meter Layer, Mediation Layer and Billing / OSS / BSS (Operating / Business Support System) Layer. The Traffic Flow Meter Layer records the network activities in Raw Data Records (RDR) like an electricity meter. The Mediation Layer collects the Raw Data Records from various Network Elements, and processes the Raw Data Records to produce the Usage Records, stores the Usage Records in database, distributes the Usage Records to different applications in layer 3. The applications (e.g. Billing, Fraud Detection, Traffic Analysis etc.) in layer 3 respectively process the Usage Records for different application purposes and generate various reports. IP accounting includes the layer 1 and layer 2 in the IP billing system architecture.

The traditional IP accounting system collects and processes the resource consumption data on the basis of the IP address, that means, in these systems an IP address is considered as one user. But it is not the case in the multi-user systems, e.g. most Unix systems or Windows Terminal Servers, in these systems many users share one or more IP addresses simultaneously. In this case one IP address is not equal to one user. Contrasting with the traditional IP address oriented IP accounting method, we suggest the User oriented IP Accounting method. Our NIPON (Nutzerbasiertes IP accounting) project aims at solving this problem.

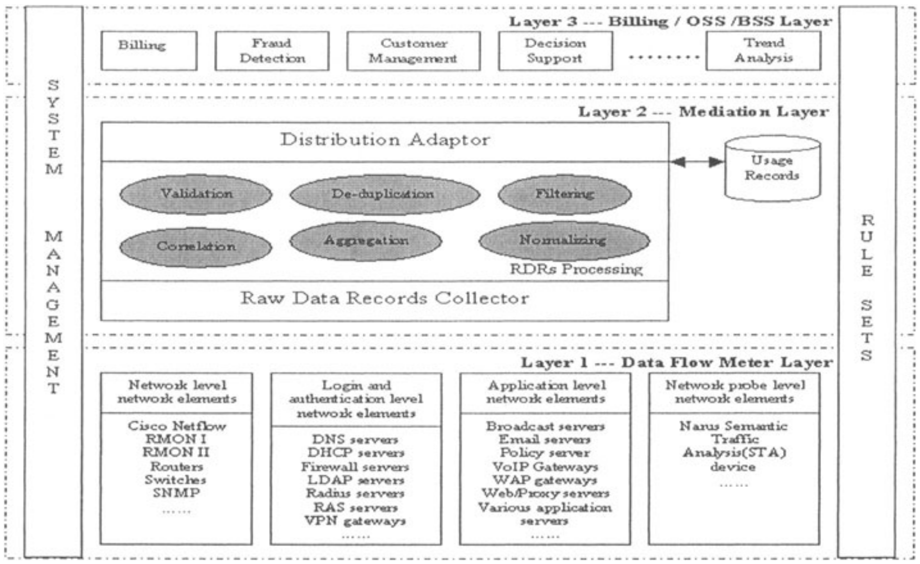


Figure 1. IP Billing System Architecture

2.2 User Information Processing Method in Traditional IP Accounting System

The general IP accounting process can be described in two steps:

1) The Traffic Flow Meter collects IP traffic information from various Network Elements, and stores these information in the form of Raw Data Records;
2) The Mediation layer collects the RDRs, and then processes (Validation, de-duplication, filtering, correlation, aggregation and normalizing) the RDRs to produce the Usage Records.

Figure 2 illustrates the process of IP traffic information processing.

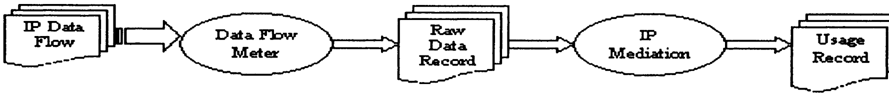


Figure 2. IP Traffic Information Processing

During the process of the IP traffic information processing, the traditional IP accounting method will regard the IP addresses in the RDRs as the user information, and these IP addresses will be mapped to corresponding users by the so called **Correlation** module, which is one of the RDR processing modules in IP mediation layer (see Fig.1). The function of the Correlation module is to merge several RDRs, which have some relationships, to create a single record, this can provide a single, complete view of information about an event [3]. In the traditional IP accounting system the Correlation process is on the basis of IP address. The Correlation module maintains an IP address & User Map Table, according to this table the Correlation module can map the IP addresses to the corresponding users. Figure 3 illustrates an example of the process of how the Correlation module mapping the IP addresses in RDRs to corresponding Users.

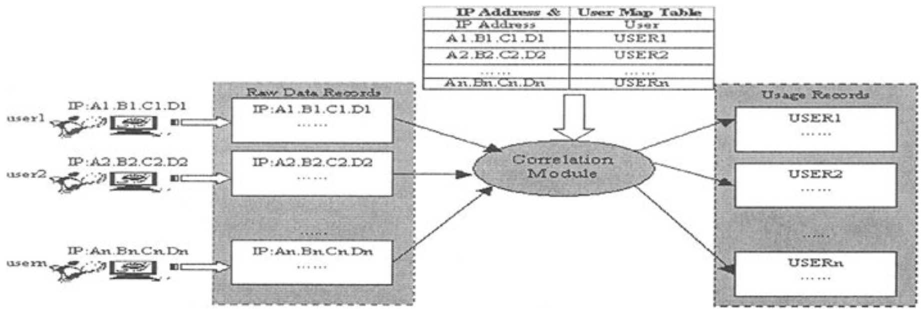


Figure 3. Mapping IP addresses to corresponding Users

The above described user information processing method of traditional IP accounting system has no problem in single-user systems or in the condition that each IP address can be considered as the user who will be responsible for the IP traffic. In these cases an IP address can uniquely represent a person or an institution that will be responsible for the IP traffic generated by this host.

But in multi-user systems, many users can share an IP address at the same time. In this case an IP address cannot uniquely represent a user. The traditional IP accounting method cannot accurately process the user information in this situation.

Figure 4 illustrates an example of the Correlation process of the traditional IP accounting method in multi-user system.

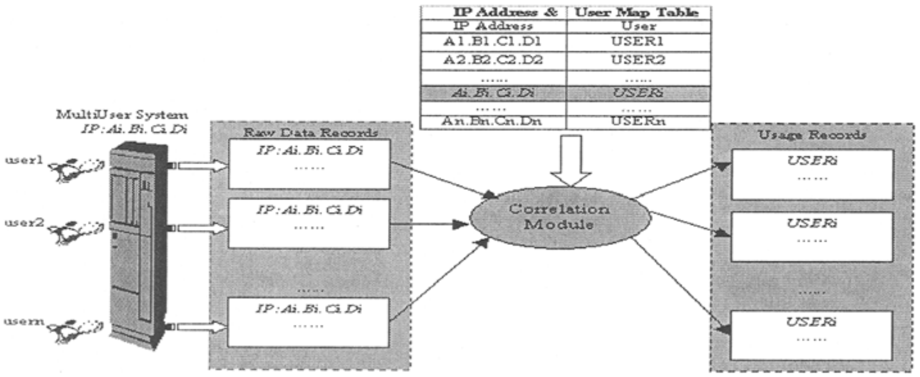


Figure 4. Traditional IP accounting method in mapping IP addresses to corresponding Users in multi-user system

In Figure 4 several users share a multi-user computer, and they share the same IP address Ai.Bi.Ci.Di. According to the traditional user information process method, only the IP address information Ai.Bi.Ci.Di in the Raw Data Records will be used to represent the user. After the Correlation module processing this IP address is mapped to USERi and the user information is recorded in the generated Usage Records. Through this user information processing method, all the IP traffics, which come from the same multi-user computer with the same IP address Ai.Bi.Ci.Di, but produced by different users (user1, user2, ... ,usern), are regarded as produced by the same user (USERi).

From the above introduced traditional IP Address oriented IP accounting method, we know that, this traditional IP accounting method can not meet the fine granularity requirement of the user information processing in multi-user systems.

3. THE PRINCIPLE OF USER ORIENTED IP ACCOUNTING IN MULTI-USER SYSTEMS

User oriented IP accounting collects traffic information and processes the RDRs on the basis of User. Before we discuss about it, we should first answer the question: What is a User? Then we will describe the principle of User oriented IP accounting method and the realization considerations.

3.1 User Model

3.1.1 Terminology

The meaning of the term *user* depends on the context where the term is used. When talking about traditional IP accounting systems a user is a host that is the source or the sink of IP traffic. Within IP billing systems the term user means the person or institution, who is responsible for some IP traffic, i.e. who has to pay for the IP traffic. In a multi-user system a login name or an identifier represents a so-called user. This user may be one real person or a group of real persons. Because of this ambiguous usage of the term user we will present some definitions of terms, which will be used within this paper:

- *Host-Identifier* is a unique identifier for an endsystem of the network layer. In the context of IP networks an IP address can be used as a synonym for a Host-Identifier, since IP addresses are unique numbers for network layer devices, at least within an administrative domain.
- *User-Identifier* or UID is a unique identifier for an account on a multi-user system. This term is commonly used in the context of multi-user systems.
- *Traffic-Originator* ::= <*Host-Identifier*, [*User-Identifier*]>. A Traffic-Originator (TO) is responsible for specific outgoing and incoming traffic flows. A TO may be described only by a Host-Identifier or by a Host-Identifier and a User-Identifier. This means a TO is an exclusively used computer or an account on a multi-user system.
- *User*::= <*Traffic-Originator1* [, ... *Traffic-OriginatorN*]> is a unique identifier for real person or a group of persons which are associated with

one or more TOs. Each TO is associated with exactly one User. Usually a User identifies one real person who has access to one or more single-user systems or accounts on multi-user systems. When a group of real persons share an account or a single-user system, this group may be described by one User.

- *Purchaser*::= <*User1* [, ..., *UserN*]> is a unique identifier of a person or an institution who will pay for the traffic that is originated by one or more Users.

3.1.2 User oriented IP accounting definition

According to the definitions of the previous subchapter, traditional IP accounting distinguishes traffic from different Host-Identifiers, i.e. IP-Addresses only. Within the mediation or billing layer Host-Identifiers are mapped to purchasers directly. In contrast to this the User oriented IP accounting distinguishes different TOs. Within the mediation or billing layers the TO will be mapped to Users which are mapped to Purchasers. It is important to distinguish between Users and Purchasers, because of their different responsibilities. The User is responsible for the traffic that is produced. If there occur some problems with some traffic flows or the amount of traffic that is produced, then it is important to know who is responsible for the traffic. But in order to send a bill to some person or institution it is only necessary to know who is responsible for paying for the produced traffic.

The User oriented IP accounting extends the concept of traditional IP accounting by considering User-Identifiers in addition to Host-Identifiers. Traditional IP accounting can be regarded as a special case of User oriented IP accounting, since in traditional IP accounting the TOs of the traffic flows have always the same UID. Comparing with traditional IP accounting, User oriented IP accounting should provide information about User-Identifiers, which must be correlated to the accounted IP traffic. Therefore three technical problems must be solved for the User oriented IP accounting:

- Accounting of User-Identifiers. More precisely a relationship between a User-Identifier and a traffic flow must be recorded. This must be done within the multi-user system, since this information is not available outside of the multi-user system.
- Correlation of TOs (which may contain User-Identifiers) with traffic flows.
- Transport of the accounting data that is recorded in the multi-user system to the correlation module.

3.2 User oriented IP Accounting Method

With the above described User model, User oriented IP accounting will identify the producer of each traffic flow or package, and the corresponding TO information will be added into the RDRs to identify who produce them. A flow is defined as a set of packets between two endpoints (as defined by their source and destination attribute values and start and end times) [4]. For example, in the realization of the traffic meter NeTraMet [5], a flow is identified by a 5-tuple, i.e. <protocol, source address, destination address, source port number, destination port number>.

The TO information is unknown to the outside of the multi-user systems. If we want to obtain the TO information from a multi-user system, a mechanism must be resided in the multi-user system to implement this function. Here we call this mechanism Agent method.

3.2.1 Agent method model of User oriented IP accounting

Figure 5 illustrates the Agent method model of User oriented IP accounting in multi-user systems. The User oriented IP accounting architecture is based on the traditional IP accounting architecture. The differences between the new method and the traditional method are:

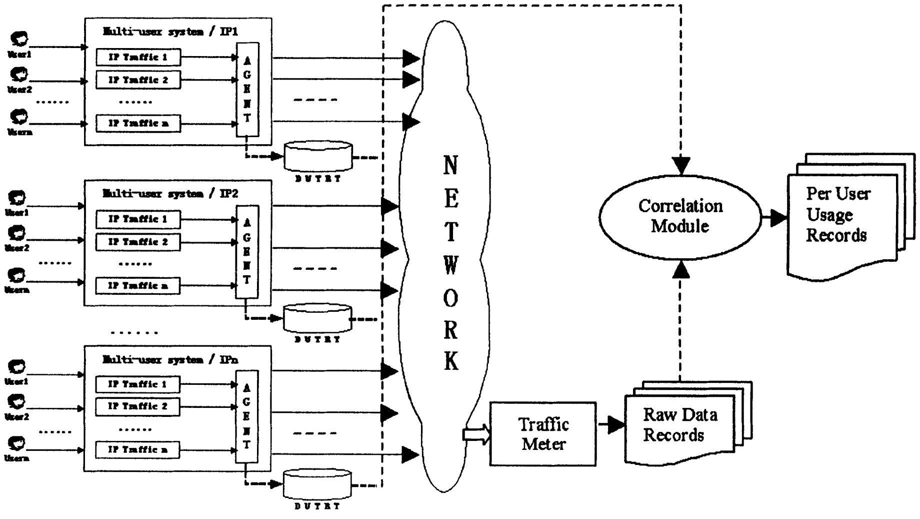


Figure 5. Agent Method Model of User Oriented IP Accounting

- An Agent is introduced into the multi-user system, which is used to collect the User-TrafficFlow relationship information according to the User Model. The collected User-TrafficFlow relationship information will be recorded into a so-called Dynamic User-TrafficFlow Relationship Table (DUTRT). This is used to record the IP traffic flows and their corresponding TO information. The Agent can also act as a standalone IP traffic meter, which measures the IP traffic from the multi-user computer, in which it locates. In this situation the entries in DUTRT can be used as RDRs. This will simplify the correlation processing, but it will contribute more overhead to the multi-user system.
- The Correlation module in the IP mediation layer uses a DUTRT, not a simple IP Address-User Table, to map the RDRs to the corresponding

users. In this case, more attributes in RDRs should be used for the user correlation purpose.

The User oriented IP accounting process with Agent method can be described as below:

1. The Agent checks all traffic flows, and then extracts the corresponding TO and other information to identify each flow. All the generated User-TrafficFlow relationship information will be stored in a DUTRT.
2. Traffic Flow Meter collects the IP traffic information to generate the Raw Data Records.
3. The RDRs and DUTRT records will be sent to or collected by IP mediation layer.
4. The Correlation Module uses the DUTRT to map the RDRs to the corresponding users and adds the user information to the new generated Usage Records.

3.2.2 Dynamic User-TrafficFlow Relationship Table

The Dynamic User-TrafficFlow Relationship Table is generated by the Agent. It is used to record the TO information of each traffic flow or package, and the Correlation Module will use it to identify the users of the traffic flows.

Whenever a new traffic flow is produced, a new entry with this flow's TO and Correlation attributes will be created into the DUTRT. And the start time of the flow will also be recorded. If the Agent is used as a standalone meter, the continuous statistic information of the flow (such as bytes or packages etc.) will be added into the same entry. After the stop of the flow, the end time of the flow will be recorded into this entry.

The records of DUTRT table will be sent to or collected by IP Mediation layer periodically.

According to the User model, several attributes in a traffic flow and corresponding TO information are collected to construct a DUTRT. For example, an entry of the table may include several items as below:

<UserID, Source IP, Source Port Number, Destination IP, Destination Port Number, Timestamp>.

A record in the DUTRT includes three kinds of attribute:

1. Traffic-Originator attribute: it is used to uniquely identify a TO, who produces the traffic. As for the above example, TO attribute includes these items: **<UserID, Source IP>**.
2. Correlation attribute: it is used to correlate a traffic flow to a corresponding user. The Correlation attribute includes flow related information that are usually extracted from IP traffic flows or IP packages, and RDRs produced by meters also record all these needed attributes. [6] defines the attributes and format of RDR. As for the above example, Correlation attribute includes these items: **<Source IP, Source Port Number, Destination IP, Destination Port Number, Timestamp>**

3. Statistic attribute: If the Agent is used as a standalone meter, the attributes such as bytes, packages count etc. will be collected for the purpose of measuring the network resource consuming.

The Agent does not generate the RDRs directly, but generates the DUTRT, the reasons are:

- This method can easily be integrated into the now existent IP Billing system.
- Generating all the RDRs and all their attributes will cost more system resource in multi-user systems and will affect the system performance.

3.2.3 Agent

The Agent can be implemented as a standalone software or a part of the multi-user system kernel. It checks all traffic flows to extract user information for the purpose of User oriented IP accounting. Its main functions include:

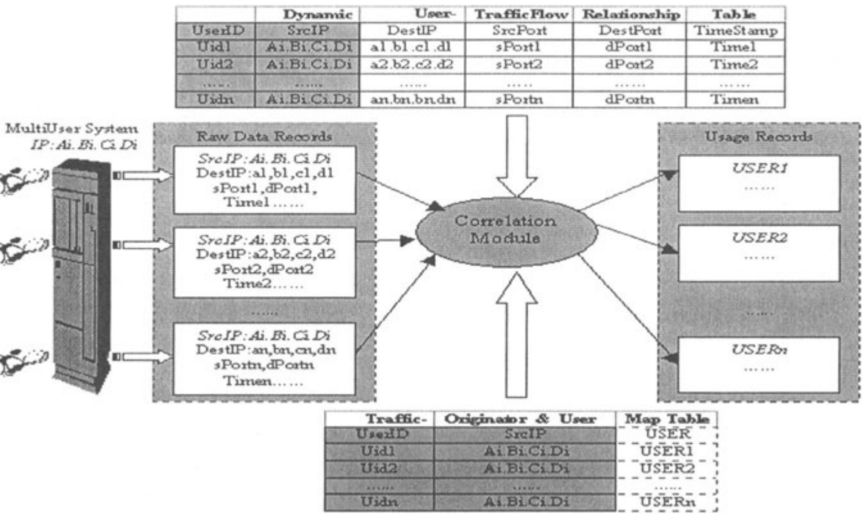
1. Capturing packages or traffic flows and extracting the Correlation attribute items from them.
2. According to the requirement of the User Model, retrieving the corresponding TO attribute items of the traffic flow from the system.
3. Combining the TO attribute and the Correlation attribute items together to generate a record into the DUTRT.
4. If the Agent works as a standalone meter, it will collect more accounting information of traffic flows and record them into DUTRT. In this case the entries of DUTRT will be used as RDRs.
5. Transferring DUTRT records to IP Mediation layer.

3.2.4 Correlation processing

The Correlation process is the same as it in traditional IP accounting systems, except that an additional Dynamic User-TrafficFlow Relationship Table, combining with a Traffic-Originator & User Map Table, will be used to map the RDRs to the corresponding users. The DUTRT is generated by the Agent, and it is used to identify the TO of each flow. The Traffic-Originator & User Map Table is used to record TO and User's static relationship. Whenever a new user account is created in a multi-user system, a new entry will be added into this table. It is managed by the IP Mediation layer. The Traffic-Originator & User Map Table will not be changed unless the user account is changed in a multi-user system.

Figure 6 illustrates an example of the correlation processing with User oriented IP accounting method.

From the Figure 6 we know, although the users in the multi-user system produce the RDRs all with the same source IP address (Ai.Bi.Ci.Di), the RDRs include other correlation attribute items<**Source IP, Source Port Number, Destination IP, Destination Port Number, Timestamp**>. With the DUTRT and the Traffic-Originator & User Map Table, the Correlation Module can correlate the RDRs to the corresponding USERs and generate the Usage Records with correct user information. Comparing with the IP Address & User Map Table in Figure 4, this method uses a DUTRT and a Traffic-Originator & User Map Table. The DUTRT includes more detailed TO information to help distinguish all traffic flows' producers in the multi-user systems.



	Dynamic	User-	TrafficFlow	Relationship	Table
UserID	SrcIP	DestIP	SrcPort	DestPort	TimeStamp
Uid1	Ai.Bi.Ci.Di	a1.b1.c1.d1	sPort1	dPort1	Time1
Uid2	Ai.Bi.Ci.Di	a2.b2.c2.d2	sPort2	dPort2	Time2
......	......	......	......	......	......
Uidn	Ai.Bi.Ci.Di	an.bn.bn.dn	sPortn	dPortn	Timen

Traffic-	Originator & User	Map Table
UserID	SrcIP	USER
Uid1	Ai.Bi.Ci.Di	USER1
Uid2	Ai.Bi.Ci.Di	USER2
......	......	......
Uidn	Ai.Bi.Ci.Di	USERn

Figure 6. User oriented IP accounting method correlation process in multi-user system

3.2.5 Traffic-Originator information storing and transporting methods

After the collection of the TO information, the next consideration is how to store and transport these TO information.

The legacy accounting protocols [1] such as Radius, Tacacs+ and SNMP etc. can be used to work together with the Agent to implement the User oriented IP accounting. For example, according to the above described User oriented IP accounting principle, the realization of the Agent can be designed as a SNMP agent in the multi-user system. At first the collected TO information will be stored into MIB database, then the SNMP protocol can be used to transport these TO information data in the MIB database to the meters. Using this method, a user oriented IP accounting MIB standard should be defined. The [7], [8] described standards can be modified to meet this requirement.

Another method called protocol header method has been discussed in [9]. The principle of this method is to utilize the option field of the IP protocol header to carry the TO information, which will be inserted by the Agent. By this means, no DUTRT is needed, and the main function of the Agent is only to identify the producer of the IP traffic, and then to add the TO information into IP packages. The TO information will not be stored in the multi-user system. Outside the multi-user systems, the IP traffic meter can collect the IP traffic's TO information directly from the protocol headers of the IP packages.

The advantage of utilizing the legacy accounting protocols is that these accounting protocols are widely accepted, but they will cause more overhead to the multi-user system and the network. The advantage of protocol method is that it will cause less overhead to the multi-user system and network. But the disadvantage is that this method needs the IP protocol to be modified, and security of the user information included in the IP protocol header is also a problem. Therefore the later method is considered to be unpractical.

3.2.6 Realization of Agent Method of User Oriented IP Accounting

According to the principle of Agent method of User oriented IP accounting, the key of the realization of user oriented IP accounting in multi-user systems is the realization of the Agent, which can generate the DUTRT. In order to collect the corresponding TO information of IP traffic flows, the Agent must locate in the multi-user system, outside the multi-user system no mechanism can obtain the TO information of the IP traffic alone.

Because the Agent needs to obtain the TO information of the IP traffic, usually the realization is OS dependent, in other words it is OS kernel dependent. For example, usually the TCP/IP drivers are implemented in kernel mode. Here we consider about two realization methods:

1. Kernel modification.

The principle of this method is, directly modifying the tcpip driver, inserting the Agent function of the user oriented IP accounting into the driver. By this means, the build-in user oriented IP accounting Agent can generate the DUTRT. Because the Agent is located in the tcpip driver, it can check all IP traffics and obtain the corresponding TO information. It can be describe as Figure 7.

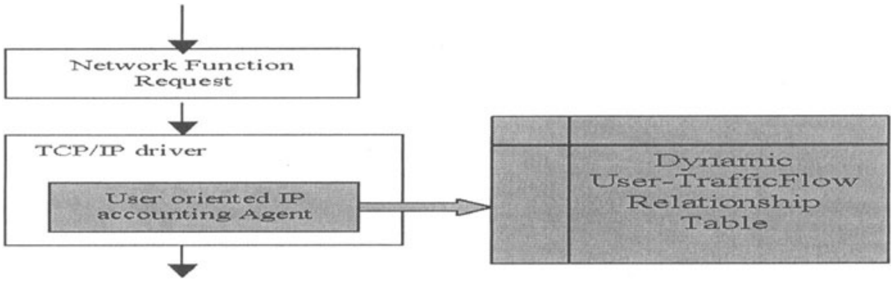

Network Function Request
TCP/IP driver
User oriented IP accounting Agent
Dynamic User-TrafficFlow Relationship Table

Figure 7. Principle of kernel modification method

This method is based on this precondition: the OS source code can be obtained and modified. It is fit for OS producer to make this modification, or for open source code OS (e.g. Linux).

[10] has implemented UserIPAcct in Linux, which is a User oriented IP accounting realization. In UserIPAcct the tcpip driver is modified and strengthened to record the RDRs with TO information.

2. Kernel patch.

The principle of this method is, making the network requirements to tcpip driver be redirected to the user oriented IP accounting Agent. This method need not modify the system kernel, the Agent will be realized as a kernel patch. Figure 8 illustrates the principle of this method.

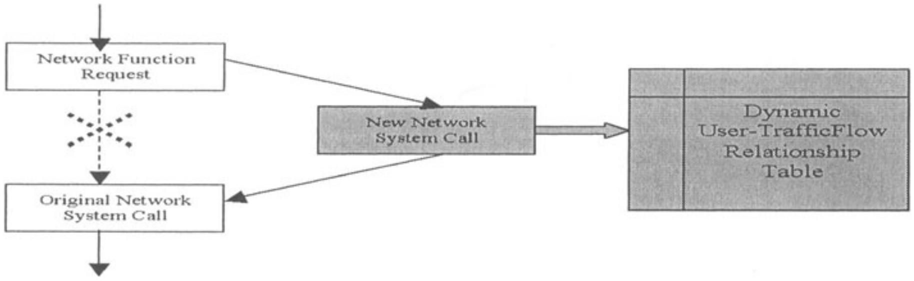

Network Function Request
New Network System Call
Dynamic User-TrafficFlow Relationship Table
Original Network System Call

Figure 8. Network system call redirection

In the redirection technique, the request to original network function will be redirected to the new defined network system call, which can capture the network traffic flows and record the traffic and corresponding TO information to generate the DUTRT.

This method is fit for the non-OS producers, who cannot get the OS source code.

In our NIPON project we have implemented the prototype Agent software IPTrafficRecorder (IPTR) respectively in Solaris and Windows 2000 Server operating systems, it can collect IP traffic and corresponding TO information to generate the DUTRT. Figure 9 is a screenshot of the prototype Agent software IPTrafficRecorder's running under the Windows 2000 server. Here we can see that each package is identified by a Traffic-Originator. In the realization of the IPTrafficRecorder in Windows 2000 Server, an Agent, the IPTR driver, works above the tcpip driver, and captures all the network request to the tcpip driver, and then it extracts the traffic information and the corresponding TO information.

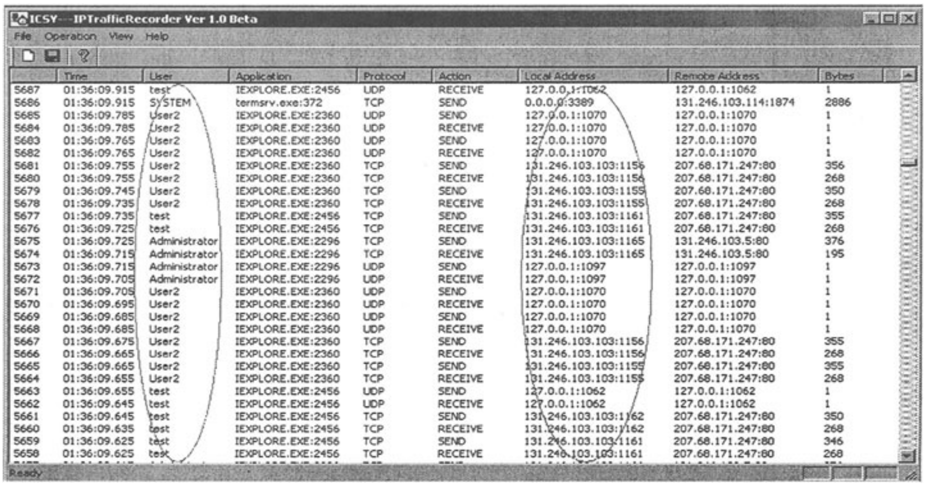
Comparing the two above described user oriented IP accounting realization methods with each other, the kernel modification method is a better solution. Because in this method, the Agent works in the tcpip driver, all the IP traffic related operation can be traced and recorded. But for the kernel patch method, since it works outside the tcpip driver, some in the tcpip driver fulfilled IP traffic related operations cannot be recorded. For example, the three-way handshake of the tcp connection is completed in the tcpip driver, the kernel patch method cannot capture the packages related with this process. For the kernel patch method, it can meter most of the IP traffic, and it is a simple method without modifying the kernel code.

The collection and transferring of the TO information of IP traffic flows will cause overhead to the multi-user system and the network. There are some ways to reduce the performance affection:

1. Agents collect only TO and Correlation attribute information. Other accounting information such as bytes count etc. will be collected by the outside meters. This can relieve the load to the multi-user system, and also

can reduce the transferring data volume from multi-user system to IP mediation system. Agents will not be used as standalone meter.

2. Using kernel modification method to implement the Agent. This will improve the efficiency of the Agent.
3. User oriented IP accounting may be configured as an optional function for the multi-user systems. If this function is not needed, or IP address can be regarded as user, the User oriented IP accounting Agent needs not be started.



	Time	User	Application	Protocol	Action	Local Address	Remote Address	Bytes
5687	01:36:09.915	test	IEXPLORE.EXE:2456	UDP	RECEIVE	127.0.0.1:1062	127.0.0.1:1062	1
5686	01:36:09.915	SYSTEM	termsrv.exe:372	TCP	SEND	0.0.0.0:3389	131.246.103.114:1874	2886
5685	01:36:09.785	User2	IEXPLORE.EXE:2360	UDP	SEND	127.0.0.1:1070	127.0.0.1:1070	1
5684	01:36:09.785	User2	IEXPLORE.EXE:2360	UDP	RECEIVE	127.0.0.1:1070	127.0.0.1:1070	1
5683	01:36:09.765	User2	IEXPLORE.EXE:2360	UDP	SEND	127.0.0.1:1070	127.0.0.1:1070	1
5682	01:36:09.765	User2	IEXPLORE.EXE:2360	UDP	RECEIVE	127.0.0.1:1070	127.0.0.1:1070	1
5681	01:36:09.755	User2	IEXPLORE.EXE:2360	TCP	SEND	131.246.103.103:1156	207.68.171.247:80	356
5680	01:36:09.755	User2	IEXPLORE.EXE:2360	TCP	RECEIVE	131.246.103.103:1156	207.68.171.247:80	268
5679	01:36:09.745	User2	IEXPLORE.EXE:2360	TCP	SEND	131.246.103.103:1155	207.68.171.247:80	350
5678	01:36:09.735	User2	IEXPLORE.EXE:2360	TCP	RECEIVE	131.246.103.103:1155	207.68.171.247:80	268
5677	01:36:09.735	test	IEXPLORE.EXE:2456	TCP	SEND	131.246.103.103:1161	207.68.171.247:80	355
5676	01:36:09.725	test	IEXPLORE.EXE:2456	TCP	RECEIVE	131.246.103.103:1161	207.68.171.247:80	268
5675	01:36:09.725	Administrator	IEXPLORE.EXE:2296	TCP	SEND	131.246.103.103:1165	131.246.103.5:80	376
5674	01:36:09.715	Administrator	IEXPLORE.EXE:2296	TCP	RECEIVE	131.246.103.103:1165	131.246.103.5:80	195
5673	01:36:09.715	Administrator	IEXPLORE.EXE:2296	UDP	SEND	127.0.0.1:1097	127.0.0.1:1097	1
5672	01:36:09.705	Administrator	IEXPLORE.EXE:2296	UDP	RECEIVE	127.0.0.1:1097	127.0.0.1:1097	1
5671	01:36:09.705	User2	IEXPLORE.EXE:2360	UDP	SEND	127.0.0.1:1070	127.0.0.1:1070	1
5670	01:36:09.695	User2	IEXPLORE.EXE:2360	UDP	RECEIVE	127.0.0.1:1070	127.0.0.1:1070	1
5669	01:36:09.685	User2	IEXPLORE.EXE:2360	UDP	SEND	127.0.0.1:1070	127.0.0.1:1070	1
5668	01:36:09.685	User2	IEXPLORE.EXE:2360	UDP	RECEIVE	127.0.0.1:1070	127.0.0.1:1070	1
5667	01:36:09.675	User2	IEXPLORE.EXE:2360	TCP	SEND	131.246.103.103:1156	207.68.171.247:80	355
5666	01:36:09.665	User2	IEXPLORE.EXE:2360	TCP	RECEIVE	131.246.103.103:1156	207.68.171.247:80	268
5665	01:36:09.665	User2	IEXPLORE.EXE:2360	TCP	SEND	131.246.103.103:1155	207.68.171.247:80	355
5664	01:36:09.655	User2	IEXPLORE.EXE:2360	TCP	RECEIVE	131.246.103.103:1155	207.68.171.247:80	268
5663	01:36:09.655	test	IEXPLORE.EXE:2456	UDP	SEND	127.0.0.1:1062	127.0.0.1:1062	1
5662	01:36:09.645	test	IEXPLORE.EXE:2456	UDP	RECEIVE	127.0.0.1:1062	127.0.0.1:1062	1
5661	01:36:09.645	test	IEXPLORE.EXE:2456	TCP	SEND	131.246.103.103:1162	207.68.171.247:80	350
5660	01:36:09.635	test	IEXPLORE.EXE:2456	TCP	RECEIVE	131.246.103.103:1162	207.68.171.247:80	268
5659	01:36:09.625	test	IEXPLORE.EXE:2456	TCP	SEND	131.246.103.103:1161	207.68.171.247:80	346
5658	01:36:09.625	test	IEXPLORE.EXE:2456	TCP	RECEIVE	131.246.103.103:1161	207.68.171.247:80	268

Figure 9. IPTrafficRecorder in Windows 2000 Server

The overhead caused by User oriented IP accounting is unavoidable, because the user information is invisible outside the multi-user system. What we can do is to lessen the performance affection to the multi-user system and the network caused by the Agent.

4. SUMMARY

In this paper we have presented a user oriented IP accounting technology in multi-user systems. It can provide more accurate accounting information than the traditional IP address oriented accounting technology, and it extends the traditional IP accounting technology.

User oriented IP accounting utilizes an Agent to collect TO information of the IP traffic from the multi-user systems, these TO information will then be stored in the Dynamic User-TrafficFlow Table, which can be used to correlate the user with the IP traffic. The extended legacy accounting protocol methods can be used to convey the TO information. In realization of the user oriented IP accounting, two methods, kernel modification method and kernel patch method, have been suggested.

Comparing with the two methods, the kernel modification method is a more precise method, which can collect the required accounting information more completely.

The suggested User oriented IP accounting architecture is based on the traditional IP accounting system. It is an extension of the traditional IP accounting architecture. The traditional IP accounting meters will be used to collect accounting information of single user systems, and the Agent will be used to collect accounting information in multi-user systems. Now existent IP accounting systems can enhance its IP accounting ability in multi-user systems without influencing its ability in single user systems.

In our NIPON project we have implemented a user oriented IP accounting prototype with the kernel patch method in Solaris and Windows 2000 server respectively. In the future we will develop a user oriented IP accounting system in the computer center of University of Kaiserslautern, and this user oriented IP accounting system will mainly run in the Solaris, Linux and Windows 2000 Server. And the kernel patch method will be used to implement this. The kernel modification method maybe a suggestion for the OS producers. To realize the kernel modification method, some standards of the user oriented IP accounting should be defined.

5. REFERENCE

[1] B.Aboda,J.Arkko, D. Harrington: "Introduction to Accounting Management", RFC2975, October 2000

[2] Ge Zhang, "Comparison and Analysis of IP billing Technologies", Internal Report, University of Kaiserslautern, November 2001

[3] Lucent Technologies, BILLDATS® Data Manager, http://www.lucent.com/

[4] S.Handelman, S. Stibler, N. Brownlee, G. Ruth: "RTFM: New Attributes for Traffic Flow Measurement", RFC2724, October 1999

[5] Nevil Brownlee, "Using NeTraMet for Production Traffic Measurement", Integrated Management Strategies for the New Millennium, December 5, 2001

[6] N.Brownlee,A.Blount: "Accounting Attributes and Record Formats", RFC2924, September 2000

[7] N. Brownlee, C. Mills, G. Ruth: "Traffic Flow Measurement: Architecture", RFC2722, October 1999

[8] N. Brownlee: "Traffic Flow Measurement: Meter MIB", RFC2720, October 1999

[9] Volker Bauer: "Analyse von Netwerk-Abrechnungs-Systemen bezueglich nutzerorientierter Datanerfassung", Diplomarbeit, Univeritaet Kaiserslautern, September 2000

[10] Lars Fenneberg, et. al., "UserIPAcct - a program to do per user ip accounting", http://ramses.smeyers.be/homepage/useripacct/

TARIFF-BASED PRICING AND ADMISSION CONTROL FOR DIFFSERV NETWORKS

Tianshu Li, Youssef Iraqi and Raouf Boutaba
School of Computer Science
University of Waterloo, Canada
{dtianshu,iraqi,rboutaba}@bbcr.uwaterloo.ca

Abstract: In a QoS-Enabled network environment, there are two major concerns from both user's and provider's points of views: are there enough resources available for a particular traffic flow and what's the price for this flow? These two questions are exactly what admission control and pricing try to answer. An architecture that integrates pricing and admission control seems very promising. In this paper, we propose a tariff-based pricing architecture that integrates pricing and admission control for the DiffServ networks. We also study some pricing setting strategies for our architecture and evaluate our strategies through simulations.

Keywords: Pricing, Admission Control, DiffServ

1. Introduction

With the Internet evolving into a multi-service network, QoS-pricing in the Internet has been one of the hottest research areas in the recent years. Two QoS architectures: Integrated Services (IntServ)[1] and Differentiated Services (DiffServ)[2] have been standardized by the IETF to support QoS in the future Internet. Due to the inherent scalability problem of the IntServ approach, it is generally believed that DiffServ is more likely to be implemented in the Internet core. Unlike IntServ, which can charge users based on the allocated resources, pricing for DiffServ networks is more complicated and has drawn a lot of attention in the networking community. Before the wide deployment of DiffServ, an effective and efficient pricing scheme has to be developed. Meanwhile, since price is such an important economic incentive for the end users, pricing is often considered as an effective mechanism for congestion control and admission control, which in turn can improve the level of QoS guarantees. Indeed, QoS pricing schemes proposed so far often entail either congestion control or admission control or even both. In this paper, we first have a close look at the relationship between the pricing and these two traffic management functions and propose a tariff-based pricing architecture that integrates pricing and admission control for DiffServ networks. The proposed architecture maintains domain and global price tables for core networks only. In this way, we decouple the pricing for the core network from the end-to-end pricing, which fits well into the DiffServ paradigm.

The remainder of this paper is organized as follows: Section 2 will review some background and related work in this area. Section 3 discusses the motivation and some design choices and presents our pricing architecture and the construction and maintenance of pricing tables. Section 4 and 5 discuss our price setting strategy and admission control scheme in details. Section 6 presents our simulation results and their analysis and finally section 7 concludes the paper.

2. Background and Related Work

Pricing for the Internet in general has long been an active research area. Example approaches such as proposed in [6–10] study the pricing for networks from various angles. More detailed review of pricing schemes can be found in [3, 4]. Most of approaches either assume a well-known user utility function or try to create a market environment for auctioning. In the first case, existence of a prior known utility function is assumed and price setting can be based on the optimization that maximizes either the social welfare globally or the user benefit locally. However, in [10], Shenker et al. argued that utility functions could not be well defined in short term and sometimes even very difficult in a long-term time scale. The effectiveness of such schemes is still questionable. Based on this observation, they proposed the Edge Pricing scheme that charges users for the estimated path and estimated cost so that pricing is pushed to the edge. In the second case, although auctioning does not require a prior knowledge of user traffic characteristics and has been generally considered as the one that achieves economic efficiency, it has significant implementation overhead. Up until now, there has not been a well-accepted solution yet.

Many of approaches mentioned above assume that users are rational to the price signals and have been using the pricing as a main mechanism for congestion control. When congestion occurs, extra congestion cost will be charged in order to address the externality issue. Since users are expected to react to the price signals, congestion-sensitive pricing schemes often emphasize user adaptation where users will adjust their sending rate in case of congestion. Some approaches that fall into this category can also be found in [12, 14].

However, in a strict sense, congestion-sensitive pricing does not address the QoS guarantee in particular. When congestion occurs, QoS is no longer guaranteed. To provide a better QoS guarantee, what we want is to avoid the congestion, not to act after the congestion occurs. Furthermore, QoS guarantee is a commitment that service providers made to the end users. Asking users to adapt to the price change or even terminate the service is undesirable. We believe that a proper interpretation for the user adaptation in a DiffServ environment is the choice of different service classes at the beginning of a service session rather than adjusting their sending rate in the middle of a service session. In other words, an elastic request would likely choose a lower level service class while an inelastic request may choose a higher level service class if the budget is sufficient. If the budget is not sufficient, then a request can either lower the service class requirement (if it is tolerable) or decide not to enter the network at all.

Another traffic management function that is closely related to the pricing is Admission control (AC). AC is often used to control the network load by restricting the access to the network and hence improve the level of QoS guarantee. Example approaches can be found in [13–16]. Studies also show that simple admission control algorithms based on estimated or measured network status are generally robust [15]. As being done in IntServ/RSVP architecture, admission control traditionally is performed at a hop-by-hop basis [1]. However, in a DiffServ environment, adding admission control functionality to all the core elements violate the DiffServ principle of keeping the core simple. End-point/edge admission control that pushes the admission control functionality into the edge of the network seems more suitable in this case. Most of the end-point AC approaches use probing [13, 16] or explicit congestion notification (ECN) [15, 16] to convey the network status back to the end points. A comprehensive study on endpoint admission control can be found in [16].

However, so far, most of the studies consider the pricing and admission control as two separate management functions. In other words, admission decisions are made solely on the load measurement or estimation and have no direct relation with the pricing. Using price as a primary admission criterion has not been studied sufficiently. In [15], authors suggest that admission decision could be made based on the user's willingness to pay for the ECN mark. However, it is not clear how users should pay for the mark (i.e. what is the price for the mark). In [12], Wang and Schulzrinne presented a complete pricing framework that integrates the admission control, congestion control, and pricing for DiffServ networks. However, they focus mainly on the congestion-sensitive pricing and do not study the admission control in details. Admission control in their framework is performed hop-by-hop and independently from pricing. Our architecture is similar to what they proposed but differs in a number of ways. This will be discussed in the subsequent sections in details.

3. Pricing Architecture

3.1 Motivation and Design Choices

From the discussion so far and the implication of our view of user adaptation, it is more desirable to tie the pricing with admission control in a DiffServ environment. Indeed, in a QoS-Enabled network environment, there are two major concerns from both user's and provider's points of views: are there enough resources available for a particular traffic flow and what's the price for this flow? These two questions are exactly what admission control and pricing try to answer. An architecture that integrates pricing and admission control is therefore very promising.

The design of our architecture is based on the following considerations.

- *Decouple the pricing for core networks from the access networks and end users:* Due to the scale and complexity of the Internet, a practical QoS implementation in the Internet is to deploy DiffServ in the core networks and give the access networks the freedom of implementing IntServ, DiffServ, or other QoS techniques [11]. This implies that a tightly integrated end-to-end pricing scheme is unlikely to be accepted. Furthermore, a large number of service providers are involved in the pricing of the Internet and each of them should have the choice of their own pricing scheme. A single pricing model that suits all is very impractical. Therefore, we emphasize a separate pricing scheme for the core networks that implements DiffServ technology. This decoupling enables us to focus on the network core only.

- *Price should reflect the availability of the network resources in the core:* As mentioned above, the ultimate goal of QoS is to avoid congestion. Therefore, it is more meaningful to charge users based on the remaining resources rather than the congestion cost. Since we are more interested in the implementation side of the problem, we do not assume a well known utility function for setting the optimal price. In other words, each network element sets the price merely based on its own load. The rarer the resource, the higher the price. We also follow a simple and intuitive rule about price setting: price changes very slowly when there are plenty of available resource and increases drastically when the resource is scarce.

- *Per-flow messaging is not acceptable in a large network such as the Internet:* In order to aggregate and convey the price information to the access networks or even the end-users, we need a flexible and efficient architecture to accumulate the price along the path. In [12] a signaling protocol called resource negotiation and pricing protocol (RNAP) is proposed to accumulate the price along the path and negotiate the price and resource with the end-users. It is easy to see that the major drawback of such an approach is the per-flow messaging. Although the possibility of aggregating the messages has been investigated, the control message overhead is still significant. Our architecture tries to avoid this problem by maintaining the global price tables for the core networks at the access networks. This aggregates the pricing messages significantly.
- *Edge/end-point admission control fits well in a DiffServ environment:* Since the price reflects the availability of the resources in the core, admission decisions can mainly or even purely be based on the price (in this case, the price is the only admission criterion). Thanks to the decoupling of pricing for core networks, we are able to maintain the global price table at the access networks via domain abstraction. This enable us to push the admission control to the access networks. In our architecture, it is the end-users or access networks that decide whether a flow should enter the network or not. Inter-domain admission control is also possible using a global price table.

3.2 Price Table Construction

There are two types of price tables in our architecture: domain price table and global price table. A price entry in the domain price table represents the price for a service class from one edge node to another edge node within a domain, where a price entry in the global price table represents the price for a service class from one domain to another domain. Maintaining price information at such a scale may sound very impractical at a first glance. However, by abstracting an entire domain into a single node in a multi-domain network, constructing a global price table becomes feasible. Figure 1 depicts the concept of domain abstraction.

Semret et al. [9] also use a similar type of abstraction in their pricing scheme. A global market is created for all the domains to bid for the capacity in order to provide a QoS guaranteed service in their own domain. In their abstraction, overall capacity requirements in a domain is abstracted into a single bottleneck capacity. Although this is certainly a valid assumption, some inaccuracy is still introduced into the abstraction. In our abstraction, we do not make any simplification or assumption but rather accumulate the price for each actual ingress-egress path.

3.2.1 Domain price tables. Once each network element calculated the price locally, the price information has to be exchanged and aggregated over the entire domain. The ultimate goal here is to accumulate the price for each ingress-egress pair. Therefore, we need to know the route from the ingresses to the egresses. This requires

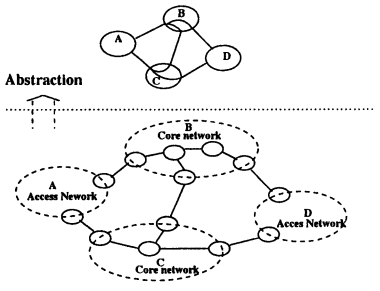


Figure 1. Domain Abstraction

Domain Price Table				
Route			**Price**	
Ingress	**Egress**	**Intermediate**		
a1	a5	a2 , a4	class 1	45
			class 2	...
			...	...
a1	a3	a2	...	
...	...	...	...	

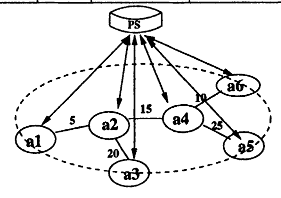


Figure 2. A Domain Price Table

some knowledge of the domain topology and routing table information within the domain. The choice of using a centralized or a decentralized approach largely depends on the routing strategy used in the domain.

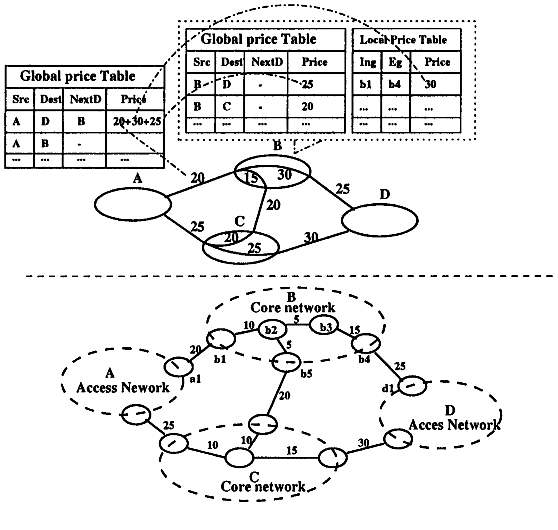
If a link state routing approach such as OSPF or IS-IS is used, a centralized approach is preferred since all the information required to construct the domain price table is available immediately. Centralized approach uses a pricing station for each domain or autonomous system. The pricing station will communicate with all network elements within the domain and collect the price information. As a result, the pricing station will eventually maintain a price table for each ingress-egress pair. When an ingress-egress pair has multiple possible routes, the domain routing table is consulted and the route from the routing table will be used. Figure 2 depicts a sample domain price table maintained in a centralized pricing station (PS).

It is relatively complicated if the distance-vector routing approach such as RIP is used within the domain. A distributed approach can be considered in this case. One alternative is to add price as an extra set of metrics into the routing table, However, the major drawback of this approach is to bind the price update with the route update. A route update implies a price update but not the other way around. Furthermore, to propagate the price information among domains and construct the global price table, it's relatively easier when a centralized and complete view of the entire domain (e.g. a whole set of ingress-egress pairs) is available. One naive centralized approach in this case is to ask each element to send its routing table to the pricing station whenever the route update happens. Pricing station is then able to collect all the route information and generate the routes for each ingress-egress pair. A centralized pricing station is also well compatible with the Bandwidth Broker (BB)[5] approach. In fact, the pricing station can be part of the BB functionality. In the rest of the paper, we assume that a centralized pricing station is used. We also assume a pricing interval in our pricing architecture for both domain and global price tables. However, the update of the domain price table is not done periodically but is rather change driven to reduce the control-message overhead.

3.2.2 The Global price table. Because of the abstraction mentioned earlier, we are now able to construct the global price table without introducing too much overhead.The price for an ingress-egress pair inside a domain becomes the price for passing through the node in the abstracted global network. Different routes that take different ingress-egress pairs through one domain will most likely have different prices. In this way, we can view the whole core networks as a single network that contains a limited number of nodes and links. Of course, further hierarchical decomposition can be applied if the size of global price table is still too large. In this paper, to give a simple and clear view of the idea, we assume only one level of abstraction and one level of global price table is constructed.

Unlike the domain price table, a global price table has to be constructed and maintained in a distributed manner. The update of the global price tables is also different. Periodical advertisement is used to propagate the price information as done in the Border Gateway Protocol (BGP). Each pricing station will advertise the price of a ingress-egress pair to its interested neighboring pricing station. Upon receiving such update information, each pricing station will update the global price table accordingly and propagate the update information to its interested neighbors at next pricing interval.

To deal with the issue of multiple routes through different domains, inter-domain routing tables are consulted during the price propagation. However, this time, we do not require a complete view of the route because the only information we want is the next domain or autonomous system which can be easily obtained from the BGP routing table (assuming that BGP is used for inter-domain routing). Figure 3 depicts the construction of global price table.


Global price Table

Src	Dest	NextD	Price
A	D	B	20+30+25
A	B	-	
...	...	...	...

Global price Table

Src	Dest	NextD	Price
B	D	-	25
B	C	-	20
...	...	...	...

Local Price Table

Ing	Eg	Price
b1	b4	30
...	...	...
...	...	...

Figure 3. Global Price Table in Access networks

The possibility that paths in the routing table may change after the advertisement has a potential impact on the pricing. An admission decision based on the price of a route may become invalid if the path in the routing table changes (packets will eventually take a different route than the route they are expected to take and on which the admission decision was based). In this case, service quality may not be guaranteed especially if the new path does not have enough resources. Another concern is that the price agreed by user and service provider is no longer valid. In this case, if there are enough resources in the new route, then we believe that service providers should absorb this price difference and simply continue the service. If the new route does not have enough resources, one possible solution is that service providers will notify the end-users to terminate the session without any charge. Since the price for the new path will be available "immediately", it is also possible to start a service renegotiation.

4. Price Setting Strategy

Now, the question left is how to set the price to reflect the availability of the network resources. Each network element will incorporate a load monitor so that price can be based on its current load level. Since network monitoring is out of the scope of this paper, we assume an existing method to monitor the traffic load for each service class.

We assume a basic unit price per unit time P_{unit} that can be computed offline for each service class. This basic unit price reflects the equipment costs, maintenance/administrative costs and business revenue consideration for a network element. The idea of differentiated service is to have a small number of classes where a higher price service class attracts less traffic and consequently have a better QoS guarantee. Hence, we consider a simple and practical approach where service providers set up a targeted capacity fill factor f_i for each class to enable the service differentiation between the classes(i.e. $f_i = T_i/C^i_{max}$, where T_i is the targeted capacity for class i and C^i_{max} is the maximum capacity for class i). Therefore, it is straightforward to see that the base price for a service class P^i_{base} is inversely proportional to this factor.

$$P^i_{base} = P_{unit}/f_i \tag{1}$$

To compute the dynamic price for a service class, Wang and Schulzrinne adopt an iterative tatonnement process in [12]:

$$P_i(t) = P_i(t-1) + a_i * (D_i - T_i)/T_i$$

Where $P_i(t)$ denotes the price for class i at time t, and D_i is the demand or current load for class i and a_i is the convergence rate factor. However in their case, the price changes gradually and can not successfully reflect the real traffic condition inside the network. We believe that when the network is severely stressed the price should increase much faster. As mentioned above, we follow an intuitive way for the price setting. Figure 4 illustrates our pricing strategy in general. When the load for a particular service class is lower than its targeted capacity, the price is simply the base price P^i_{base} for a particular service class. When the load exceeds its target capacity the price will be increased rapidly and even dramatically when the load is close to the maximum capacity. we adopt the exponential growth in this case to reflect our strategy of price setting:

$$P_i(t) = \begin{cases} P^i_{base} & \text{if } D_i(t) \le T_i \\ P^i_{base}\, e^{\alpha_i[\frac{D_i(t)}{T_i} - 1]} & \text{otherwise} \end{cases} \tag{2}$$

Note that when demand exceeds C_{max}, we can simply set the price to ∞ to indicate that there is no longer available resource for new requests. Admission control can be enforced in this situation. Alternatively, there can be no price limit even if the demand exceeds the maximum capacity. P^i_{max} then is not the price upper bound but rather the

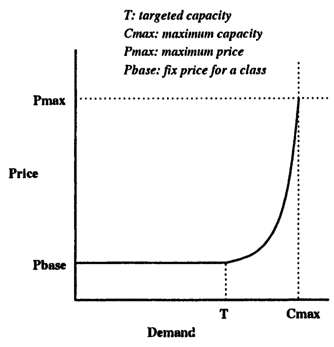


Figure 4. General price setting strategy

price when the demand reaches the maximum capacity of that particular service class. Since the price increases exponentially and becomes so high that we expect no user would actually accept the price.

There are several ways to decide the α_i factor. Here, we consider a simple case where P^i_{max} is known. One can use the base price for class $i+1$ as the P^i_{max} for class i, or alternatively decide the P^i_{max} based on some business considerations, which is more likely the case in the real world. In the later case, P^i_{max} is allowed to be greater than the P^{i+1}_{base}, which enables the switching between service classes when the price for a lower level service class is higher than the price for upper level service class and hence balances the load among service classes.

Now, assuming that a maximum price P^i_{max} is available for each service class, Since $f_i = T_i/C^i_{max}$, we can obtain the α_i factor by solving the following equation.

$$P^i_{max} = P^i_{base}\, e^{\alpha_i[\frac{C^i_{max}}{T_i}-1]} \Longrightarrow \alpha_i = \log(\frac{P^i_{max}}{P^i_{base}}) * (\frac{f_i}{1-f_i}) \tag{3}$$

5. End-to-End Pricing and Admission Control

5.1 End-to-End Pricing

One of the main advantages of our pricing framework is that it enables us to focus on the pricing in the core networks only. Depending on the particular technology implemented in the access networks, various pricing schemes can be applied to achieve the end-to-end pricing. For example, access networks can implement flat rate pricing or time of day pricing for their simplicity and predictability. In this case, the cost of accessing core networks could be absorbed by the access networks and it is the access networks that choose an appropriate service class for the user traffic based on some policy defined by the access networks. Alternatively, access networks can charge user for the reserved resource if IntServ is implemented or even use the same pricing scheme as in the core if DiffServ is used.

Since the pricing is strictly for the network core and global price tables are available at the access networks, end-to-end pricing can be implemented without involving the core nodes at all. This eliminates possible scalability problems caused by pricing and admission control signaling. it also gives the access networks the flexibility of implementing an end-to-end pricing scheme in different protocol layers. For example, if users or access networks are really keen on the exact end-to-end pricing, a light weight signaling protocol can be used between the sender and receiver access networks without any involvement of the core nodes. This protocol can be implemented in the network layer as the pricing for core networks does or even in the upper layers such as the transport layer or the application layer.

5.2 Admission Control

Another desirable property of our pricing framework is that the admission control decision can be made based on this price information since it effectively reflects the availability of the path. Our admission control algorithm then consists of the following two parts when a flow request is generated:

1 Lookup the price for a service class in the global price table. End-users or access networks decide whether the price is acceptable or not based on their budget constraints and other service level agreement (SLA) parameters. If the price can not be accepted then check the possibility of switching among service classes (with possibly different service requirements).

2 If the advertised price is accepted, then final admission control decision is made by the service providers based on the estimated bottleneck traffic load l_{max} against the targeted threshold for the service class. This deals with the situation that when users are willing to pay for the service but there is no resource available.

Ideally, we want to know the relationship between the total price and the available resource at the bottleneck link. In other words, we are most interested in the relationship between the total price $P^i_{total}(t)$ and the load of bottleneck link. From section 4 we know that the accumulated price at time t for class i observed at the edge is:

$$\begin{aligned} P^i_{total}(t) \quad &= \quad \sum P^{ij}_{base} + \sum P^{ik}_{base}\, e^{\alpha_{ik}\,[\frac{D_{ik}(t)}{T_{ik}}-1]}\,, (D_{ij}(t) \leq T_{ij}, D_{ik}(t) > T_{ik}) \\ & \quad\quad \text{where } D_{ij}(t) \text{ is the demand for class } i \text{ at link } j \text{ at time } t \end{aligned} \tag{4}$$

Given only the value of $P^i_{total}(t)$, it is almost impossible to obtain the exact value of the max $[D_{ij}(t)/T_{ij}]$. However, in our pricing scheme, $P_i(t)$ at the bottleneck link becomes the dominant term in the $P^i_{total}(t)$ when it reaches a fairly high level. This is mainly because of the exponential growth of $P_i(t)$ and it enables us to estimate the value of the max $[D_{ij}(t)/T_{ij}]$ with much less error.

For the sake of understanding, in the rest of this section, all notations are for service class i unless otherwise specified. Let $l_j(t)$ denotes the traffic load at link j at time t (i.e. $l_j(t) = D_j(t)/T_i$), and $l_{max}(t)$ denotes the load for the link that has the highest load along the entire path). From equation 4, we have

$$P_{total}\,(t) = \sum P^j_{base} + \sum P^k_{base}\, e^{\alpha_k\,(l_k(t)-1)} + P^h_{base}\, e^{\alpha_h\,(l_{max}(t)-1)} \tag{5}$$

where h is the link that has the highest load $l_{max}(t)$ at time t along the path and the last term in the equation is the price for the bottleneck link h. The fist term is the sum of the price for all the links that have the base prices and the second term is the sum of the price for the rest of the links respectively. Then a bound is given by,

$$P_{total}\,(t) \geq \sum_{all} P_{base} - P^h_{base} + P^h_{base}\, e^{\alpha_h (l_{max}(t)-1)} \tag{6}$$

where $\sum_{all} P_{base}$ is the sum of the P_{base} for all the links along the path, which is denoted as P^{total}_{base}. By rearranging the terms in the equation and plugging in the value of α_h from equation 3, we can see that $l_{max}(t)$ must satisfy the following relation.

$$l_{max}(t) \leq 1 + \frac{\log\left(\frac{P_{total}\,(t) - P^{total}_{base}}{P^h_{base}} + 1\right)}{\log(\,P^h_{max}/P^h_{base}) * (\frac{f_h}{1-f_h})} \tag{7}$$

We will use the $\min\{f_h\}$, $\max\{P^h_{base}\}$, and $\min\{P^h_{max}\}$ respectively and use f, P_{base} and P_{max} to denote them. Since they are constant for the path, they can be collected and computed offline.

For the purpose of admission control, using upper bound often provides quite conservative results. To examine the relationship between $P_{total}(t)$ and $l_{max}(t)$, we first consider the case where there is a single bottleneck link along the path. Clearly, the upper bound value obtained above will be very close to the exact $l_{max}(t)$ because the rest of the terms in equation 5 are indeed ignorable. However, it is more complicated when there are multiple bottleneck links along the path. Some estimations are required to handle this situation. Since we expect that f_h, P^h_{max}, and P^h_{min} vary within a relatively small range, without loss of generality, we can rewrite equation 6 into

$$P_{total}(t) \simeq P^{total}_{base} - r * P_{base} + r * P_{base}\, e^{\alpha\,(l_{max}(t)-1)} \tag{8}$$

where r is the number of the bottleneck links along the path that all should be taken into consideration. Note that we are using the f, P_{base}, and P_{max}. However, in this case, they are not necessarily the maximum or minimum values since a number of links are involved. Further refining such as average or weighted average can be applied depending on the specific network condition. Solving the above equation gives us

$$l_{max}(t) \simeq 1 + \frac{\log\left(\frac{P_{total}\,(t) - P^{total}_{base}}{r * P_{base}} + 1\right)}{\log(\,P_{max}/P_{base}) * (\frac{f}{1-f})} \tag{9}$$

It is then clear that l_{max} most likely lies between the upper bound and the above estimated value. It is difficult to estimate the value of r without any knowledge of network topology or explicit messaging. However, for a long time scale, it is possible that each domain is able to identify the approximate number of bottleneck links and the probability of multiple bottlenecks existing inside the domain. In this way, one can compute a weighted average $\overline{r}$ value for each ingress-egress pair and exchange them offline. Of course, one can always set r to 1, which is the case of upper bound admission control. In this case, the difference of these two values also indicates the error range of our estimation. We should also notice that the error caused by the incorrect value r is reduced significantly because of the logarithmic nature of our estimation.

6. Simulation and Results

6.1 Simulation Model

In order to study the behavior of our pricing strategy, we setup a DiffServ network environment using the *ns 2* simulator. We modified the DiffServ implementation in *ns* developed by Nortel Networks to incorporate our pricing and admission control mechanisms. Since the goal of the simulation is to evaluate our price setting strategy and admission control scheme, we only simulate a single DiffServ domain and do not focus on the construction and maintenance of the domain and global price tables.

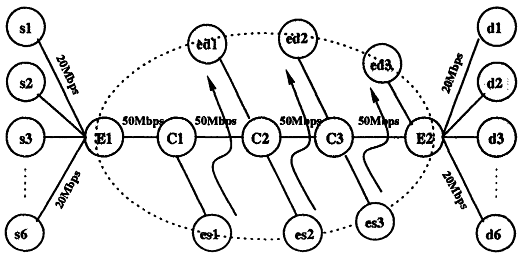


Figure 5. Simulation Network Topology

Figure 5 illustrates the network topology used in our simulation which consists of three core routers and eight edge routers. Three core routers implement the dsRED core queue which has no policing and marking functionality but only PHB forwarding. All edge routers implement the dsRED edge queue which supports the DiffServ packet classifying, marking, and policing. Edge router E1 acts as the pricing station and handles user requests generated at sources. Additionally, six extra edge routers are configured inside the DiffServ domain to create the cross traffic and simulate bottlenecks at link C1C2, C2C3, and C3E2.

The total capacity of each link from source nodes to the edge of DiffServ domain is set to 20Mbps, and 50 Mbps for all links within the DiffServ domain. Propagation delay for all the links are set to 5ms. All links are full duplex outside the DiffServ domain and DropTail queue management is used in this case. Inside the DiffServ domain, only the links that connect core routers are full duplex and the rest of the links are simplex links because different type of dsRED queuing techniques are used in different directions. Weighted Round Robin (WRR) scheduling is used in each link. In our simulation we consider three service classes and the weights for the three classes are distributed as 3, 3, and 4, and the expected load for each class is set to 50%, 70%, and 90% respectively. The base price P^i_{base} and full load price P^i_{max} for each class are set as 0.16/0.7($), 0.09/0.35 ($), and 0.04/0.16($) per time unit respectively. A pricing agent is attached to each link to set the price locally and communicate with each other to propagate the price information back to the edge. The admission threshold is set to 0.75, 0.9, and 1.1 respectively.

For each class, there are two types of traffic sources in our simulation. CBR and Pareto on/off traffic are generated independently to the edge E1 and flow requests are modelled by a Poisson arrival distribution. The holding time for each flow is exponentially distributed with a mean value of 150s. The average rate for CBR is

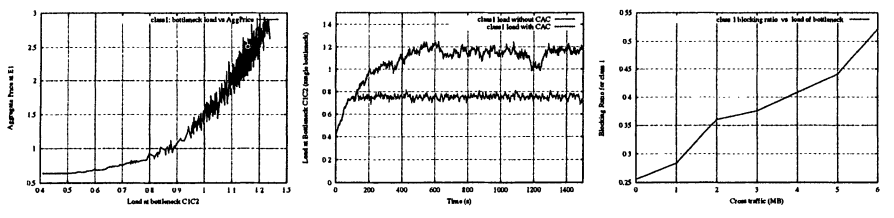


Figure 6. (single bottleneck a→c from left to right) (a)aggregated Price at E1 vs. Load at C1C2 (b)load at C1C2 without/with CAC (c)request blocking ratio vs. Traffic(Mb) at C1C2

128k. For the Pareto traffics, the shape parameter is set to 1.5, where the on and off time are both set to 500ms, and the peak rate is set to 128K. The rest of the parameters are set to the default values in *ns*. The total time for each run is 1500s.

6.2 Result Analysis

6.2.1 Single Bottleneck. We first consider the case of single bottleneck to examine the basics of our approach. The rate of cross traffic at C1C2 is set to be approximately 40% of the class 1 capacity at C1C2. The rate of the other two cross traffics are set to be less than 10% of the class 1 capacity accordingly at C2C3 and C3E3. Therefore, C1C2 will be the only bottleneck along the path.

Figure 6a gives the aggregate price observed at E1 vs. the load at C1C2 when admission control is not used. As we expected, it is a well shaped exponential curve because there is only one bottleneck and its price dominates the total price observed at E1. To test the effectiveness of our admission control algorithm, we first run the experiment without admission control and then repeat the experiment with admission control under the same traffic condition. Figure 6b shows that our estimation of network load based on the price is indeed accurate in the case of single bottleneck. The load at the bottleneck link is well controlled at about 0.75, which is the admission threshold preselected for class 1.

We repeat the simulation with different sending rates of cross traffic at C1C2. As expected, the load at C1C2 are all well controlled but with different request blocking ratios. Figure 6c shows the request blocking ratio for all the sources vs. the sending rate of cross traffic at C1C2.

Throughout the experiment, we experience very few packet losses except when the total sending rate of flows for all service classes exceeds the total capacity of the bottleneck. This is mainly due to the use of WRR scheduling and no individual dropping enforced for each class. When a class load exceeds its class capacity, it tends to steal the bandwidth from other service classes. Since we are mostly concerned about controlling the traffic load and keeping it lower than a threshold, we do not present the packet loss and delay results here. Ideally, by carefully setting the admission control threshold for each service class, we should not see any packet loss.

6.2.2 Multiple Bottlenecks. To simulate a multiple bottlenecks situation, we increase the sending rates of cross traffics at C2C3 and C3E2 to about 35% so that they are close to but still lower than the sending rate of cross traffic at C1C2. There-

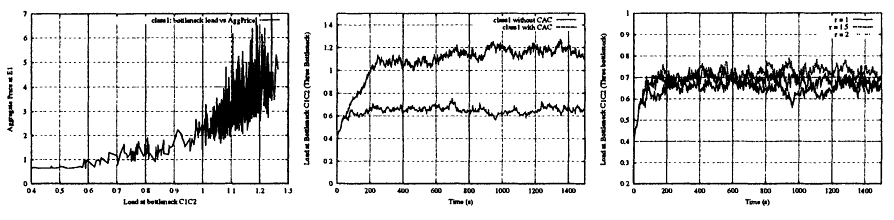


Figure 7. (three bottlenecks, a→c from left to right)(a) aggregated Price at E1 vs load at C1C2 (b) Load at bottleneck C1C2 without/with CAC (c) Load at bottleneck C1C2 with different r values

fore, three bottlenecks are simulated in our network. As in the singe bottleneck case, we first run the simulation without admission control and then repeat the simulation with admission control.

Figure 7a shows the aggregated price observed at E1 vs. the load at bottleneck C1C2 when admission control is not enforced. Because no single bottleneck can dominate the price for the entire path, the aggregated price is a little harder to predict when the links are heavily loaded. Fortunately, the fluctuation of the price does not affect the effectiveness or stability of our admission control algorithm. Figure 7b shows that the traffic load at bottleneck link C1C2 is also well controlled consistently. We do observe that the estimated load is not as accurate as in the single bottleneck. This is mainly because we are using the upper bound admission control and the result is expected to be conservative. The simulation result shows a fairly close traffic load estimation (0.7 vs. 0.75). Throughout the experiment, we can see that our approach has a very consistent performance. This indicates that our admission control approach is robust and the fluctuation of price would not affect the stability of our approach.

Based on the discussion in section 5.2, we also vary the factor r to see how it will affect our simulation results. Figure 7c shows three sets of load at bottleneck C1C2 throughout the experiment with r set to 1, 1.5, and 2. As we can see, r has a small but positive impact on the load estimation. In other words, choosing a close enough r value could indeed improve the efficiency of our admission control approach. However, the improvement is quite small. This is because the error of our estimation is reduced significantly by the logarithmic nature of our load estimation algorithm.

7. Conclusion

The main objective of having the global price table at the level of access networks is to enable an accurate and fast decision-making process. In this paper, we presented our approach to the pricing in DiffServ networks and proposed a pricing architecture that separates the pricing for core networks from the end-to-end pricing through domain abstraction and maintaining domain and global price tables. We also described our pricing strategy and suggested an associated admission control mechanism. Since the price in our scheme effectively reflects the resource availability inside the network, it is used not only as an economic incentive but also as a mean of estimating the bottleneck traffic load.

Our architecture is flexible in the sense that end-to-end pricing is decoupled from the network core and scalable thanks to the domain abstraction. Admission control is pushed to the edge and no per-flow based messaging for either pricing or admission control is needed. This way, our architecture follows the philosophy of the edge-pricing scheme but with better network utilization and QoS guarantee. Maintaining the price tables for core networks also enables the split of revenue among the service providers.

We believe that the benefit gained from maintaining the price tables can certainly overweight the overhead it introduces. Our future work includes developing a user-behavior model that models the user reaction to price change and studying the impact it may have on the performance of our admission control mechanism. We are also investigating the applicability of our architecture in other contexts. For example, QoS routing using price as a constraint may be an interesting application of our pricing architecture since the price is available immediately and reflects the availability of resources inside the network.

References

[1] S. Shenker, et al, *"Specification of guaranteed quality of service,"* RFC 2212, IETF, Sept. 1997.

[2] S. Blake et al, *"An Architecture for Differentiated Services,"* RFC 2475, IETF, Dec. 1998.

[3] M. Falkner, M. Devetsikiotis, and I. Lambadaris, *"An Overview of Pricing Concepts for Broadband IP Networks,"* IEEE Communications Surveys & tutorials, 2nd Quarter, pp.9-13, 2000.

[4] L.A.DaSilva, *"Pricing for QoS-enabled Networks: A Survey,"* IEEE Communications Surveys & Tutorials, 2nd Quarter, pp.2-8, 2000.

[5] B. Teitelbaum et al. *"Internet2 QBone: building a testbed for differentiated services,"* IEEE Network, vol.13, no.5, pp.8-17, September 1999.

[6] A. M. Odlyzko, *"Paris Metro Pricing for the Internet,"* Proceedings of the ACM Conference on Electronic Commerce, pp.140-147, 1999.

[7] J. MacKie-Mason, L. Murphy, and J. Murphy, *"Responsive Pricing in the Internet,"* Internet Economics, L. W. McKnight and J.P. Bailey, Eds., Cambridge, Massachusetts, MIT Press, pp.279-303, 1997.

[8] J. K. MacKie-Mason, and H. R. Varian, *"Pricing Congestible Network Resources,"* Selected Areas in Communications, IEEE Journal on, vol.13, no.7, pp.1141-1149, 1995.

[9] N. Semret, R.R.-F. Liao, A.T. Campbell, and A.A. Lazar, *"Pricing, provisioning and peering: dynamic markets for differentiated Internet services and implications for network interconnections,"* Selected Areas in Communications, IEEE Journal on, vol.18, no.12, pp.2499 -2513, Dec. 2000 .

[10] S. Shenker, D. Clark, D. Estrin, and S. Herzog, *"Pricing in Computer Networks: Reshaping the Research Agenda,"* ACM Computer Communication Review, vol. 26, no. 2, pp. 19-43, April 1996.

[11] V. Fineberg, *"A Practical Architecture for Implementing End-to-End QoS in an IP Network,"* IEEE Communications Magazine, vol. 40, no.1, pp.122 -130, Jan. 2002.

[12] X. Wang and H. Schulzrinne, *"Pricing Network Resources for Adaptive Applications in a Differentiated Services Network,"* In Proceeding of INFOCOM 2001, Anchorage, Alaska, April 2001.

[13] Bin Pang, Huairong Shao, Wenwu Zhu, and Wen Gao, *"An admission control scheme to provide end-to-end statistical QoS provision in IP networks,"* Performance, Computing, and Communications Conference, 21st IEEE International, pp.399 -403, 2002.

[14] A. Ganesh and K. Laevens, *"Congestion Pricing and User Adaptation,"* in Proceedings IEEE INFOCOM, Anchorage, USA, April 2001.

[15] F. P. Kelly, P. B. Key, and S. Zachary, *"Distributed admission control, "* IEEE Journal on Selected Areas in Communications, Special Issue on Internet QoS, pp.2617Ũ2628, Dec. 2000.

[16] L. Breslau, E. W. Knightly, S. Schenker, I. Stoica, and H. Zhang, *"Endpoint Admission Control: Architectural Issues and Performance,"* ACM SIGCOMM 2000, Stockholm, Sweden, August 2000.

SHORT PAPER SESSION 1

Monitoring and Security

Co-Chairs: Kurt Geihs
Technical University Berlin, Germany

Mark Burgess
Oslo University College, Norway

MONITORING DISTRIBUTED SYSTEMS
A Publish/Subscribe Methodology and Architecture

Karen Witting, James Challenger, Brian O'Connell
IBM T. J. Watson Research Center and IBM Global Services Special Events

Abstract: To support complex, rapidly changing, high-volume websites many components contribute to keeping the content current. Monitoring the workflow through all these components is a challenging task. This paper describes a system in which monitoring objects created by the various heterogeneous, distributed components are distributed to any application choosing to present monitoring information.

Key words: Publish-Subscribe, Monitoring, Distributed Systems, Workflow Monitoring, Queue Monitoring, High Volume Web Serving, Content Management

1. INTRODUCTION

Systems comprised of a large number of interacting components require a highly flexible monitoring system. Modern, high volume web sites and their supporting infrastructure are an example of this kind of large system. "24x7" availability requires extremely flexible monitoring to cope with ever-changing hardware and software components. New types of components may be needed, and previously active components may be removed from the system. Any particular component may provide different types of monitoring data over time.

In this paper we describe the system designed and implemented to monitor flows within the publishing and content distribution systems for the Sydney 2000 Olympic Website [1] [3] and the IBM sponsored Special Events websites [2].

2. SYSTEM DESCRIPTION AND ARCHITECTURE

The serving infrastructure is comprised of several geographically distributed complexes. Content for the serving complexes flows from its originator, through one or more stages, to its final destination. The number and configuration of the stages varies by event. An application specific probe gathers monitoring data from the components at each stage. This data is published to the distribution system which delivers it to subscribers. Consumers subscribe to selected monitoring data and present it in various views for display. *Figure 1* shows an abstract view of the flow, where M1 is delivering content to M2 and M3.

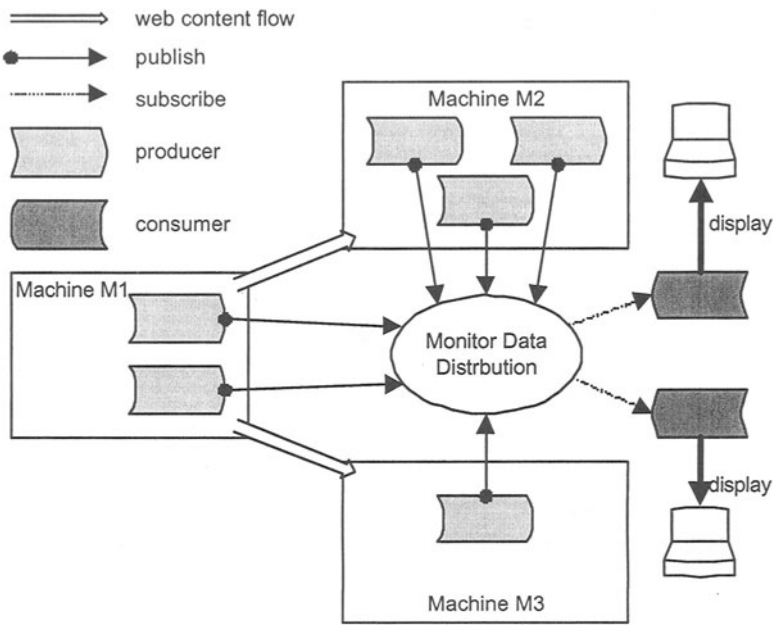


Figure 1: Monitoring System Architecture

The monitoring system consists of three main elements: *producers, consumers,* and a *distribution* mechanism. Producers gather and send out monitoring data, consumers receive data. The distribution mechanism coordinates the delivery of the data. The monitoring data itself is encapsulated into an opaque, self-describing *monitor object* which is designed to be independent of both the distribution mechanism and the consumer.

Monitor objects have properties that allow selection criteria to be applied by consumers. Three main properties are associated with every object: *event name*, ("www.wimbledon.org") *host name* ("server1.ibm.com") and *component name* ("SaveFile") to create a selection space for use by consuming applications. Beyond these base properties, a component may add any relevant data to the object. Data is accessed by interrogating the self-describing object allowing it. to change independently of both distribution system and consumers.

Producers create and then *publish* monitor objects to the distribution system. Each producer extracts monitor data that is specific to the monitored component. All producers use common facilities for creating and publishing monitoring objects.

Consumers receive data via subscription. After connecting to the distribution system, consumers specify selection criteria to control which objects they will receive. For example, a consumer may choose to receive data only associated with a particular event, data from a particular host, data from a specific component, or any combination of the above.

From the perspective of the *distribution system,* monitoring data is opaque. Producers and consumers interact only with the distribution system and thus are decoupled from each other. Because consumers are aware only of the self-describing *monitor objects* (and thus not explicitely aware of producers), producers can be added to or removed from the system and can change the type of object and data they are producing.

The systems we monitor are composed of a series of cascading hierarchically organized task/queue structures. Work flows through the system as tasks on queues. Every queue collects data about things like the number of tasks waiting and executing. Each queue is a producer and publishes a monitoring object containing the data collected about the queue.

Queues form a workflow hierarchy, where the output of tasks on one queue results in the addition of tasks onto queues below it in the hierarchy. Since each queue is a producer, monitoring data is generated from each node in the hierarchy.

3. EXPERIENCES

The original implementation and experiences with the system occurred while hosting the Sydney 2000 Olympic Website [1]. The general design and flow of the system was re-used for monitoring the Events Infrastructure [2]. These two experiences are similar in that they both are primarily involved in distributing work via queues and consist primarily of ensuring that work travels through the system without significant delay. Our methodology for monitoring the systems is a product of our experiences running these sites.

The Sydney 2000 Olympic Website [1] was hosted on a network of IBM RS/6000 SP2 complexes interconnected by a high-speed dedicated private network. Producers ran on AIX machines; consumers ran on a variety of platforms. The events infrastructure is currently hosted on a network of Netfinity X86 machines connected by a virtual private network. All producers of monitoring data run Linux while consumers of monitoring data run on a variety of platforms.

A key function of the monitoring systems is to provide data for management reports. Predicting the level of detail needed for these reports in advance is impossible. A novel hierarchical view of queues and tasks showing workflow enabled rapid identification of potential bottlenecks and provided a high level of flexibility in identifying and reporting problems. Detailed information about queues in the system was displayed in tabular views. When high queue counts are a concern the tabular views enable rapid diagnosis and correction.

Queues with work flowing through them were classified as *active, slow,* or *busy.* An *active* queue is receiving significant workload but is not overloaded. Active queues show large numbers of tasks flowing through them but relatively low numbers for queued counts. That is, an *active* queue has high throughput but low queue lengths, which indicates that tasks in that queue have very little wait time.

A *slow* queue is not working at its expected capacity. Slowdowns generally indicate an undesirable system condition such as a networking problem. Throughput is lower than expected, generally resulting in excessive queue wait time. A *slow* queue has low throughput and may or may not have long queue lengths.

A *busy* queue is receiving more work than it has workload capacity for. A busy queue could be indicative of component failure or simply indicate a spike in workload. A *busy* queue has both high throughput and long queue lengths. It is usually acceptable for a queue to be *busy* for some period of time (for example, as a result of a load spike) but extended *busy* conditions could indicate subtle system failures.

The ability for all queues to keep up with the workload demand is a significant focus of the entire support team. When a queue is falling behind and unable to process work in a timely manner, a great deal of focus and detailed understanding of the situation is required. Most of the time these situations are caused by a sudden, temporary, increase of work being added to the system, or a networking problem.

4. FUTURE ENHANCEMENTS

Enhancements include more sophisticated tools for log playback and database driven post-event analysis tools. Failure detection can be difficult; more specialized monitors to detect and rapidly report highly critical failures, increasing the granularity of reporting would be useful. Integration with SNMP and other standard protocols would allow other monitor clients to benefit from our queue based collection and reporting scheme.

5. ACKNOWLEDGMENTS

Several people have contributed to this work including Sandy Cash, Paul Dantzig, Cameron Ferstat, Ed Geraghty, Arun Iyengar, Herbie Pearthree, and Paul Reed.

6. REFERENCES

1. Sydney 2000 Olympic Website www.olympics.com from September 15 through October 1, 2000.
2. Selected IBM Sponsored Web sites: www.ausopen.org, www.masters.org, www.rolandgaros.org, www.wimbledon.org
3. Challenger, Jim, et al., A Publishing System for Efficiently Creating Dynamic Web Content. In *Proceedings of IEEE INFOCOM 2000*, March 2000.

PROACTIVE INTRUSION DETECTION AND SNMP-BASED SECURITY MANAGEMENT: NEW EXPERIMENTS AND VALIDATION

J.B.D. Cabrera[1], L. Lewis[2], X. Qin[3], C. Gutiérrez[1], W. Lee[3] and R.K. Mehra[1]
Scientific Systems Company[1]: University of New Hampshire[2]: Georgia Institute of Technology[3]

Abstract: In our earlier work we have proposed and developed a methodology for the early detection of Distributed Denial of Service (DDoS) attacks. In this paper, we examine the applicability of Proactive Intrusion Detection on a considerably more complex set-up, with hosts associated with three clusters, connected by routers. Background TCP, UDP and ICMP traffic following Interrupted Poisson Processes are superimposed on the attack traffic. We have examined six types of DDoS attacks. In four of the attacks we have obtained valid MIB-based precursors with no false alarms in all experiments. In the remaining two attacks precursors were obtained, but false alarms were observed. Procedures for eliminating these false alarms are discussed.

Keywords: Security Management; Data Warehousing and Statistical Methods in Management; Network and Systems Monitoring; Information Modeling

1. INTRODUCTION

In [2] we developed a methodology for utilizing traffic MIB variables for the early detection of Distributed Denial of Service (DDoS) attacks: Proactive Intrusion Detection, as opposed to the passive detection enabled by current IDSs.

In the present paper we evaluate the Proactive Intrusion methodology in a more realistic scenario. We performed the experiments on a network formed by three sub-networks connected by routers. The DDoS master, the DDoS slave and the target are placed in different clusters, allowing us to investigate the effect of routing in our ability to extract the precursors. We produced traffic for TCP, ICMP and UDP following Interrupted Poisson Processes and superimposed it on the attack traffic. Finally, we experimented with six types of DDoS attacks. We refer the reader to [2][3] for a detailed discussion of the Proactive Intrusion Detection Methodology.

2. THE NEW TEST BED AND EXPERIMENTS

Topology: R1 - R3 are routers; H1 through H6 are normal hosts.

Data collection: 64 MIB variables from the `ip`, `udp`, `tcp` and `icmp` groups were collected for 2 hours, at a sample rate of 3 seconds.

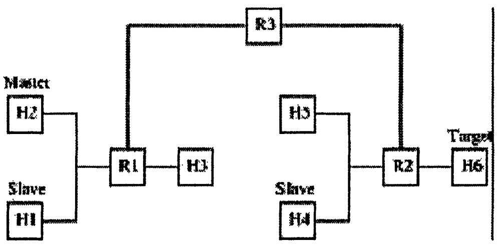


Figure 1 The NCSU Test Bed

Attack Runs and Normal Runs: 12 attack runs corresponding to 6 types of attacks were obtained. The attacks are: TFN2K SYN Flood, TFN2K Ping Flood, TFN2K Targa3 Flood, TFN2K Mix Flood, TFN2K UDP Flood and Trin00 [6]. Two runs are available for each attack type. 12 normal runs are also available, in which no attacks are present.

Placement of Malicious Agents: The master agent is placed in H2 during all attacks. Slave agents are placed in H1 and H4. The target is H6.

Background and Malicious Traffic: During normal runs, H1 to H6 generate TCP, ICMP and UDP traffic according to Interrupted Poisson Processes [5]. During attack runs, attack traffic was produced by malicious agents at H1, H2 and H4 and superimposed on the normal traffic. Live TCP connections among the various nodes of the network allowed us to observe the phenomenon of connection "time-out" caused by congestion.

3. EXPERIMENTAL RESULTS

3.1 Step 1: Determining key variables at the target

Key variables were selected by comparing the time series of normal and attack runs. Since the attacks can be classified as flood attacks, MIB variables related to packet transmission are the best candidates for attack detection. `ipInReceives`, belongs to this category and is the most promising as it appears in all attacks.

3.2 Step 2: Determining key variables at the attackers

We search for possible causal relationships connecting MIB variables that could be considered key variables for each attack at the attacker hosts (H1 through H5) with the MIB variable selected in step 1 (`ipInReceives`) at the target host (H6). The value for the causality index between these variables was obtained using the

Granger Causality Test (GCT) [2]. The MIB variables found are then compared to the ground truth variables, selected from domain knowledge about the attacks [6].

Two types of precursors are defined. The first, called *T2 precursors*, correspond to the communication between master and slave. Second are *T3 precursors*, which correspond to the type of flood depending on the DoS attack.

3.3 Steps 3 and 4: Determining key events at the attacker

We break finding the key events at the potential attacker hosts that can be considered as precursors of an attack into two steps: step 3, *training*, where we determine the precursors at the potential attacker host and step 4, *testing*, where we test these precursors. In step 3, we define a precursor as an abnormal change on a precursor variable. To find this abnormal change, a Normal Profile for each MIB variable is constructed from the normal runs. Any variable whose change in each attack run was larger than the one in the Normal Profile was considered a precursor event for step 3 for that run.

Table 1. TFN2K Ping Flood Precursor Events - Run 1

T = 250		*Test Attack Data*										*Normal Data*				
Training Attack	**Precursor Variable**	**Detections**					**False Alarms**					**False Alarms**				
		H1	H2	H3	H4	H5	H1	H2	H3	H4	H5	H1	H2	H3	H4	H5
PING_1_1	ip.ipOutRequests	x														
through	tcp.tcpRetransSegs								x							
PING_1_6	tcp.tcpInErrs	x			x											
	udp.udpInErrors	x			x											

In step 4, the precursor events of one attack run were tested on the other run. Table 1 shows the results for the first run of TFN2K Ping Flood. *Precursor variables* are the MIB variables that were selected following step 2 and step 3. Under *Test Attack Data*, *Detections* corresponds to the variables that belong to ground truth, and *False Alarms* are variables detected that do not belong to ground truth. Under *Normal Data*, the variables that are marked are those whose jump during a the normal test run was greater than that in the Normal Profile, in other words, a false positive.

Table 1 shows that there are T2 and T3 precursor events in H1 and H4, and `tcpRetransSegs` appears as a false alarm in H3. H1 and H4 can be detected as the attack hosts during this TFN2K Ping Flood.

4. SUMMARY AND CONCLUSIONS

The overall results demonstrated the applicability of Proactive Intrusion Detection using more realistic background traffic and sub-networking. In three of the attacks (TFN2K Syn Flood, TFN2K Mix Flood and TFN2K Targa3) both T2 and

T3 precursors were obtained, with no false alarms. For TFN2K UDP Flood, no false alarms were recorded, but T3 events were not obtained. For TFN2K Ping Flood and Trin00 UDP Flood false alarms were observed, related to the variable `tcpRetransSegs`. During an attack the network becomes congested and the open TCP connections start to time-out and TCP packets are retransmitted. `tcpRetransSegs` is detected as a precursor at H3, which is. neither an attacker nor a target. Since the TCP time-out effect and retransmission occur after the attack starts a second pass may eliminate these false alarms. We are currently exploring an application of precursors which is more directly related to "conventional" (not proactive) Intrusion Detection. It has been noted [1][4] that the high rate of false alarms in current IDSs represents perhaps the main challenge for the deployment of IDSs. Precursors could be useful to corroborate the "hits" from a conventional IDS.

Finally, using more general traffic models instead of Interrupted Poisson Processes for background traffic would not have changed the results for this particular set of experiments and topology. We noted that key precursors MIBs are not related with traffic counting processes, but with counting anomalies in the protocol stack.

ACKNOWLEDGEMENTS

This work was supported by the Air Force Research Laboratory (Rome, NY) under contract F30602-01-C-057 to Scientific Systems Company and by Aprisma's University Fellowship Program 1999/2000. SSCI acknowledges the support from the Defense Information Warfare Branch at the Air Force Research Laboratory.

REFERENCES

1. S. Axelsson. *The base-rate fallacy and the difficulty in intrusion detection*, ACM Transactions on Information and Systems Security, vol. 3, no. 3, 2000.
2. J.B.D. Cabrera, L. Lewis, X. Qin, W. Lee and R.K. Mehra. *Proactive Intrusion Detection and Distributed Denial of Service Attacks – A Case Study in Security Management*, Journal of Network and Systems Management, vol. 10, num. 2, pp. 225-254, June 2002.
3. J.B.D. Cabrera and R.K. Mehra. *Extracting Precursor Rules from Time Series – A Classical Statistical Viewpoint*, Proceedings of the Second SIAM International Conference on Data Mining, Arlington, VA, pages 213-228, April 2002.
4. S. Northcutt. *Intrusion Detection: An Analyst's Handbook*, New Riders, 1999.
5. H.G. Perros. *An Introduction to ATM Networks*, John Wiley and Sons, 2001.
6. P.J. Criscuolo. *Distributed Denial of Service – Trin00, Tribe Flood Network, Tribe Flood Network 2000, and Stacheldraht*. Technical Report CIAC-2319, Department of Energy (CIAC – Computer Incident Advisory Capability), February 2000.

NETLOGGER
A Toolkit for Distributed System Performance Tuning and Debugging

Dan Gunter (dkgunter@lbl.gov), Brian Tierney (bltierney@lbl.gov)
Lawrence Berkeley National Laboratory, 1 Cyclotron Road, Berkeley, CA 94720

Abstract: Developers and users of high-performance distributed systems often observe performance problems such as unexpectedly low throughput or high latency. Determining the source of the performance problems requires detailed end-to-end instrumentation of all components, including the applications, operating systems, hosts, and networks. In this paper we describe a methodology that enables the real-time diagnosis of performance problems in complex high-performance distributed systems. The methodology includes tools for generating timestamped event logs that can be used to provide detailed end-to-end application and system level monitoring; and tools for visualizing the log data and real-time state of the distributed system. This methodology, called NetLogger, has proven invaluable for diagnosing problems in networks and in distributed systems code. This approach is novel in that it combines network, host, and application-level monitoring, providing a complete view of the entire system. NetLogger is designed to be extremely lightweight, and includes a mechanism for reliably collecting monitoring events from multiple distributed locations.

Key words: distributed systems performance analysis and debugging

1. INTRODUCTION

The performance characteristics of distributed applications are complex, rife with "soft failures" in which the application produces correct results but has much lower throughput or higher latency than expected. Bottlenecks can occur in any component along the data's path: applications, operating systems, device drivers, network adapters, and network components such as switches and routers. We have developed a methodology, known as NetLogger (short for *Net*worked Application

Logger), for monitoring, under realistic operating conditions, the behavior of all the elements of the application-to-application communication path in order to determine exactly what is happening within a complex system.

Distributed application components, as well as some operating system components, are modified to perform precision timestamping and logging of "interesting" events, at every critical point in the distributed system. The events are time-correlated with the system's behavior in order to characterize the performance of all aspects of the system and network in detail during actual operation.

NetLogger has demonstrated its usefulness in a variety of contexts, but most frequently in loosely coupled client-server architectures. We began developing NetLogger in 1994, and in previous work we have shown that detailed application monitoring is vital for both performance analysis and application debugging [6]. This paper gives a very brief summary of the main NetLogger components and provides a case study. A longer version of this paper that includes extended descriptions, details on recent NetLogger enhancements, and a more complete set of references, is [6].

There are a number of systems that address application monitoring. *log4j*, part of the Apache Project [4], has produced a flexible library for Java application logging. However, the performance of *log4j* is far lower than is necessary for detailed monitoring.

Another instrumentation package is the Open Group's Enterprise Management Forum's [5] Application Response Measurement (ARM) API, which defines function calls that can be used to instrument an application for transaction monitoring. Tools to visualize and discover patterns of ARM events are described in [3].

2. NETLOGGER TOOLKIT COMPONENTS

The NetLogger Toolkit consists of four components: an API and library of functions to simplify the generation of application-level event logs, a set of tools for collecting and sorting log files, a set of host and network monitoring tools, and a tool for visualization and analysis of the log files. Instrumentation is performed by modifying source code and linking with the NetLogger library. All the tools in the NetLogger Toolkit share a common log format, and assume the existence of accurate and synchronized system clocks. We have found that the NTP tools that ship with most Unix systems (e.g.: *ntpd*) can often provide the desired level of synchronization.

Figure 1 shows sample results from the NetLogger Visualization tool, *nlv*, using a remote data copy application. The events being monitored are shown on the y-axis, and time is on the x-axis. From bottom to top, one can see CPU utilization events, application events, and TCP retransmit events all on the same graph. Each semi-vertical line represents the "life" of one block of data as it moves through the application. The gap in the middle of the graph, where only one set of header and data blocks are transferred in three seconds, correlates exactly with a set of TCP retransmit events. Thus, this plot makes it easy to see that the "pause" in the transfer is due to TCP retransmission errors on the network.

We have found exploratory, visual analysis of the log event data (as opposed to rule-based correlation such as that presented in [3]) to be the most useful means of determining the causes of performance anomalies. The NetLogger Visualization tool, *nlv*, has been developed to provide a flexible and interactive graphical representation of system-level and application-level events. For more details, see [7].

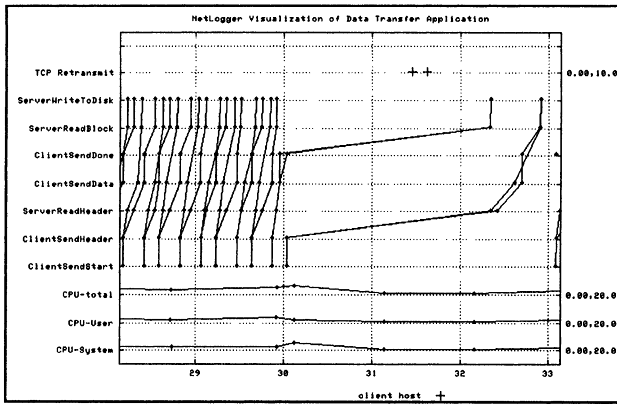


Figure 1. NetLogger Visualization Tool

NetLogger events can be formatted as an easy to read and parse ASCII format. To address the overhead issues discussed above, NetLogger includes a highly efficient self-describing binary wire format, capable of handling over 600,000 events per second. NetLogger also includes a remote activation mechanism, and reliability support.

3. CASE STUDIES

Note: due to space limitations, the figures illustrating these two case studies are online at http://www-didc.lbl.gov/NetLogger/examples/ *under* radiance_pic.png *and* globus-logs/gridftp_select_bug.png *for the first and second case study, respectively.*

In the first case study, NetLogger was used to instrument a 3-dimensional visualization engine called Radiance [2] that read data off disk, rendered it, and sent it out to clients for display. The *lifelines* in these graphs represent the data flow to generate one image. The upper graph shows the results before NetLogger tuning. The developer in this case had assumed that the I/O time was greater than the image rendering time, and therefore didn't bother to make the program multi-threaded and overlap processing with I/O. After seeing these results, however, the developer made the program multi-threaded. The new code produced the results in the lower graph; almost double the performance.

In the second case study a high-performance FTP client/server called GridFTP [1] was instrumented. Among other enhancements, GridFTP extends the FTP protocol to transfer a single file across several parallel TCP streams. In some WAN environments this can cause a dramatic (almost linear) speedup.

The bottom three groups of lifelines show headers and packets arriving on three sockets for a parallel FTP client. Data should be steadily arriving on all three sockets, but clearly the client was not servicing all three sockets equally. Further analysis showed that there was a months-old bug in the way the Unix *select*() call was being used. Despite the bug, the multi-stream version of the FTP client was faster than the single stream version, so no one had noticed this problem. This is the type of subtle bug that NetLogger is very good at tracking down.

These two case studies demonstrate the NetLogger's ability to analyze a single application. In both cases *nlv* made it easy to spot problems. However, NetLogger's real power is demonstrated by analyzing a distributed application, and time-correlating monitoring from the application, host, and network.

4. CONCLUSIONS

In order to achieve high end-to-end performance in widely distributed applications, a great deal of analysis and tuning is needed. The top-to-bottom, end-to-end approach of NetLogger has proven to be a very useful mechanism for analyzing the performance of distributed applications in high-speed wide-area networks. All NetLogger Toolkit components under an Open Source license, and can be downloaded from http://www-didc.lbl.gov/NetLogger/.

ACKNOWLEDGMENTS

This work was supported by the Director, Office of Science. Office of Advanced Scientific Computing Research. Mathematical, Information, and Computational Sciences Division under U.S. Department of Energy Contract No. DE-AC03-76SF00098. This is report no. LBNL-51276.

REFERENCES

[1] Allcock B., Bester, J., Bresnahan, J., Chervenak, A., Foster, I., et.al. *Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing*. IEEE Mass Storage Conference, 2001.

[2] Bethel, W., B. Tierney, J. Lee, D. Gunter, S. Lau. *Using High-Speed WANs and Network Data Caches to Enable Remote and Distributed Visualization*. Proceeding of the IEEE Supercomputing 2000 Conference, Nov. 2000.

[3] Burns, L., JL Hellerstein, S Ma, CS Perng, DA Rabenhorst, D Taylor, *A Systematic Approach to Discovering Correlation Rules for Event Management*, IFIP/IEEE International Symposium on Integrated Network Management, 2001.

[4] log4j: http://jakarta.apache.org/log4j/docs/index.html

[5] Open Group, Enterprise Management Forum. 2002, http://www.opengroup.org/management/arm.htm.

[6] Tierney, B., W. Johnston, B. Crowley, G. Hoo, C. Brooks, D. Gunter. *The NetLogger Methodology for High Performance Distributed Systems Performance Analysis*. Proceeding of IEEE High Performance Distributed Computing, July 1998, LBNL-42611. http://www-didc.lbl.gov/NetLogger/

[7] Tierney, B. and D. Gunter, NetLogger: A Toolkit for Distributed System Performance Tuning and Debugging , LBNL Tech Report LBNL-51276. http://www-didc.lbl.gov/papers/NetLogger.overview.pdf

A CASE STUDY OF THREE OPEN SOURCE SECURITY MANAGEMENT TOOLS

Hilmi Gunes Kayacik, A. Nur Zincir-Heywood
kayacik@cs.dal.ca, zincir@cs.dal.ca
Dalhousie University, Faculty of Computer Science, Canada

Abstract: Three open source security management tools – *Snort, Pakemon, and Argus* – are benchmarked against DARPA 1999 Intrusion Detection Evaluation Data Set. Performance is characterized using multiple performance metrics. *Snort* is found to have the best performance in terms of detection rate, however it creates more false positives than desired. The results show that different tools perform well under different attack categories; hence they can be run at the same time to increase the detection rate of attack instances.

Key words: Security management, Case Study, Open Source Software, IDS

1. INTRODUCTION

Security management plays an important role in today's network management tasks. Defensive information operations, and intrusion detection systems are primarily designed to protect the availability, confidentiality and integrity of critical network information systems [3]. The automated detection and immediate reporting of these events are required in order to provide a timely response to attacks [2]. A balance therefore exists between the use of resources and the accuracy and timeliness of intrusion detection information. Since most of the commercial intrusion detection systems are at typically thousands of dollars and they tend to represent a significant resource requirement in themselves, for small networks, use of such IDS is not feasible. The objective of this work is therefore to evaluate three open source security management tools in order to understand which one of them will be more useful for network intrusion detection. To achieve this, we have chosen *Snort*, *Pakemon* and *Argus*, since they are three of the most popular open source tools [1, 5, 6]. "Pakemon has been developed to share IDS components based on the open source model" [6]. It is an experimental IDS, which aims to detect evasion methods such as fragmentation, disorder, duplication, overlap, insertion, and de-synchronization at the IP or TCP layer. Pakemons's signature structure is simpler than other IDS (such as *Snort*), where this simplicity is both a strength, and a weakness. That is to say, it takes time for IDS organizations to release new signature files. Meanwhile, as the signatures of new attacks are revealed, it is much easier to add them to the lightweight IDS signature databases such as Pakemon [6]. Snort is one of the best-known lightweight IDSs, which focuses on performance, flexibility and simplicity [5]. It is an example of active intrusion detection systems that detects possible intrusions or access violations while they are occurring. Although not as straightforward as the Pakemon system, flexible rule writing is supported in Snort. In contrast to Pakemon and Snort, *Argus* is not an IDS but it is an open source general network management tool [1]. This means that Argus monitors and inspects network traffic and connections both for attempted connections and established connections. In other words, it is a specific IP auditing tool, hence in the case of this work, it is used to analyze the superset of the traffic logged by the other two intrusion detection systems.

2. TEST SET UP AND PROCEDURES

The test setup of this work consists of the following components: DARPA 1999 data set, traffic re-player, and the three open source systems. The data set [4] represents TCP dump data generated over five weeks of simulated network traffic in a hypothetical military local area network (LAN). This data was processed into some 7 million TCP connection records. Our work concentrates on the internal and external traffic collected by the sniffers. In this case for, reasons of expediency, we concentrate on the 2.5 GB of data present in the week 4 data set (week 5 is even larger and beyond the computing resources available). The data used for testing therefore either represented a normal connection or one of the 80 attacks [4]. This work employed one machine as the IDS server and another machine to replay the network traffic using *TCPReplay* [7]. Moreover, all the software used are installed and configured using their default values, and the latest signature files available (February 2002) are used for *Pakemon* and *Snort.* It should be noted that log files of the tools that are evaluated contain different types of entries including different amounts of information about the events that occurred on the network. Therefore, 4 confidence levels are defined for determining the degree of match in order to detect different attacks, table-1.

3. RESULTS

Out of total number of 80 attack instances, *Snort* detected 35 and *Pakemon* detected 27 in total. Indeed, it should be noted that even if we had an intelligent way to mine *Argus* log file, we could have only detected 70 attacks out of the 80 present in the test data set. To actually determine which tool performs better, two other parameters are analyzed: (1) the number of false alarms and (2) the number of entries that it takes to be parsed by a network administrator to detect those attacks.

Figure 1 shows the number of attack related entries over the total number of entries in the corresponding log files. Thus, in both cases it is costly to examine all log files. On the other hand, when the attacks detected by *Snort* and *Pakemon* are examined more closely, a strong commonality exists between the types of attacks detected. As it can be seen in figure 2, Snort on its own is much better than *Pakemon,* however if they work together Their performance increases by approximately 20%. For both of them though, the confidence level of detection is mostly at level-3, figure 3.

	Source IP	Destination IP	Source Port	Destination Port
Levell	X	X	X	X
Level2	X	X		X
Level3	X	X		
Level4	X			

Table 1: Summary of the confidence levels (X indicates the match required)

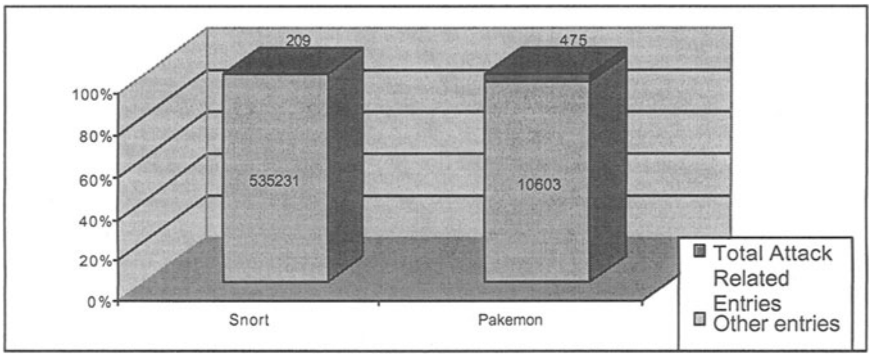


Figure 1. Number of attack related entries in the corresponding log files

4. CONCLUSION

The work presented is a case study, but we believe sufficient to warrant continued development. In particular, we have demonstrated a benchmark evaluation of popular open source security management tools. The results show that none of the tools could capture all the different attack instances: Snort captured ~44% and Pakemon ~34%. Moreover, Snort has ~99% false alarms whereas Pakemon has ~95%. In other words all three generate very large log files, which in return makes it difficult to analyze for network managers. Therefore, it is important to develop filters for these tools to decrease the number of false alarms. Furthermore, we believe that different tools need to be used together to increase the detection rate.

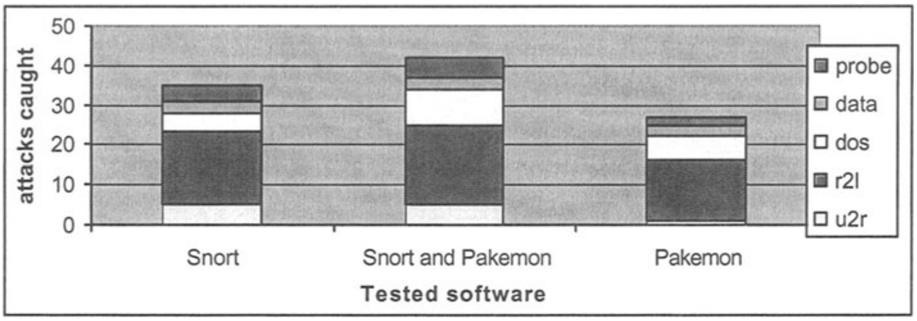


Figure 2: The distribution of attacks that are caught by *Snort* and *Pakemon*

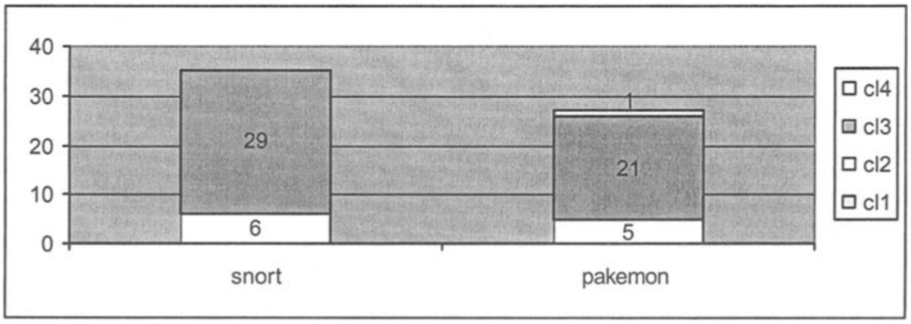


Figure 3: Number of attacks and their corresponding confidence levels for each tool

ACKNOWLEDEGEMENTS

The authors gratefully acknowledge the financial support of the Natural Sciences and Engineering Research Council of Canada.

REFERENCES

[1] Argus, http://www.qosient.com/argus

[2] Bass T., "Intrusion Detection Systems and Multisensor Data Fusion", Communications of the ACM, Vol. 43, No. 4, pp 99-105, April, 2000.

[3] Kayacik G., Zincir-Heywood A. N., "Evaluation of the Cisco IOS Firewall with Darpa 99 Dataset", Technical Report, Faculty of Computer Science, Dalhousie University, http://www.cs.dal.ca/~kayacik/download/report.pdf, November 2002

[4] MIT Lincoln Laboratory, http://www.ll.mit.edu/IST/ideval/data/data_index.html

[5] Snort, http://www.snort.org

[6] Takeda K., Takefuji Y., "Pakemon – A Rule Based Network Intrusion Detection System", International Journal of Knowledge-Based Intelligent Engineering Systems, Vol. 5, No. 4, pp 240-246, October 2001.

[7] TCPReplay traffic replay utility, http://tcpreplay.sourceforge.net/

MTREEDX: A MULTICAST NETWORK DIAGNOSIS TOOL

Jaiwant Mulik
Computer and Information Sciences Department
Temple University, Philadelphia
jmulik@temple.edu

Phillip Conrad
Computer and Information Sciences Department
Temple University, Philadelphia
conrad@acm.org

Brian Drake
Information Services and Advanced Technology Division
Goddard Space Flight Center, NASA
brian.drake@gsfc.nasa.gov

Saroj Biswas
Electrical and Computer Engineering Department
Temple University
saroj.biswas@temple.edu

Musoke Sendula
Electrical and Computer Engineering Department
Temple University
hsendaul@nimbus.temple.edu

1. Introduction

MTreeDx is a tool that is being developed to fulfill a gap in most currently available Multicast Network Management Tools. Specifically, this tool is being developed as an aid to diagnose problems in NASA's multicast network. Management of the NASA multicast network present several unique research challenges.

2. Motivation

The images and other data acquired from space telescopes is distributed to research institutions using a multicast network. The network manager would like to prevent unauthorized hosts from joining the group. Detecting senders is not difficult; the tool can simply listen for transmissions from unknown sources on the group. Detection of unauthorized receivers can be tricky. Unauthorized receivers on a subnet with no authorized receiver can be detected by the presence of an "extra branch" in the multicast distribution tree. We are investigating techniques to use mtrace [3] or similar tools to generate and display the current multicast tree. Guaranteeing the detection of unauthorized receivers on a shared media subnet with authorized receivers in provably impossible. We are investigating router-based techniques to detect unauthorized hosts.

The second problem is that of monitoring the multicast traffic. Several commercial and open source tools are available that require specific feedback from the network. For example, a multicast monitoring extension to a popular commercial network management system requires SNMP capability within the network. Mhealth [2] requires rtcp reports to display loss statistics. In the absence of rtcp sources or widespread SNMP support we are currently evaluating techniques that involve placing "monitoring stations" at strategic locations within the network. These stations could use techniques similar to those developed for the MINC [1] project.

3. Features of MTreeDx

Currently, the main features of MTreeDx are:

1 Displays the tree topology of group members from the perspective of the monitoring station [Fig. 1]. The group membership information is obtained from a pre-configured list of hosts. Each host display includes its IP address and the last measured round-trip time from the monitoring host. We are currently investigating router-based techniques to obtain group membership.

2 Provides visual information about all senders, active, and inactive.

3 Provides visual alarm, and logs identity of unauthorized sender [Fig. 3]. A list of authorized senders is pre-configured. Any sender not found in the authorized list is deemed unauthorized.

4 Provides visual information about authorized receivers.

5 Provides visual information about currently unreachable hosts [Fig. 2]. Unreachable hosts are moved from the tree display and displayed in a separate list.

6 Provides color-coded indications of the status of a node.

7 Multiple sessions of MTreeDx can be run in parallel.

Another challenge of creating a tool like MTreeDx comes from its requirement as an integrated tool for both security and traffic monitoring. We want to be able to create functionality that is common to both security and traffic

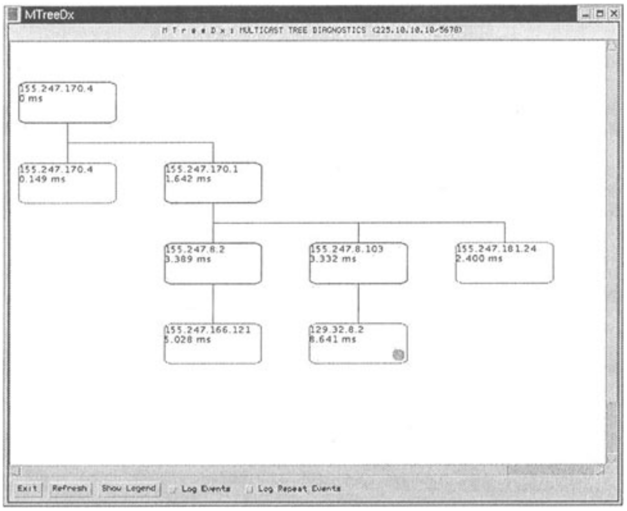


Figure 1. MTreeDx with one active sender

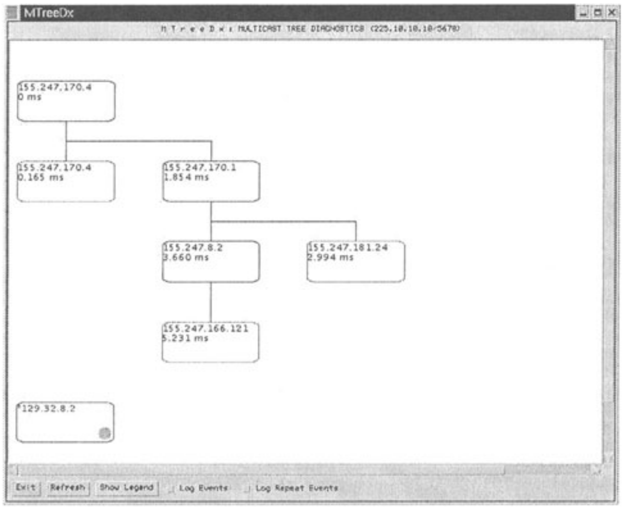


Figure 2. MTreeDx with an unreachable host

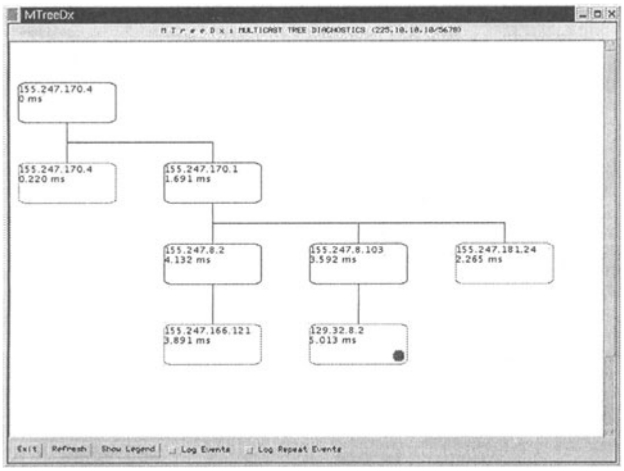


Figure 3. MTreeDx with one intruder

monitoring. For example, we are developing an intelligent display and logging system. Visual alarms are an important characteristic of both security and traffic monitoring. Given potentially large networks, we are developing intelligent interfaces that will allow the network administrator to prioritize, based customizable rules, portions of the network currently in view.

References

[1] Multicast-based inference of network-internal characteristics. http://www-net.cs.umass.edu/minc/.

[2] Makofske D. B and Almeroth K. C. MHealth: A Real-time Multicast Tree Visualization and Monitoring Tool. Technical report, University of California, Santa Barbara. Makosfske's Master Thesis.

[3] Fenner W. and Casner S. A 'traceroute' facility for IP multicast. Technical report, IETF, August 1998. Work in progress.

MULTIPLE AUTHORIZATION

A Model and Architecture for Increased, Practical Security

Gerald Vogt
Munich Network Management Team — University of Technology Munich
Oettingenstr. 67, 80538 Munich, Germany
Gerald.Vogt@informatik.uni-muenchen.de

Abstract: Security of systems and management infrastructure is crucial for a successful, reliable and safe use. Most currently deployed systems are based on simple subject/object-relations where control of access happens. This has the drawback that any access decision occurs just on the behalf of the single subject accessing. In this paper we describe an extended access control model that allows to include authorization of multiple subjects thus overcoming this drawback while still focusing on practical aspects of simple integration in many of the existing systems.

Keywords: Security Management, Access Control, Authorization

1. INTRODUCTION

Access control is the part of a security system that actively maintains security checking authorizations and thereby controlling access to protected resources and functions. The main parts involved in access control are shown in figure 1. The active entity accessing the protected operation is called the *subject* for this access. The subject is usually linked with a real person who has been identified during authentication. The *object* of an access can be any kind of resource or function that is accessible and protected for security reasons. A subject performs an *operation* on an object by calling a method/sending a message/making a procedure call, depending on the underlying paradigm and passing all necessary parameters that define the details of the operation.

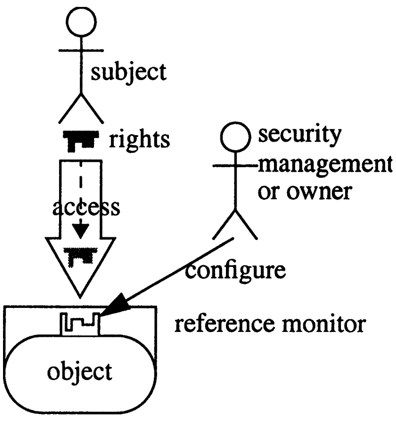


Figure 1. Simple access control

The *reference monitor* is the part of the security system that actively enforces access control. It checks the authorization of the accessing subject and decides whether the subject is allowed to execute the operation or not. The identity of the subject and/or other concepts like tickets or capabilities can be used to determine the rights and to prove the required authorization. It is the duty of security management to assign the necessary rights to subjects depending on the tasks they have to work on. In addition, many systems also allow an owner of a resource to assign rights, e.g., the owner of a file grants access to other subjects. The responsibility of the owner often coincides with security management.

The access control model described in this section very much follows the idea of discretionary access control (DAC) [1], one of the traditional access control models, that is the foundation of many current operating systems like UNIX and languages like Java. The discussion sections briefly covers some other access control models and how they relate to multiple authorization presented in the following section.

2. MULTIPLE AUTHORIZATION

One of the limiting factors of current access control systems is the dependence on a single subject for authorization. It is usually not possible to directly involve other entities, i.e., other people in a particular access control decision. This, however, would be reasonable if the access involves an object that is critical and severe effects could occur unless done properly. In the simplest case of two subjects this is the 'four-eyes' principles from real-world scenarios when something must be done in mutual agreement of two persons. *Multiple Authorization* as presented in this section is a practical mean to integrate this idea in existing systems. It is an extension of the simple, single-subject authorization model underlying most currently used systems. Instead of depending on a single subject, it allows to include several subjects where necessary (Figure 2). In the following, we use fractional rights to demonstrate the concept, e.g. 1/3 for a third of the full right. Yet, it is not limited to fractions and arbitrary constraints could be used here as well, though they certainly add some complexity.

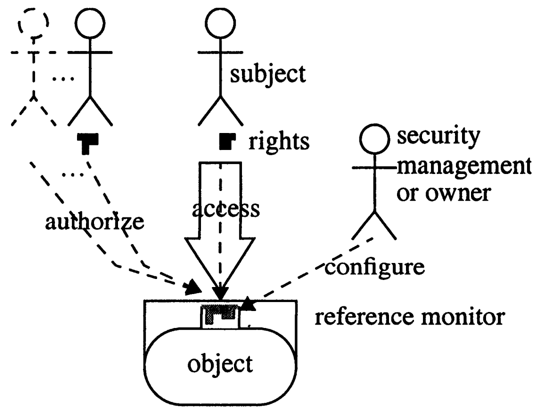


Figure 2. Access control with multiple authorization

First of all, when an *accessing subject* accesses an operation it expresses its will to execute it. If the subject does not have sufficient rights (e.g., only 1/3) to execute the operation alone, the authorization is in *incomplete* state. Other subjects in the new role of the *authorizing subject* can now supplement (e.g. with another 1/3) until authorization is *complete* (i.e. the sum is >= 1). Supplementing authorization of an operation is a new system function that does not exist in simple authorization. As the authorizing subject authorizes an operation access of a different subject, it needs information about the operation in question before it can decide what to do. This requires the ability to *inspect* the incompletely authorized operation with all its parameters as well as the progress of authorization until now and further context or state information available. The authorizing subject is completely free to choose its decision logic. Due to access details and further knowledge of the current tasks being executed, the decision happens on a more exact level than other models with a priori rights assignments can reach. At the end of the decision making process the authorizing subject must either authorize the operation (with all, e.g. 1/2, or only a part of its own rights, e.g., 1/3 instead of 1/2) or reject authorization. Rejection prevents the operation to be executed. A conditional authorization is used to include further subjects in the authorization process.

As only authorization and thus access control is directly involved for the extension of the existing model, it is not necessary to completely redefine the execution model from the perspective of the accessing subject. The main control flow of the accessing subject remains the same. Handling of multiple authorization happens through callbacks in a new separate extension.

This leads to a number of interfaces that an architecture integrating multiple authorization must provide or modify to meet the new needs. There are basically four interfaces provided by the security system:

- the security management interface to configure the access rights and define the conditions when an authorization is sufficient for an operation,
- the access control interface to check authorization of a particular operation and delaying its execution if it is subject to multiple authorization,
- the inspection interface to request details about the operation to be authorized for the authorizing subject (this interface itself must be secured from unauthorized access), and
- the authorization interface to finally grant or reject authorization for a particular operation.

Moreover, the participating subjects must provide two interfaces used for communication:

- the callback interface of the accessing subject called by the reference monitor if authorization is incomplete, and
- the authorization request interface of the authorizing subject called by the accessing subject to request authorization for the deferred operation call.

3. DISCUSSION & RELATED WORK

The presented architecture focuses on the practical integration on the base of existing security systems which often use rather simple DAC models. In this sense, it is not considered as a replacement of other more sophisticated security models which, however, also require a certain level of complexity in respect to the implementation of the reference monitor, the supporting infrastructure and the management of the whole system. In contrast, the presented concept with fractional rights is still simple to understand, requires less complex changes to the reference monitor leading to more robust implementations, and has a straight-forward integration of the existing DAC model. For example, for an integration in Java [2], it is necessary to extend the Policy class to support new fractional rights beside the old grant rules. A new SecurityManager must then be able to check for the fractional rights and do the callbacks if necessary as well as provide the interfaces for inspection and authorization as outlined above. A prototype for the Java integration is being developed to demonstrate the feasibility and possible applications.

The basic idea of involving several people in critical operations is not new. Various access control models have concepts to deal with separation of duty (e.g., RBAC [3], Policies [4]). With separation of duty two operations calls, e.g. on a particular object, must be performed by distinct subjects. This can then be used to increase security when, for example, someone who writes an order cannot approve an order and vice-versa. This, however, requires two different operations to exist for this purpose, i.e., a write method and an approve method on an order object. This assumption is not valid in many existing systems, e.g., in file systems that usually do not offer something like an approve read method on a file. (The presented architecture provides this split operations, thus could be used for this purpose.)

Other approaches using multiple credentials which are passed to the accessing subject are also possible. Here, though, the authorizing subjects loose direct control of their credentials once they have been given away. In addition, it requires them to fully identify all possible constraints they can put on the credential to limit the extent of the right they give away. In the end, this often means, that they have to foresee the exact purpose of the credential. With multiple authorization they authorize an actually happening operation call with 'real live' parameters and their final decision can even be based on some manual, in-person inspection, if in doubt.

Acknowledgement. The author would like to thank the members of the Munich Network Management (MNM) Team for helpful discussions and valuable comments on drafts of this paper. The MNM Team directed by Prof. Dr. Heinz-Gerd Hegering is a group of researchers at the University of Munich, the Munich University of Technology, and the Leibniz Supercomputing Center of the Bavarian Academy of Sciences.

Its webserver is located at http://wwwmnmteam.informatik.uni-muenchen.de/

REFERENCES

[1] M. Harrison, W. Ruzzo, J. Ullman. Protection in Operating Systems. In: *Communications of the ACM*, 19(8):461–471, August 1976.

[2] S. Oaks. *Java Security.* O'Reilly & Associates, 2nd edition, May 2001.

[3] R. Sandhu, E. Coyne, H. Feinstein, C. Youman. Role-Based Access Control Models. In: *IEEE Computer*, 29(2):38–47, February 1996.

[4] M. Sloman, E. Lupu. Security and Management Policy Specification. In: *IEEE Network*, 16(2):10–19, March/April 2002.

A CONTROLLER AGENT MODEL TO COUNTERACT DoS ATTACKS IN MULTIPLE DOMAINS

Udaya Kiran Tupakula Vijay Varadharajan
Information and Networked System Security Research
Division of Information and Communication Sciences
Macquarie University, Australia. *{udaya, vijay}@ics.mq.edu.au*

Abstract: In this paper we discuss techniques to prevent Distributed Denial of Service (DDoS) attacks within the ISP domain and extend the scheme to prevent the attack in multiple ISP domains. With a new packet marking technique and agent design, our model is able to identify the approximate source of attack with a single packet and has many features to minimise DDoS attacks.

Key words: Denial of Service, DoS, Packet Marking, Routing Arbiter.

1. OUR APPROACH

Our architecture involves a Controller-Agent model. In each ISP domain, we envisage that there exists a controller, which is a trusted entity (within the domain) and is involved in the management of denial of service attacks. We consider external attacks where attacks originate outside the ISP domain and target the victim, which is also outside the ISP domain. Routers are mainly classified into internal and external routers. Internal routers belong to the ISP and external routers belong to customers or other ISP's. If internal routers are connected to one or more external routers, they are called as edge routers, otherwise they are referred to as transit routers. In principle, the controller can be implemented on any internal (transit or edge) router or at a dedicated host. Agents are implemented on all edge routers. If transit routers were known to contribute a large amount of attack traffic, then the agents can be deployed on the transit routers as well and this requires no modifications to our scheme. The controller and agents are identified with their ID's. The controller assigns an ID for itself and a unique ID for each agent.

During the time of an attack, the victim requests the controller in its domain to prevent the attack. A session is established between the victim and the controller after proper authentication of the victim. Depending on the number of agents present within its domain, the controller will generate and issue the controller ID and unique agent ID to each agent and commands its agents to mark the traffic to the victim. Now the controller updates the victim with the controller ID and the unique agent IDs. The agents filter the traffic that are destined to the victim and mark the traffic with controller ID and its unique agent ID in the fragment ID field. Packets will be marked in such a way that only the first agent that sees the traffic will mark the packet. If an agent receives a packet that is already marked then it checks the packet for a valid controller ID. Packets with valid controller ID are passed and the rest are dropped. All the fragments and packets that are marked by an attacker will be dropped at this stage. Since agents are deployed on all the edge routers, all the traffic to the victim is marked with the controller ID and the ingress/first agent ID in the fragment ID field. As we have assumed that agents are deployed only on the edge routers, the traffic originating in the backbone will be marked by the egress agent of the ISP that is connected to victim's network. Since the victim knows the controller ID and valid agent IDs, it can identify different attack signatures based on agent ID. Now the victim updates the controller with different attack signatures that are to be prevented at different agents. The controller retrieves the 32-bit IP address of the agent based on agent ID and commands that particular agent to prevent the attack traffic from reaching the victim. As attack signatures are identified based on the agent ID, only the agent through which the attack traffic is passing will receive this command. Now all the agents that receive this command will start preventing the attack traffic from reaching the victim. Only the traffic that is matching with the attack signature will be dropped and logged at the agent. The traffic that does not match the attack signature will be marked with the controller ID and agent ID and destined to the victim. This is to enable the victim to easily track the changes in attack traffic. The agents will update the controller on how much attack traffic they are receiving. Prevention will be done until the agent receives a reset signal from its controller.

2. EXTENDED MODEL

We now extend our model to prevent the attack in multiple ISP domains. Each ISP domain is to have a controller. The controller maintains the database of all the agents in its domain and the controller's in the other domains. Whenever there is DDoS attack, the prevention process will initially begin within the victim's domain. If the attack persists for a long time or if the victim requests to prevent the attack upstream, then the prevention can be performed in other ISP domains. Now the controller in the victim's domain requests to have a session with the controllers in other ISP domains. A session is established between the controllers after proper authentication between them. Now the controller in the victim's domain requests the controllers in other domains to mark the packets destined to the victim. To avoid overlapping of controller IDs, the controller in the victim's domain issues a unique controller ID for every controller. The decision is based upon the number of agents present for each controller. The controllers in each domain assign unique agent ID to its agents and update the controller in the victim's domain with the valid agent IDs. Now the controller in the victim's domain updates its agents with the valid controller IDs in other ISP domains and updates the victim with valid controller IDs and agent IDs for each controller. The controllers in other domain will command their agents to mark the traffic to the victim. The process is similar to the marking in the victim's domain except that the unused bit in the flags field of the IP packet is enabled in this case to indicate that the packets are marked in other ISP's domain. Now the traffic to the victim is marked in all domains. When the packets marked in other ISP's domain enters the victim's ISP domain, if the unused flag bit is enabled, then the agents in the victim's domain can identify that the packets are marked in the other ISP's domain. Since the controller in victim's domain has already updated its agents on the valid controller IDs, the agent still only needs to check for a valid controller ID to pass the packet.

Another way to implement our model is by introducing the notion of hierarchy. In this approach each ISP will have a controller. All these controllers will be implemented as agents to other controller, which we call the master controller. The master controller can be implemented at an ISP with large network or by grouping few ISP's. Whenever there is an attack, the victim can contact the controller in its ISP's domain and the prevention of attack is done only within the ISP domain. If the attack is to be prevented in other ISP domains, the controller in the victim's domain will request its master controller to prevent the attack in other ISP domains. As prevention of attack has already started in the victim's domain, the controller in the victim's domain will also update the master controller with the controller ID used in its domain to mark the packets to the victim. Now the master controller will assign a unique controller ID for the controllers in other ISP's domain and commands the controllers to prevent the attack. The prevention process is similar to the first approach. We prefer to use the second approach, as there is already an implemented (working) architecture that suits our model (Routing Arbiter [2]) with a little modification. The Routing Arbiter project is deployed at the Network Access Point (NAP) where multiple ISP's peer with each other. The routing arbiter is developed to simplify the routing process in multiple domains by taking all the policies of the ISP's into consideration. There are many tools that can be used to enhance the functions of routing arbiter. For example, there are tools that can be used to simplify the process of routing, maintain a centralised database of all the policies of ISPs and to automatically generate low-level router configurations from the high-level policy specifications.

Our model will be implemented in a hierarchy where the routing arbiter is the master controller and the backbone routers at the network access points are the agents for the master controller. These agents are the controllers for each ISP, which have their agents within their domains. There are several advantages of implementing our extended model with the Routing Arbiter. For instance, since the Routing Arbiter/ Route server does not forward any IP packets, it is itself protected from DDoS attack. Also it is possible to prevent multiple attacks on multiple victims in multiple domains.

3. CONCLUSION

In this paper we have proposed a Controller-Agent model to prevent DDoS attacks within the ISP domains and extended the scheme to be implemented in multiple ISP domains. A detailed description of our model and single ISP implementation can be found in [1]. In a separate paper, we will discuss the implementation of the model in more detail and also describe secure authentication between different entities in our architecture.

REFERENCES

1. U.K.Tupakula, V.Varadharajan, "Model and Mechanisms for Counteracting Distributed Denial of Service Attacks", Technical Report, Macquarie University, 2002.
2. D.Estrin, J.Postel, Y.Rekhter. "Routing Arbiter Architecture," http://www.isi.edu/div7/ra/Publications.

TOWARD UNDERSTANDING SOFT FAULTS IN HIGH PERFORMANCE CLUSTER NETWORKS

Jeffrey J. Evans,[1] Seongbok Baik,[1] Cynthia S. Hood,[1] William Gropp[2]

[1]*Department of Computer Science*
Illinois Institute of Technology
10 West 31st *St.*
Chicago, Illinois 60616
Email: {evanjef, sbbaik, hood}@iit.edu

[2]*Mathematics and Computer Science Division*
Argonne National Laboratory
9700 South Cass Avenue
Argonne, Illinois 60439
Email: gropp@mcs.anl.gov

Abstract: Fault management in high performance cluster networks has been focused on the notion of *hard faults* (i.e., link or node failures). Network degradations that negatively impact performance but do not result in failures often go unnoticed. In this paper, we classify such degradations as *soft faults*. In addition, we identify consistent performance as an important requirement in cluster networks. Using this service requirement, we describe a comprehensive strategy for cluster fault management.

Keywords: Cluster, fault management, interconnection networks, soft faults

1. Introduction and Motivation

Cluster computing systems have been rapidly evolving over the past decade. A variety of system architectures exist ranging from tightly coupled proprietary systems to loosely coupled commodity-based systems. The relatively low cost of commodity-based systems along with the availability of public domain software makes them an attractive option. In research environments, clusters are replacing supercomputers. The processing power of multiple networked PCs or workstations is utilized through parallel computing software. Computational clusters are used to tackle complex problems that require large amounts of computing resources.

Our research focuses on computational clusters. Paramount here is the concept of a "coordinated team" of nodes and their communication environment working together as a single entity. One of the key elements of a computational cluster system is the interconnection mechanism. Since the nodes do not physically share memory, they rely on message passing through the network. In order to achieve good performance in message-passing multicomputer systems, consistently low latency and high bandwidth are required. Given these stringent requirements on the network, effective fault management is critical.

The negative impact of performance degradation, which we term *soft faults*, is often greater than that of hard faults. Soft faults are *performance degrada-*

tions requiring corrective action. The requirement of corrective action is then a function of the system performance *expected* or *assumed* by an application.

For example, when a user wishes to execute an application, a *request for service* is submitted to a centralized scheduler. The user requests a subset of compute nodes for a finite period of time, thus requiring the user to know approximately how long the job will take to run. If the application execution happens to run longer than the user expected (or guessed), a timeout results. The application is terminated by the scheduler and the user may or may not receive useful data - a waste of time and resources. Conversely, if the user reserves compute nodes conservatively and the application executes in a shorter period of time (i.e., no faults or optimum performance), resources are again wasted (unused), since the compute nodes were conservatively reserved.

Inconsistent performance can cause scheduling problems at many different levels. Soft faults may cause synchronization problems when the impact of a network fault accumulates through the run. Many variables affect application performance in computational cluster networks, and further investigation is required to fully understand the impact of network faults. The area of soft fault (or degraded service) management in cluster environments will grow in importance as the area of computational Grids and Grid computing evolves.

To better understand performance degradation in the context of clusters, one must consider performance relationships in both the horizontal and vertical planes. The horizontal plane includes the causal relationships that occur between "peers" at any layer (physical, link, kernel, application, etc.). The vertical plane includes causal relationships between layers, adjacent or not, all the way up through the operating system and into the applications themselves. We are studying performance degredation on the Chiba City cluster [6] at Argonne National Laboratory.

2. Related Work

Considerable effort both in commercial products and in the research community has been devoted to traditional "hard" fault management issues in cluster environments. There have also been efforts exploring performance issues in parallel program execution. These efforts include evaluation of network effects [2, 7], performance analysis using application and kernel code instrumentation [4, 9, 11], performance prediction [8, 10], and program steering [3, 13]. Additionally, adaptive techniques have been explored for predictive signaling and control in cluster environments for performance management [12] and in highly distributed networks for use in fault management [5]. Our focus, however, is on soft faults. Specifically, we wish to understand the mechanisms behind network contributions to soft faults and to identify ways to signal or ultimately control such faults.

3. The Problem of Cluster Fault Management

As high-performance systems, clusters require strong performance from each component of the system, including the application, the operating system, and the communication network itself. Additionally, when determining how to distribute processing across the nodes of a cluster, parallel computing software assumes consistent network performance. Therefore, the type of service required from cluster interconnection networks is different from that for traditional best effort or telephone networks. In best effort networks, applications tolerate variations in network performance, and real-time fault management primarily focuses on hard faults. In telephone networks, faults are tightly coupled to voice service. In clusters, however, good system performance is required to execute a large-scale application in a timely fashion. Hence, network performance degradations (soft faults) need to be addressed along with link and node failures (hard faults).

This distinction is necessary because of the time scale of action. Once detected, hard faults are corrected. Soft faults, however, are generally tolerated in the short term and may be monitored for longer-term trends. The goal of cluster fault management is to address both hard and soft faults to maintain consistent network performance. The execution of a parallel application is complex by definition. Performance tuning to achieve the optimum balance between computation and communication for a given data set can be both time consuming and unproductive. Tuning models depend on the computation and communication speeds of the hardware and software as well as the specifics of the application data set. Another major factor that is more difficult to incorporate into the models is run-time environment.

Cluster fault management can be used to maintain good system performance in two different ways. First, fault management techniques can detect and correct soft faults, thereby maintaining consistent network performance. In addition, when correction is not possible, feedback can be provided to the parallel computing software, allowing a more accurate description of current network conditions to be reflected in the modeling.

4. Summary and Ongoing Work

This paper has described cluster fault management in terms of both hard and soft faults. We defined a soft fault as a degradation resulting in inconsistent performance. Network behaviors impacting performance include localized hot spots, dropped packets, retransmissions or unordered messages, routing effects, and delayed transmissions because of flow control. Ongoing research is in two directions, (1) understanding the propagation or impact of soft faults and (2) developing mechanisms to detect and correct soft faults.

To better understand the impact of soft faults on cluster applications and other system components, we are currently running experiments on the Chiba City cluster at Argonne National Laboratory. In area (1), we are exposing horizontal and vertical performance relationships that can be cast into classes of soft faults. Once cast, these relationships can be further explored to better understand their causes, propogation and impact on the overall cluster system. In area (2), we are developing new adaptive routing techniques for Myrinet interconnection networks [1]. Myrinet uses source routing, with minimal intelligence and monitoring within the network, so existing techniques cannot be used.

Our next step is to extend low level network monitoring capabilities to better understand component issues. These tools will be used in conjunction with system and application monitoring tools to enhance our understanding and develop strategies for adaptive cluster management.

Acknowledgments

This work was supported in part by the U.S. Department of Energy, under Contract W-31-109-Eng-38 and NSF 9984811.

References

[1] S. Baik, C. Hood, and W. Gropp. Prototype of am3: Active mapper and monitoring module for the Myrinet environment. In *Proceedings of the HSLN Workshop*, Nov. 2002.

[2] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. Eicken. Logp: Towards a realistic model of parallel computation. In *Proceedings of the Fourth ACM SIGPLAN Symposium on Princples and Practices of Parallel Programming*, May 1993.

[3] W. Gu, G. Eisenhauer, and K. Schwan. Falcon: On-line moniroting and steering of parallel programs. In *Ninth International Conference on Parallel and Distributed Computing and Systems (PDCS'97)*, Oct. 1997.

[4] J. Hollingsworth and B. Miller. Dynamic control of performance monitoring on large scale parallel systems. In *International Conference on Supercomputing*, July 1993.

[5] C. S. Hood and C. Ji. Proactive network-fault detection. *IEEE Transactions on Reliability*, 46(3):333–341, September 1997.

[6] Argonne National Laboratory. Chiba City, the Argonne scalable cluster, 1999. http://www-unix.mcs.anl.gov/chiba/.

[7] R. P. Martin, A. M. Vahdat, D. E. Culler, and T. E. Anderson. Effects of communication latency, overhead, and bandwidth in a cluster architecture. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, pages 85–97, June 1997.

[8] C. Mendes and D. Reed. Performance stability and prediction. In *IEEE International Workshop on High Performance Computing (WHPC'94)*, March 1994.

[9] D. M. Ogle, K. Schwan, and R. Snodgrass. Application-dependent dynamic monitoring of distributed and parallel systems. *IEEE Transactions on Parallel and Distributed Systems*, 4(7):762–778, July 1993.

[10] J. M. Orduna, F. Silla, and J. Duato. A new task mapping technique for communication-aware scheduling strategies. In *International Conference on Parallel Processing Workshops*, pages 349–354, 2001.

[11] D. A. Reed, R. A. Aydt, R. J. Noe, P. C. Roth, K. A. Shields, B. W. Schwartz, and L. F. Tavera. Scalable performance analysis: The pablo performance analysis environment. In *Proceedings of the IEEE Computer Society Scalable Parallel Libraries Conference*, October 1993.

[12] J. Vetter and D. Reed. Managing performance analysis with dynamic projection pursuit. In *Proceedings of SC'99*, November 1999.

[13] J. Vetter and K. Schwan. Progress: A toolkit for interactive program steering. In *Proceedings of the International Conference on Parallel Processing*, August 1995.

SHORT PAPER SESSION 2

Tools and Information Models

Co-Chairs: Hanan Lutfiyya
University of Western Ontario, Canada

Takeo Hamada
Fujitsu Laboratories of America, USA

RDF-BASED KNOWLEDGE MODELS FOR NETWORK MANAGEMENT

Jun Shen and Yun Yang

Center for Internet Computing and E-Commerce, School of Information Technology, Swinburne University of Technology, PO Box 218, Melbourne, Australia, 3122

{jshen, yyang}@it.swin.edu.au

Abstract: SMIng (next generation structure of management information), an information model for network management, is a prospective structure of management information. When deploying the multi-agent systems to network management environments, we have established a lightweight self-contained knowledge model based on RDF (Resource Description Framework) and its extensions. We also present an implementation prototype to support agent communication and coordination by RDF-based languages.

Key words: information modeling, mobile agents, knowledge frameworks

1. INTRODUCTION

Mobile and intelligent agents play active roles in network management platforms and products nowadays. Meanwhile, new information models and protocol interfaces are emerging within the Internet communities, for example Script MIB (Management Information Base) [1]. The next generation structure of management information has become critical and the work towards SMIng [2] is in progress.

However, agent communications are critical when deploying multi-agents system to network management platforms. In our previous prototype [3], KQML [4] was used among managing agents by taking advantage of JatLite toolkit (java.stanford.edu). The basic contents of agents' dialogs include script codes and attribute-value pairs, but they lack sufficient support to describe relations and semantics of either managing agents or managed agents.

In this paper, we discuss how to construct a self-contained knowledge model based on the RDF (Resource Description Framework) specifications [5] and their extensions. Our seed information model is SMIng, which is independent of ASN.1 but explicitly defines terms that had been derived from former versions of structure of management information (SMI) [6]. SMIng is devised as a long-term network information model and has a minimal but complete set of data types [3].

Nevertheless, intelligent management agents should understand each other through a language with more formal semantics.

In comparison to other content languages, the triples of RDF statements in XML syntax describe relations between resources and properties naturally and flexibly. RDF and RDF Schema (RDFS) have absorbed theories of object-oriented programming, relational databases and knowledge representations and well adapted to semantic Web. The most important enrichment of RDF used by us is OIL-Ontology Inference Layer [7] and its extension and integration with agent language DAML [8], as well as RDF Context [9] and FIPA-RDF [10]. At current stage, similar to Common Information Model (www.dmtf.org), XML has been incorporated to specify DTD or schema of SMIng (www.ibr.cs.tu-bs.de/projects/sming). Therefore, XML versions of SMIng provide a basic tag vocabulary to link up a more complex management knowledge model.

2. RDF DESCRIPTIONS OF SMING MODULES

All object variable resources of the SNMP architecture can be described in the RDF framework. RDFS description is modeling SMIng modules while the RDF model specification modeling SMIng instances, which is actually MIB.

Every SMIng module has its namespace, which is identified by its authors' organisation and its version. We define a namespace *xmlns:sming* in order to describe meta classes. Besides seven basic data types, other data types are defined as subclasses. The common statements within the SMIng parameter blocks, such as 'default', correspond to properties like *rdfs:comment.* These properties share the same *rdfs:domain* as *rdfs:Class* and *rdfs:range* as *rdf:Literals*. Except *zeroDotZero*, every node may have its corresponding identifier like 'Parent.Key', where 'Parent' is the identifier of its parent node. A 'scalar' statement will be in the following RDFS form – a whole table will be a nested form:

```
<rdfs:Class ID="ScalarVariableIdentifier">
  <rdfs:subClassOf rdf:resource="ParentIdentifier"/>
  <rdfs:subClassOf rdf:resource="#DataType"/>
</rdfs:Class>
```

Instances of SMIng modules are implementations of MIB, which is abstracted as a set of statements that declare values of managed object variables within a specific managing or managed entity at a specific time point. We introduce properties *sming:time* and sming*:value* with *rdfs:domain* as *ScalarVariableIdentifier* or *ColumnIdentifier*. The RDF description of instances of column objects will be a nest structure of *rdf:Seq*, *rdf:li* and *rdf:Bag* of similar instance values of class *ScalarVariableIdentifier* (*xmlns:agent* is a reification of *xmlns:sming*):

```
<agent:ScalarVariableIdentifier>
            <sming:time>yy:mm:dd:hh:mm:ss</sming:time>
            <sming:value>value of certain DataType</sming:value>
</agent:ScalarVariableIdentifier>
```

More specification details are discussed in [11].

3. IMPLEMENTATION OF KNOWLEDGE BASES

Relational MIB, which is developed based on traditional SMI, may be extended to the management knowledge base with rich semantic capabilities of RDFS.

Assuming every management agent has a predicate set, rules set and action scripts set, we apply the RDF context and FIPA-RDF to describe agent's knowledge base. *sming:value* becomes a basic predicate of MKB, while a description of a MIB variable instance becomes a proposition specification of subject-predicate-object and truth-value relationship (*fipa:Proposition*).

The rich predicates of MKB may replace *sming:value* in order to describe more complex relationships between managed resource objects. Similarly, the operations on managed objects are expanded with new management action scripts. Compared with Script MIB, our knowledge base system for network management can provide more diversified functions with more flexibility [11]. Within FIPA-RDF, the rules are regarded as compositions of two parts: selection and manipulation. *fipa:selection* selects resources according to the specific expressions with the SQL-like RDF Query specification (www.w3.org/TandS/QL/), and *fipa-manipulation* describes corresponding actions. The management process of every agent is the replicated applications of *fipa:Rule* or *sming:Rule* to invoking *fipa:Action* so as to operate on sets of *fipa:Proposition*, which are defined by *rdfc:asserts* or *rdfc:assumes* within a certain *rdfc:Context*.

Our prototype is based on MCT (Mobile Code Toolkit) [12] and JMX. KQML messages become carriers of contents of dialogs between agents. The KQML message parameters are redefined, *:ontology* becomes SMIng modules and *:language* should be languages of RDF and its extensions. Besides SiRPAC (Simple RDF Parser and Complier), there are also some available toolkit packages for processing the RDF syntax, for example, RDF API (www-db.stanford.edu/~melnik/), and Jena (www-uk.hpl.hp.com/people/bwm/rdf/jena/), which may run upon popular XML parsers such as SAX (megginson.com/SAX) as well as Xerces (xml.apache.org/xerces-j/).

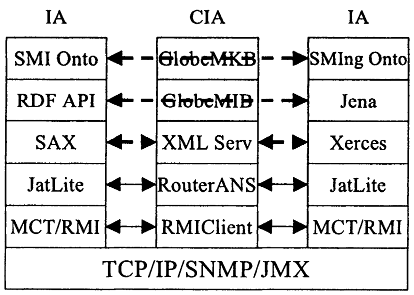


Figure 1. Multi-Agent Implementation Model

The whole implementation model is shown in Figure 1, where Router/ANS helps to route messages. The CIA (coordinating intelligent agent) maintains the global knowledge of a group of intelligent agents (IA), they communicate with each other in SMI or SMIng ontology. The dialogs are encapsulated packets with the headers of RDF(S) and XML as well as the performatives of KQML. RDF-enabled agents can understand each other by explicit semantics, for example, special requests for coordination and inferences are added-values to traditional network management frameworks, which solely query and manipulate on static variables.

4. CONCLUSIONS

Knowledge representation among management agents is a critical issue. We attempt to establish a lightweight knowledge model based on RDF. With SMIng acting as a seed, mapping from SMIng modules and related MIB (management information base) onto RDF schema (RDFS) definitions of classes, properties and related descriptions has become feasible. Moreover, elements of the management knowledge base, especially, rule bases and action scripts can be described by RDF Context, FIPA-RDF and OIL. Our implementation model integrates Java based tools at different levels to coordinate agents more effectively.

ACKNOWLEDGEMENTS

This work is partly supported by Swinburne Vice Chancellor's Strategic Research Initiative 2002-2004. The early prototype was implemented at Southeast University and funded by Chinese National Science Foundation (No. 90204009).

REFERENCES

[1] J. Schönwälder, J. Quittek, C. Kappler, Building distributed management applications with the IETF Script MIB. IEEE Journal on Selected Areas in Communications, Vol.18, No.5, pp.702-714, 2000

[2] C. Elliot, D. Harrington, J. Jason, J. Schönwälder, F. Strauß and W. Weiss, SMIng objectives. IETF Request for Comments 3216, December 2001

[3] J. Shen, J. Luo, G. Gu, An object-oriented net graph model for agent group-based network management. In: Proceedings of Technology of Object-Oriented Language and Systems, (TOOLS 31), IEEE Press, pp.126-132, Sept. 1999

[4] T. Finin, R. Fritzson, D. Mckay, R. McEntire, KQML as an agent communication language. In: Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM'94), ACM Press, pp.456-463, Nov. 1994

[5] S. Decker, S. Melnik, F. van Harmelen, D. Fensel, M. Klein, J. Broekstra, M. Erdmann, I. Horrocks, 2000, The semantic web: The roles of XML and RDF. IEEE Internet Computing, Vol. 4, No. 5, pp.63-74, 2000

[6] K. McCloghrie, D. Perkins, J. Schoenwaelder, J. Case, M. Rose and S. Waldbusser, Structure of Management Information version 2 (SMIv2). IETF Standard 58, Request For Comments 2578, April 1999

[7] I. Horrocks, et al., The Ontology Inference Layer OIL. Technical Report of On-To-Knowledge Project in the IST Program, http://www.ontoknowledge.org/oil, 2000

[8] D. Connolly, et al., DAML+OIL reference description. World Wide Web Consortium Note, http://www.w3.org/TR/daml+oil-reference, 2001

[9] G. Klyne, Context for RDF information modelling. Available at http:// public.research.mimesweeper.com / RDF / RDFContexts.html, Oct. 10, 2000

[10] FIPA, FIPA Content Language Library. FIPA 99 Specification, v 0.2, Part 18, Foundation for Intelligent Physical Agents, Geneva, Switzerland, 1999

[11] J. Shen, Research on Multi-Agent System Based Network Management Models, Ph.D. Thesis, Southeast University, Nanjing, China, 2001

[12] Bieszczad, Mobile agents for network management, IEEE Communications Surveys, http://www.comsoc.org/pubs/surveys. Fourth Quarter, Vol. 1 No.1, 1998

PROCESS MANAGEMENT AND CONTROL FOR HETEROGENEOUS DOMAIN MODELS

Takeshi MASUDA
NTT Access Network Service Systems Laboratories

Abstract: The workflow of telecommunication operation support systems (telecom-OSSs), which are a typical example of a heterogeneous domain model, often involves complicated cancellation processing. This paper proposes a rule-driven cancellation method and a middleware component that implements the method. They separate the cancellation process from the workflow and enable the OSSs to be made more flexible.

Key words: process management, domain model, workflow, access network

1. INTRODUCTION

The diversification of communication services and the shortening of release periods have necessitated the use of commercial off-the-shelf (COTS) software for telecommunication operation support systems (telecom OSSs). Furthermore, when constructing a system that consists of multiple subsystems and that executes operations across all of the operation menus provided by the subsystems, the application of a workflow management system (WfMS) has been found to be effective [1].

When the subsystems developed for each domain are interconnected and the domain models for state transition and cancellation operations of the subsystems differ, cooperative processing becomes difficult. Because it is not always possible to easily change the domain models, the WfMS must be made highly flexible to enable different models and systems to cooperate through it. However, the scale and complexity of the workflow description increases when the WfMS has to handle exceptions and failures [2].

In this paper, we introduce a telecom-OSS, as a typical example of a system that has several subsystems with different domain models. We describe the problem of processing the telecom-OSS workflow. In Section 1, we show that in telecom-OSSs, this problem becomes even more serious than in a typical business flow. In section 2, we describe a middleware component we developed that implements a rule-driven cancellation method to handle exceptions and failures in workflow processing. We simulated this system and found that the total number of activity nodes on practical workflows can be reduced roughly by 50%.

2. PROCESSING THE TELECOM-OSS WORKFLOW

Telecommunication operation support systems need a workflow more complicated than that of a general business process mainly because a telecom-OSS domain consists of various related subdomains. This variety arises from many differences among the subdomains, e.g., in the network structure, equipment type, and service classification. This paper describes a system for access network element allocation (ANEA), which is an example of a telecom-OSS. The ANEA domain consists of three subdomains: (1) an outside cable, (2) an intra-office cable, and (3) access equipment (Figure 1 (a)). The internal state of each subsystem changes from an initial state to a completion state. There are also two interim states: allocated and reserved. Each subsystem has several operation menus to initiate forward processes, which move the subsystem to the next state. The host system executes the menus of the subsystems in a certain order; it also manages and controls the whole process. When the host system changes a subsystem's state back from an interim state to the initial state, a cancellation process must be performed. These transitions are modeled as the optimal transition within the subdomain. To be more precise, each ANEA subdomain needs a different amount of time to provide network elements. For example, the outside-cable subdomain needs several weeks, but the access equipment subdomain needs several seconds. Therefore, the access equipment subdomain has only one interim state (allocated), while the other subdomains have two interim states (allocated and reserved).

Figure 1 shows the effect of adding a subdomain (division or subsystem) to an ANEA system and a typical business processing system on the processing order of these systems' workflow. The processing order of a typical business flow (Figure 1 (b)) is affected only by the stable and organized order in the whole domain, e.g., the order of the official sanctioning. It is clear from the figure that the change of the order is local, and the addition of the subdomain is easy: only one flow should be added. However, adding a new subdomain (subsystem) to the ANEA system affects the ANEA workflow (Figure 1 (a)), which makes the addition all the more difficult: in this case, five flows should be added. This is because the priority between the agreement level and the subdomains is inverted. Therefore, the processing order of the ANEA flow is inherently more complicated than that of the typical business flow.

On the other hand, when viewed from the perspective of all the subsystems, the processing order consists of several phases. The internal states of all the subsystems should be synchronized to guarantee macro data consistency. The important point is that the number of times the state of each subsystem changes may differ according to the subdomain model. In this paper, these subdomain models are referred to as heterogeneous domain models (HDMs). The complexity of the HDM workflow adversely affects the process of operation cancellation. When the user cancels an operation or the system generates an exception, an appropriate combination of cancellation operations must be performed for each subdomain (subsystem). Throughout the workflow, each subdomain goes through a series of interim states. As a result, the combinations of cancellation operations needed by each subdomain change depending on the execution point in the workflow. When a WfMS controls a combination of cancellation operations, its workflow becomes far more complex than a simple combination of forward procedures to provide network elements. To simplify the workflow, we developed a status management system (StMS) described in the next section.

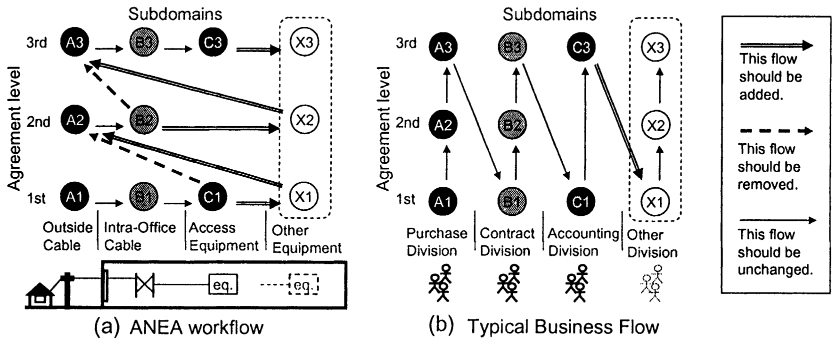


Figure 1. Comparison of the execution orders of an ANEA system and a typical business processing system and the effect of adding a new subdomain on these systems' workflow

3. IMPLEMENTATION OF STMS

The basic architecture of StMS is shown in Figure 2. The StMS is a middleware component that works together with the WfMS, and it exclusively handles cancellation processing. The StMS consists of two parts, a working space and a rule accumulator. The working space holds two categories of information as nodes in a graph structure shown in the left part of Figure 2. These categories show the execution history of forward operations of the subsystems and the execution schedule for cancellation operations corresponding to the forward operations. There are several links between the nodes indicating three types of relationships between the operations: dependency, (e.g., operations depend on one another) cancellation (e.g., operations have different cancellation operations), and inclusiveness (e.g., some cancellation operations can be substituted with other cancellation operations).

These links and nodes are built according to the generation rules prepared beforehand when the WfMS requests a subsystem to process an ANEA operation. The generation rules are stored in the rule accumulator. The rules have a condition part and an action part. These parts have several operation identifiers. An operation identifier indicates the type of corresponding operation, showing whether it is a forward operation or a cancellation operation. It is embodied as a unique value in the whole system. For example, an operation identifier can be coded as variable length character data or variable length integer data. The operation identifiers in the condition part of a rule indicate the nodes that the StMS should apply the rule to. The operation identifiers in the action part of a rule indicate the nodes that the StMS should look for or create. The rules holding a forward operation identifier in their condition part have several forward operation identifiers and several cancellation operation identifiers in their action part. The forward operation identifiers are used to create dependency links. The cancellation operation identifiers are used to create cancellation links. In contrast, the rules holding a cancellation operation identifier in their condition part have only cancellation operation identifiers in their action part. These rules are used to create inclusive links.

This basic StMS composition makes it possible to explicitly isolate cancellation processes from the forward operations in the workflow. We simulated this system and found that when both the WfMS and the StMS are used for an ANEA domain, the total number of activity nodes on 18 workflows can be reduced from 51 to 26 (Figure 3).

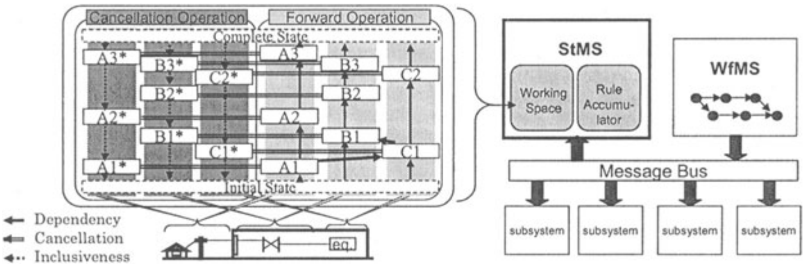


Figure 2. Basic architecture and working space in the StMS

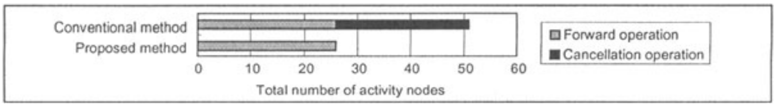


Figure 3. Total number of activity nodes on 18 workflows

4. SUMMARY

We analyzed the causes of complexity in the workflow of telecom operation support systems and clarified their workflow processing on the basis of a heterogeneous domain model. We then proposed a rule-driven method for automatic cancellation processing in this domain model, and described its implementation in the form of a status management system. The proposed method can reduce the total number of activity nodes by 51%.

REFERENCES

[1] V.P. Wade, and T. Richardson, "Workflow – A Unifying Technology for Operational Support Systems", Network Operations and management Symposium, Vol. 07, pp. 231-246, Sep. 2000.

[2] N. Edelweiss, and M. Nicolao, "Workflow modeling: exception and failure handling representation", Computer Science, 1998. SCCC '98. XVIII International Conference of the Chilean Society of, 1998, pp. 58 -67.

SEMANTIC MANAGEMENT: APPLICATION OF ONTOLOGIES FOR THE INTEGRATION OF MANAGEMENT INFORMATION MODELS

Jorge E. López de Vergara, Víctor A. Villagrá, Julio Berrocal, Juan I. Asensio[†], and Roney Pignaton
Dpto. Ingeniería de Sistemas Telemáticos, Universidad Politécnica de Madrid, Spain.
{jlopez, villagra, berrocal, jasensio, roney}@dit.upm.es
[†] *Visiting researcher from Universidad de Valladolid, Spain*

Abstract: The multiplicity of Network Management models (SNMP, CMIP, DMI, WBEM...) has raised the need of defining multiple mechanisms to allow the interoperability among all involved management domains. One basic component of such interoperability is the mapping between the information models that each domain specifies. These mappings, usually carried out with syntactical translations, can reach the semantic level by using ontologies. This article shows the advantages of using formal ontology techniques to improve the integration of current network management models.

Keywords: Ontology, Management Information Models Integration, Semantic Mapping, Behavior Information.

1. INTRODUCTION

Different standardization efforts for a unique integrated management model have set up several models: different management domains currently exist using IETF's SNMP, ISO's CMIP, DMTF's DMI or even OMG's CORBA. The definition of interoperability mechanisms among these models has become essential to perform an integrated management in a multiple domain scenario. Existing studies about this topic [1] divide the problem in issues related to their diverse communication protocols and information models: If a rule set can be specified that translates both the access and the definition of the information, interoperability is possible.

DMTF's WBEM management model includes some features to deal with this heterogeneity. In this new model, access interoperation is solved by using providers that act as gateways, but information interoperability is still an obstacle: CIM, WBEM's Common Information Model, defines three different mappings, which are technique, recast and domain [2]. However, the fact is that usual translations are at most recast mappings. Recasts only give a syntactic-equivalent definition that is not integrated in the semantic hierarchy of the CIM schemas, which could only be done with domain mappings.

Domain mappings are not easy to define, because they cannot be done automatically. To solve this problem, the knowledge representation discipline known as Ontology can be useful, because it provides all necessary constructs to add semantics to described information. In fact, some ontology tools exist that assist in the information merging and mapping task [3].

The following section introduces ontologies and compares them to CIM. Next, a method is proposed that applies ontology techniques to the integration of different management information specifications. Finally, some conclusions are presented.

2. ONTOLOGIES

An ontology can be defined as *"an explicit and formal specification of a shared conceptualization"* [4]. Briefly, it can be said that an ontology is the definition of a set of concepts, its taxonomy, interrelation and the rules that govern such concepts. In this way, existing management information models could be understood as a kind of ontologies: Models like CIM define the information of the management domain in a formal way and by common consent of working groups. However, they do not incorporate axioms or constraints that provide the additional semantics usually included in the so called heavyweight ontologies, and that would eventually enable the inference of knowledge based on existing one.

Ontologies are usually defined following a pyramidal structure in which more general and also more reusable ontologies are at the bottom level, and more usable and also more specific and less reusable are at the top [5]. CIM has a similar structure although it lacks both a General Common Ontology and any Task Ontology levels (see figure 1).

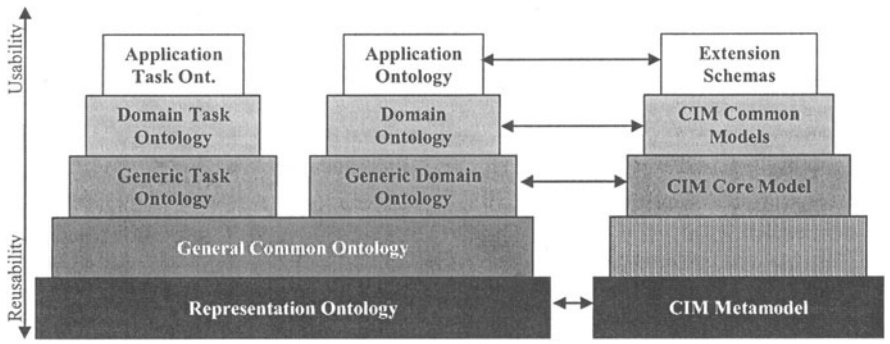


Figure 1. Correspondence between CIM and Ontology architectures.

3. APPLYING ONTOLOGIES TO NETWORK MANAGEMENT INTEGRATION

As stated before, semantic interoperability is not completely achieved in CIM and thus, it should be extended. This section proposes a method to create a network management information model based on formal ontologies. The resulting ontology would be an extension of CIM, adding the necessary axioms and constraints to obtain a heavyweight ontology. Furthermore, this CIM-based ontology should also contain all information defined in other management models. This can be achieved by merging every model with CIM, including all necessary mapping rules.

Therefore, a set of steps can be defined to obtain the desired management global ontology, which can be used by a manager as an interoperable information model:

1. Translate all management information models to work with a single ontology representation language.
2. Merge the models in a global ontology, defining at the same time mapping rules between the global ontology and each model.
3. Add a set of formulae or axioms to the ontology to make it heavyweight.

The merging and mapping tasks can be assisted by means of ontology tools. In this case, a merging of CIM and SNMP MIBs has been done with one of such tools [6], using a subset of CIM and the whole HOST-RESOURCES-MIB (see figure 2). Both MOF and SMI specifications were manually translated to be readable by the ontology tool.

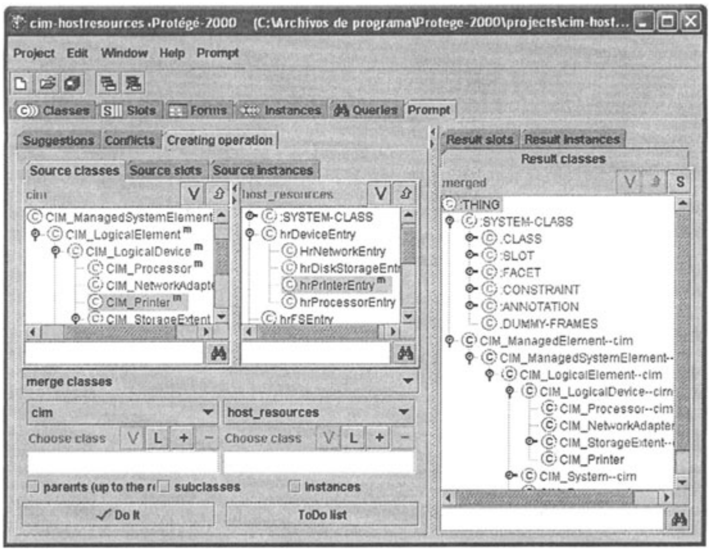


Figure 2. Merging CIM and HostResources with an ontology tool

With this approach, a manager can use the merged ontology to have a unified view of all the information it manages and translate it, by applying the mapping rules, into each domain information model taking into account the semantics of the concepts.

Constraints contained in the description of an OBJECT-TYPE or a CIM Property are defined in natural language and cannot be automatically enforced by a manager. If a formal definition of these constraints is provided, specified management information can be used to define certain manager behavior rules. Ontology tools also allow the definition of such axioms or constraints to complete the management ontology: For instance, the following rule could be set to impose that "the AvailableSpace of a CIM_FileSystem instance is going to be greater than a 10% of the FileSystemSize":

(**defrange** ?fs :FRAME *CIM_FileSystem*)
(**forall** ?fs (> (*AvailableSpace* ?fs) (* 0.10 (*FileSystemSize* ?fs))))

4. CONCLUSIONS AND FUTURE WORK

This paper has presented a novel approach in which the definition of an ontology-based management information model has been proposed. The advantages of this approach can be applied when trying to map and merge different management models from a semantic viewpoint. Also, ontology axioms and constraints provide a way to define the behavior related to the information model.

Current developments include a program that translates MOF and SMI specifications automatically to RDFS and DAML+OIL, dealing also with SMI particularities. At the same time, the ontology tool [6] is being adapted to the peculiarities of the management information when merging and mapping it.

REFERENCES

[1] Pramod Kalyanasundaram, Adarshpal S. Sethi, "Interoperability Issues in Heterogeneous Network Management", in *Journal of Network and Systems Management*, Vol. 2, No. 2, June 1994.

[2] Distributed Management Task Force, Inc., *Common Information Model Specification version 2.2*, DMTF Document DSP004, June 1999.

[3] Jorge E. López de Vergara, Víctor A. Villagrá, Julio Berrocal, "Semantic Management: advantages of using an ontology-based management information meta-model", in *Proceedings of the HP Openview University Association Ninth Plenary Workshop (HP-OVUA'2002)*, Böblingen, Germany, June 2002.

[4] R. Studer, V.R. Benjamins, D. Fensel, "Knowledge Engineering: Principles and Methods", in *Data & Knowledge Engineering*, Vol. 25, 1998.

[5] Asunci6n G6mez Pérez, V. Richard Benjamins, "Overview of Knowledge Sharing and Reuse Components: Ontologies and Problem-Solving Methods", in *Proceedings of the IJCAI-99 Workshop on Ontologies and Problem-Solving Methods (KRR5)*, Stockholm, Sweden, August 1999.

[6] Natalya F. Noy, "Managing Multiple Ontologies in Protégé-2000", in *Proceedings of the Fifth International Protégé-2000 Workshop*, Newcastle, England, July 2001.

A CONCEPTUAL FRAMEWORK FOR BUILDING CIM-BASED ONTOLOGIES

Emmanuel Lavinal, Thierry Desprats and Yves Raynaud
IRIT-UPS, UMR 5505 CNRS. 118 Route de Narbonne, F-31062 Toulouse Cedex 4, France
{lavinal, desprats, raynaud}@irit.fr

Abstract: Cooperative Management has appeared as a promising paradigm on which Network and System Management (NSM) solutions should be partially or entirely based. This paper proposes a conceptual framework to obtain, from any CIM schemas, NSM frame-based ontologies which can then be used in cooperative management solutions.

Key words: Information models, cooperative management, CIM, ontology

1. INTRODUCTION

From a technological viewpoint, Multi-Agents Systems (MAS) constitute the natural candidate to support the implementation of this cooperative management paradigm [1, 2]. In such a context, autonomous entities interact in a cooperative way to reach a common objective of NSM. This implies that the involved entities understand each other and share a common representation of the domain knowledge. More precisely, these entities interact by exchanging primitives specified by an Agent Communication Language (ACL) like FIPA-ACL. These messages, called performatives, convey information about illocutionary power of the interaction within the dialogue (e.g. assertion, question, order, etc.); this information, devoted to protocol aspects, is totally independent from the application domain. Another conveyed information is directly related to the domain because it defines the real semantics of the message. A shared vocabulary is required to represent the "universe of discourse" by specifying the common concepts, attributes and relationships which describe the target application domain. This vocabulary is called an ontology and is used by any agent to express its knowledge about its environment [3, 4].

If we want to promote the use of MAS in the NSM domain, having ontologies related to this domain is absolutely necessary. As there are none existing at the moment, our objective is to provide a generic and automatic way to obtain NSM ontologies which then should be used in a cooperative NSM context. Rather than trying to specify one NSM ontology able to cover all the needs in this field, our idea was to benefit from the very important works done by the DMTF: the CIM effort currently appears as the most successful and recognized approach.

The conceptual framework we propose is based on a precise mapping which can be applied to build NSM ontologies from existing CIM schemas. We show how it is possible to express the CIM Meta Schema thanks to the OKBC Knowledge Model, and therefore, how to obtain frame-based NSM ontologies from any CIM schema.

2. OKBC AND CIM OVERVIEW

OKBC (Open Knowledge Base Connectivity) [5] is a standard for frame-based ontology. It specifies a knowledge model (KM) which provides a set of constructs commonly found in frame-based knowledge representation systems. It includes frames, classes, slots and facets. A *frame* is a primitive object that represents an entity in the domain of discourse. A frame that represents a class is called a *class frame*, a frame that represents a slot is called a *slot frame,* etc. Basically, *classes* are similar to object-oriented approaches, *slots* describe properties or attributes of classes and *facets* describe properties of slots. Protégé-2000 [6] is an OKBC-compatible ontology editing and knowledge acquisition environment. The use of metaclasses makes Protégé-2000 easily extensible and enables its use with other knowledge models. Due to its capacity to be expressed in a formal language like KIF [7] and to its numerous implementations, we selected OKBC as the target environment to obtain NSM CIM-based ontologies.

CIM [8] is a conceptual information model for describing management that is not bound to a particular implementation. CIM defines a collection of schemas which provides the actual model descriptions. The CIM Schema is structured into three distinct layers: the *core model*, the *common models* and the *extension models*. These generic schemas constitute a very important work for vendors who can model their management information on a common and standard basis. CIM also specifies a meta schema which defines the terms used to express the model and its usage and semantics. Some elements of the meta model (MM) are descended from object-oriented approaches (Classes, Properties, Methods, Associations, References) and others descend from management or data base domains (Qualifier, Trigger, Indication).

3. THE CONCEPTUAL FRAMEWORK

On the one hand, ontologies and CIM serve very different purposes: ontologies allow sharing and reuse of knowledge to provide a common understanding of some domain whereas CIM modelizes management information to provide integrity constraints for management systems. On the other hand, ontologies and CIM have some common points: they both provide vocabulary and structure for describing particular information and they also have similar modeling levels (Figure 1). It is therefore legitimate to compare both. We chose the highest level of abstraction to perform our mapping (meta model level), that is between the OKBC knowledge model and the CIM meta schema. This choice is motivated by the fact that we wanted to provide a high level of genericity to exploit this work in all the underlying levels, allowing therefore the construction of ontologies from any CIM model.

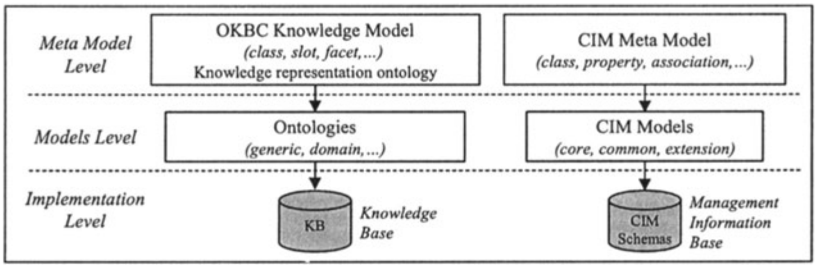

Meta Model Level
OKBC Knowledge Model (class, slot, facet,...) Knowledge representation ontology
CIM Meta Model (class, property, association,...)
Models Level
Ontologies (generic, domain,...)
CIM Models (core, common, extension)
Implementation Level
KB
Knowledge Base
CIM Schemas
Management Information Base

Figure 1. OKBC vs. CIM modeling levels

We first made a comparative analysis between the data types, concepts and standard elements defined in the two approaches, and then we established a set of precise correspondences at a modeling level to reconcile the differences. We will briefly expose our work on the concepts.

We identified three categories of concepts: (*i*) concepts that are exactly the same, (*ii*) concepts that are the same but expressed in a different way and finally (*iii*) concepts of the CIM meta model that do not have an equivalent in the OKBC knowledge model. These categories were concealed either by direct equivalences or by the extension of the OKBC knowledge model to express the new CIM meta model concepts. For example, classes, slots as properties and inheritance are concepts exactly the same in the two approaches. Concerning category (*ii*), we can quote the concept of association that can be represented as a special slot in OKBC whereas CIM defines an association as a class. Therefore we created a metaclass CIM:ASSOCIATION to respect the CIM specification. Finally, the concepts of qualifier, method, trigger, etc. fall in the category (*iii*): we defined new metaclasses and metaslots that we configured to reflect exactly the CIM concepts.

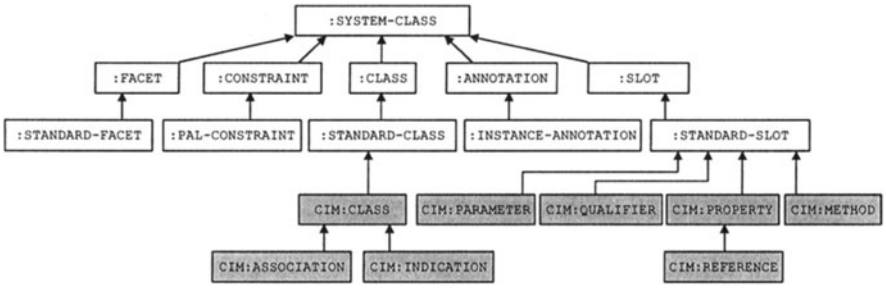


Figure 2. Specialization of the KM of Protégé-2000 to express the MM of CIM

The result of this work is a complete expression of the meta model of CIM in the OKBC knowledge model. We applied this work in the environment Protégé-2000, adding metaclasses and metaslots to the system architecture. Figure 2 illustrates the Protégé knowledge model extended by the new concepts descended from CIM.

4. CONCLUSION AND FUTURE WORKS

The presented work allows us to express the CIM Meta Schema as a frame-based ontology. This has been achieved by extending the OKBC knowledge model in the Protégé-2000 environment in order to specify all the concepts proper to CIM. We also realised a full mapping concerning both the data types and the CIM standards qualifiers. This precise and complete projection has been done at the respective "Meta" levels.

Based on that extended knowledge model, we have obtained the expression of the entire standard CIM "Network" schema as a Protégé-2000 ontology. This result shows that the mapping defined at the "Meta" level may constitute a solid basis to obtain NSM-related ontologies from any existing CIM models. Both the Core and Common CIM schemas defined by the DMTF may be expressed in such a way as ontologies. In the same manner, all the CIM extension schemas which have been (or will be) defined by all the categories of CIM users may be transformed into ontologies. By providing a "Meta-mapping" solution, we have reached a high level of genericity to easily reuse the whole efforts made around the CIM modeling approach: any CIM-based model can become a NSM frame-based ontology.

Current works intend to automate the mapping process. Based on our proposal, a tool which allows to generate a Protegé-2000 ontology from the analysis of a CIM-XML file (describing CIM schema) is under construction. This tool will help to integrate CIM models as ontologies within cooperative NSM solutions without requiring any expertise on ontologies.

REFERENCES

[1] J.P. Martin-Flatin, S. Znaty. Two Taxonomies of Distributed NSM Paradigms, Emerging Trends and Challenges in Network Management, Chapter 3, June 2000

[2] K. Boudaoud, Z. Guessoum, C. Mc Cathie Neville, P. Dubois. Policy-based Security Management Using a Multi-Agent System, Workshop HPOVUA, Berlin, June 2001

[3] A. Gómez-Pérez. Ontological Engineering: a State of The Art, Expert Update, 1999

[4] C. van Aart, R. Pels, G. Caire, F. Bergenti. Creating and Using Ontologies in Agent Communication, Workshop on Ontologies in Agent Systems, Italy, July 16, 2002

[5] V.K. Chaudhri & al. Open Knowledge Base Connectivity 2.0.3, April 1998

[6] N.F. Noy, R.W. Fergerson, M.A. Musen. The knowledge model of Protégé-2000: combining interoperability and flexibility, EKAW'2000, France, October 2-6, 2000

[7] Knowledge Interchange Format. Draft proposed American National Standard: http://logic.stanford.edu/kif/dpans.html

[8] Common Information Model Specification. DMTF, http://www.dmtf.org/standards/

POLICY-BASED COOPERATION OF SERVICES IN UBIQUITOUS ENVIRONMENTS

Toshio Tonouchi, Tomohiro Igakura, Naoto Maeda, Yasuyuki Beppu, and Yoshiaki Kiriha
Network Laboratories, NEC

Abstract: Various kinds of nodes, including cellular phones and information appliances, are to become popular and are expected to provide a variety of services. Cooperation of these services will result in more convenient services than keeping them isolated would. A ubiquitous network is characterized by changeable system configurations. Because of this and the fact that a node is so frequently connected to and disconnected from the network, the global cooperation of services is difficult to describe in flow languages such as Web Services Flow Language (WSFL). One of the solutions to this problem is a policy technology. A policy attached to a node can be added or removed when the node is connected or disconnected. The policies can re-configure a changed system.

Keywords: Management of Grid Computing, Clusters, Peer-to-Peer Applications, and Ubiquitous Computing Environments, Policy, Message-oriented system

1. INTRODUCTION

The ubiquitous network environment is maturing. Cellular phones with Internet access, personal data assistants (PDAs), and wireless local area networks (LANs) are becoming more and more popular. About 10 years ago, Weiser developed an original PDA called 'Tab' and invented a proprietary protocol for wireless communication [1].

Some ubiquitous nodes, especially information appliances, provide simple services. For example, an air conditioner with network access function can be turned on and off or can have the temperature set by a remote user. Cooperation of these simple services provides a valuable service. For example, a cellular phone with a global positioning system (GPS) can automatically give the location of the user to the network-connected air conditioner, which is automatically set to turn on when the user (e.g. Tom) comes near his house.

One of the characteristics of ubiquitous networks is that some of the ubiquitous nodes constituting the systems are not always operational. A cellular phone may be off when the battery is dead or the network-connected air conditioner breaks down. We call this characteristic *fickle*. A fickle node may suddenly disappear, and the system suddenly stops due to this. For example, when the air conditioner breaks down, the cellular phone cannot communicate with the air conditioner. A fan should work instead of the air conditioner when the air conditioner breaks down.

We propose a policy-controlled message-oriented system that overcomes the fickle-node problem.

2. RELATED WORK

A partial solution to the fickle-node problem is a publisher-subscriber system[2]. A publisher-subscriber system has a message router that automatically forwards a message to some of the nodes registered with the message router. Stopped and disconnected nodes will be manually unregistered. They can forward a message to adequate nodes in normal cases but they cannot handle the message when an error occurs. It is, therefore, difficult for the publisher-subscriber system to realize the example of the broken air-conditioner, which replies an error message.

Web Services Flow Language (WSFL[3]) and XLANG[4] were interesting trials for specifying the workflow among Web services. However, these technologies encounter the fickle-node problem because the description in the control flow languages requires deterministic routing information. The unplanned appearance and disappearance of nodes totally affects the workflow. The programmer therefore must rewrite the workflow.

3. ARCHITECTURE

Our architecture is basically the same as that for publisher-subscriber systems. The architecture is shown in Figure 1. A message router called the distributor is connected to a network. All the messages that the ubiquitous nodes (e.g., personal computers, PDAs, and cellular phones) send go through the distributor. The distributor determines where the messages go next.

Just as for the publisher-subscriber message-oriented systems, nodes that receive messages must be registered with the distributor a priori. In our system, policies describing which messages the joined nodes accept are also registered with the distributor.

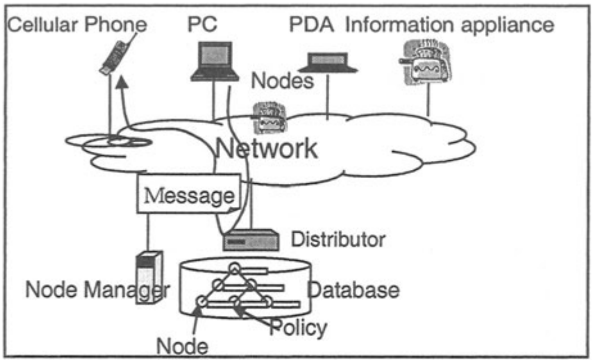


Figure 1. Architecture

Policies are the key to our architecture. We give an example of policies in Figure 2 (a). A policy is composed of a matcher (before "|") and a generator (after "|"). A matcher specifies what kind of message a node accepts. P_1 and P_2 accept any message because the matcher does not specify any condition. P_1 and P_2 could even accept the same message. However, the distributor non-deterministically chooses one of them.

The interesting syntax of our policy language is the generator. Generator "*" creates a copy of an accepted message and distributes it to the other policies. Suppose that P_1 accepts a message. P_1 forwards it to Node N_1. P_1 then generates an internal message copied from the original message with attribute "after=P_1". An internal message is a pseudo message that is used to explain the behavior of the policy processing of the distributor. Policy P_2 accepts the generated internal message because P_1 has already been used and only P_2 can be matched with the internal message. Next, suppose that P_2 is matched earlier than P_1. P_1 will match the message generated by P_2. In either case, N_1 and N_2 are chosen in the case of the example in Figure 2 (a).

N_3 is a 'fickle' backup server, whose policy, P_3, accepts messages that include "after=P_2". This means that a message to Node N_2 is copied to the backup server. N_1 and N_2 work well even if fickle node N_3 is removed. Only the backup function does not work. However, the backup of N_2 will work automatically when N_3 is connected to the distributor with Policy P_3. This shows that our policy approach solves the fickle-node problem.

Three nodes (N_1, N_2, and N_3) are connected to Policies P_1, P_2, and P_3. P_1 := | * after=P_1 P_2 := | * after=P_2 P_3 := after=P_2 | *	P_a is connected to the air-conditioner and P_f is a policy to the fan. P_a := sender=tom's-phone distance =10? P_f := message=error receivers=air-conditioner
(a) Back-up service	(b) Air conditioner and fan

Figure 2. Examples of policies

Figure 2 (b) shows the policies of the example in Section 1. Policy P_a is fired when the distance between Tom's cellular phone and his house is less than 110 meters and more than 100 meters ("distance=10?"). If the air conditioner is broken, the error message is issued. The P_f is fired because it matches the error message. Notice that both P_a and P_f have no generator. These do not generate internal messages, and no more policy is fired.

4. CONCLUSION

We proposed a policy-based message system. We showed, using the examples, that this system solves the fickle-node problem. Nevertheless, the syntax and semantics of our policy language are inadequate. We are trying to improve the policy language without losing its simplicity. Another challenge is the effectiveness of policy processing. A distributor may have to handle a lot of ubiquitous nodes. In such a situation, fast policy processing is required. Therefore, we are now studying an optimization method for policy processing. The correctness of the optimization method is proved based on the operational semantics of our policy language.

REFERENCES

[1] Weiser, M.: *Some Computer Science Issues in Ubiquitous Computing*, Communication of the ACM, Vol. 36, No. 7, pp.74-84, July 1993

[2] Sun Microsystems, Inc: *Java Message Service*, 1999

[3] Leymann, F.: *Web Services Flow Language (WSFL Ver 1.0)*, May 2001

[4] Thatta, S.: *XLANG – Web Services for Business Process Design*, 2001

DESIGN AND IMPLEMENTATION OF AN INFORMATION MODEL FOR INTEGRATED CONFIGURATION AND PERFORMANCE MANAGEMENT OF MPLS-TE/VPN/QOS

Taesang Choi, Hyungseok Chung, Changhoon Kim and Taesoo Jeong
ETRI, {choits, chunghs, kimch, tsjeong@etri.re.kr}

Abstract: Multi Protocol Label Switching (MPLS) is generally considered a mature technology. Many Internet Service Providers (ISPs) and telecommunication carriers have deployed it or are considering deploying it. An easy-to-use integrated management solution is requested by these ISPs. To realize a truly integrated management solution, a combined management information model is essential. In this paper, we propose an information model for integrated configuration and performance management of MPLS-Traffic Engineering (MPLS-TE)/VPN/QoS.

Key words: Information Model, MPLS-TE, MPLS-VPN, Diffserv, Configuration Management, Performance Management

1. INTRODUCTION

As of today, Multi Protocol Label Switching (MPLS)[1] is considered as a mature technology. Many Internet Service Providers (ISPs) and telecommunication carriers have deployed it or are considering deploying it for various reasons: efficient usage of their valuable network and system resources and meeting customer's emerging service requirements such as provider managed IP virtual private networks (VPNs) and quality of service (QoS) guaranteed IP services for voice, video or mission critical applications.

One of the major requirements for a successful deployment is easy, efficient, scalable and reliable management of networks and services based on MPLS. This includes automated provisioning, network and service performance monitoring, fault management and billing. And service providers want a more general management solution and, thus, a common integrated information model for all these management functionalities is needed more than ever before.

In this paper, we propose an information model for integrated configuration and performance management of MPLS-TE/VPN/QoS. It is an OO-based abstract information model which means it is independent from the existing data models, encoding schemes and management protocols. In our proposal, we defined an information model by using Unified Modeling Language (UML) [2] which is used most widely to describe OO-based information models.

2. DESIGN OF THE INFORMATION MODEL

The main focus of the proposed information model is OO-based object model, protocol independency and integration of configuration and performance management. Our information model consists of four sub-models: one for MPLS-TE, MPLS-VPN, Diffserv[3]-based QoS, and Topology. Each sub-model is divided into configuration and performance parts. Topology sub-model is common to all three services. IP layer topology is a common denominator of MPLS-TE, MPLS-VPN and QoS topologies. Given the IP layer topology, MPLS-TE, MPLS-VPN or Diffserv topological information can be further added depending on an underlying network's capabilities. For example, if the underlying network supports MPLS-TE then both IP and MPLS-TE topological information is captured in the same topology information model. MPLS-TE sub-model models configuration and performance management information of MPLS-TE such as a traffic trunk, Label Switched Path (LSP) tunnel, LSP Path and traffic statistics of LSP tunnels. Similarly MPLS-VPN and QoS sub-models define conceptual management information and their relationship of respective services' configuration and performance functionalities. Due to the limited space, we only describe one sub-model: MPLS-TE.

2.1 Information Model for MPLS-TE

Figure 1 shows the information model for MPLS-TE configuration management. It shows required object classes and their relationships. There are three important classes: a traffic trunk, an LSP tunnel, and an LSP. The traffic trunk models the one defined by RFC2702, "The requirements for traffic engineering over MPLS" [4]. It represents an aggregate of customer traffic flows belonging to the same service class or classes. It can be mapped into zero or more LSP tunnels for load sharing purposes. Each LSP tunnel is mapped into one or more LSPs which are represented by dynamically calculated paths or explicitly specified paths. Other classes are tightly coupled with these main classes. For example, a traffic trunk (TtC) and LSP tunnel (LspTunnelC) need forwarding equivalence class (FEC) and a RSVP traffic profile (RsvpTp). FEC is a traffic classification filter and RsvpTP is a traffic profile attribute specification such as bandwidth, delay, jitter, etc.. Classes with letter "C" like TtC contain configuration and static information only. When a new class object is created, three objects (with letter "C", "M", and "S") are created at the same time and stores configuration-, performance-, and simulation-specific information separately for information consistency.

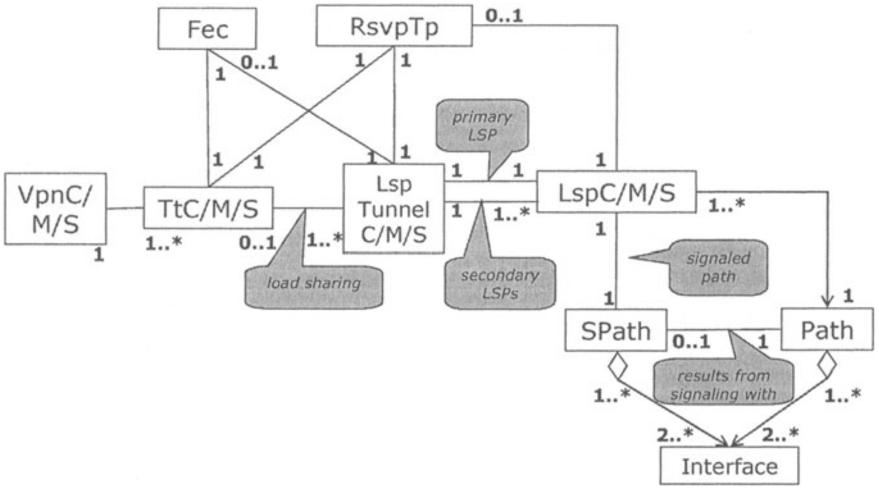


Figure 1. MPLS-TE Configuration Information Model

Classes with letter "M" denote that they are used for performance management. Traffic trunks, LSP tunnels and LSPs have their operational status information and statistics information such as packet per second every five minutes and bits per second every five minutes. Letter "S" stands for simulation. We identified some of the simulations which are very useful for the performance management. MPLS LSP path availability check, Node/Link failure, Traffic trunk and/or LSP tunnels attribute change and global optimization simulations are some of possible candidates. Simulations can be performed off-line with historical data acquired by off-line means. In such a situation, the simulation is typically considered to be a separate auxiliary mechanism to help performance management. In our proposed information model, we approached in a different way. These simulations are performed on-line with data collected and monitored live from the managed networks, which are stored in topology and performance sub-models. Since simulation results can modify the existing topological and performance information, separate place holders are needed. We modeled these "S" classes for that purpose.

The path availability check function provides an efficient way of simulating a setting up process of an LSP. The CSPF (Constrained Shortest Path First) algorithm, which is resident in our integrated management system, can compute the availability of a path of an LSP without actual enforcement. The server-based CSPF can also extend its scope to add additional constraints, e.g. actual usage instead of the required bandwidth of an LSP, besides what the online CSPF allows. This feature is one of the big advantages that an offline TE management system can provide. The LSP attribute modification simulation allows network managers to evaluate the side effects of LSP and VPN attribute modification. Modification of attributes ranges from simple single value change (e.g., the affinity value) to an entire path alteration for an LSP. This simulation helps the network managers create a detour route when a particular link is congested and see the link state changes in real-time. The link and node failure simulation depends on an online protection and recovery mechanism and visualizes its effects. Features like standby secondary paths, as well as explicit or dynamically configured primary and secondary paths of LSPs are also recognized for this simulation. Depending on the situation, the simulation can just visualize the overall status of a newly optimized network status or visualize all the paths computed by the server's CSPF algorithm. The global optimization is performed by a customized algorithm based on linear programming (LP). The algorithm can find near optimal paths that satisfy a given traffic demand under some constraints, such as bandwidth, a maximum hop count, and a preferred or avoided node or link list. Our integrated management system generates a mixed integer programming formulation for a given optimization problem and solves it with a dedicated LP solver. The optimization result contains each LSP's routing paths and the traffic split ratio, in case an LSP requires multiple paths. For easy representation at network nodes, the split ratio is chosen among discrete values (0.1, 0.2 etc.). The globally optimized set of paths can then be applied to the MPLS networks that permit explicit path setup. For more details, please refer to [5].

3. SUMMARY

In this paper, we proposed an information model for integrated configuration and performance management of MPLS-TE/VPN/QoS. Separate information models make difficult to manage tightly coupled multiple services. However, integration eliminates duplication of possible management functionality and enhances flexibility of managing such services.

The proposed information model is implemented and used in the integrated configuration and performance management system called Wise<TE> [6]. We have tested it over a network with 50 nodes and found out that it showed a good performance. The main concern is how the implementation can scale in terms of the number of CORBA objects as the number of nodes increase. We are planning to test our system over a network with upto several hundreds nodes. At the same time, we are in the process of testing it in one of major ISPs in Korea.

REFERENCES

[1] E. Rosen, A. Viswanathan, R. Callon, "Multiprotocol Label Switching Architecture ", RFC3031, IETF, Jan. 2001.

[2] Object Management Group, "Unified Modeling Language (UML), Version 1.4", formal/2001-09-67, September 2001.

[3] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss, "An Architecture for Differentiated Services", RFC2475, IETF, December 1998.

[4] D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, "Requirements for Traffic Engineering Over MPLS", RFC2702, IETF, September 1999.

[5] Y. Lee et al., "A Constrained Multipath Traffic Engineering Scheme for MPLS Networks," IEEE ICC 2002, New York, May 2002.

[6] TS Choi, SH Yoon, HS Chung, CH Kim, JS Park, BJ Lee, TS Jeong, "Wise<TE>: Traffic Engineering Server for a Large-Scale MPLS-based IP Network", NOMS 2002, April 2002.

USING THE ACCESS GRID AS A TESTBED FOR NETWORK MANAGEMENT RESEARCH

C.S. Hood, S. Devarapalli, N. Gadhia, S. Hegde, V. Mallikarjuna, S. Shankar, S. Yoginath

Department of Computer Science
Illinois Institute of Technology

Abstract: There are many open issues in the field of network management. System data and/or operational experience can provide insight into these issues. Researchers often do not have access to such data or practical experience. To overcome this, we propose using the Access Grid as a testbed for network management research. The Access Grid is a multimedia collaboration environment that utilizes many different network technologies. We are using the Access Grid to collect data, gain operational experience and evaluate new management techniques.

Key words: Network management, testbed, monitoring, multimedia, dataset

1. INTRODUCTION

Network management is a very practical, yet complex and theoretically interesting problem. Understanding the problem requires exposure to the issues faced in managing a complex system. We can get insight into the issues through data analysis and operational experience. Toward this end, we are proposing the Access Grid (AG) as a testbed for network management research. The AG is a large-scale distributed collaboration environment where audio, video, and data is exchanged over high-performance networks using multicasting [2]. It is a research environment that is actively used for collaboration across geographic distances. Over the past year, we have installed our own AG node at IIT.

Our AG node primarily serves as an environment for fault management research. Data for this type of research is not readily available, so to begin with we focus on generating a comprehensive dataset. The performance requirements of the audio and video streams provide service expectations that can be used to label the data. In addition to the data collection, we are creating simulation models to broaden the research capabilities. The goal is to put both the data and the simulation models in the public domain. This paper provides background about the access grid and describes our ongoing efforts in using the AG as a network management testbed.

2. ACCESS GRID

The Access Grid is a collection of AG nodes interconnected by high-speed networks. There are currently over 100 AG nodes around the world [8]. An AG node is comprised of the computational resources and equipment necessary to send and receive audio, video, and data streams. Each node typically sends multiple video streams and a single audio stream. The AG facilitates both formal and informal group communication. There are eight virtual meeting rooms that can be utilized by participants. These virtual meeting rooms accommodate events ranging from large-scale distributed meetings to smaller informal collaborations.

Each node has one or more operators. The operator installs and maintains the node hardware and software. During an event, the operator troubleshoots in real-time. One of the tools most frequently used by the community is the AG mud, a text-based chat system. During an event, as problems occur the operators communicate to resolve the problems via the mud. All mud sessions are archived providing significant insight into fault, performance and configuration management issues. The established community of AG node operators and technical experts is willing to discuss and assist on topics ranging from operational issues to research problems. The resources of this community can be drawn upon to answer questions, arrange multi-node experiments and test solutions.

The AG community welcomes new members interested in deploying nodes. Documentation covering necessary equipment, software and installation instructions can be found at [3]. The cost of equipment for an AG node is approximately $45,000.

3. MEASUREMENTS

The AG spans a variety of interesting networking technologies involving LANs, WANs, multicast and multimedia. It is a large-scale networked system that allows us to study problems in a realistic manner. The ability to collect significant amounts of meaningful data is key. The data collected may include statistical measurements, events, configuration information, problem descriptions, and any other information that will provide insight into the system or network state. Additional information on the network infrastructure can be found through the Quilt [10] and Abilene Network and Operations Center website [1].

We focus on audio and video applications. Application performance is used to identify periods of degraded service. There is a measurement infrastructure in place to collect network, system, and application level measurements across the grid. This infrastructure takes advantage of existing measurements and tools to create a unified measurement log. Links to many of the measurement efforts can be found at [7]. CAIDA has comprised a list of monitoring tools at [4]. Using these tools, different types of information can be collected in different ways.

Measurements can be collected using active or passive methods. Active methods add synthetic load to the system or network and observe the resulting performance. Passive methods collect measurements resulting from existing load. Our measurement infrastructure utilizes both of these methods. We consider the infrastructure to be a starting point for data analysis. The goal is to determine the utility of different types of information under different circumstances. We are collecting

- SNMP Management Information Base (MIB) information from hosts and routers. MIB data contains both static and dynamic information
- End-to-end measurements including Round Trip Time (RTT), packet loss, one-way delay, out of order packets and duplicate packets
- Routing histories and changes
- System information regarding processor, memory, disks, network and other devices
- Application (vic and rat) performance measures. Vic and rat are the video and audio streaming applications on the AG
- Multicast beacon measurements [6].

We have been logging many of these measurements for several months. We are in the final stages of enabling measurement collection from Abilene and regional gigapops. Presently, we are getting readings on most of the measurements every fifteen seconds. As the utility of different measurements is better understood, the monitoring frequency can be adjusted. Additionally, since the AG software is in the public domain, we can add instrumentation if necessary. The goal of this measurement infrastructure is to provide a data set for studying the management problem, not for real-time monitoring.

4. SIMULATION

To complement the "real" data, we also have a simulation of the AG. This allows us to simulate performance problems and test solutions. A simulation model of the AG can be an important tool for understanding how problems occur and propagate, as well as for testing the trigger models that are developed as part of this research. The simulation is particularly useful in terms of the network, because the opportunity for inserting faults and testing models is very limited there.

We simulate the AG [5] using the OPNET [9] network simulation tool. OPNET provides models for different types of equipment and protocols across many different technologies. The simulation project is broken into two pieces, (1) the AG node simulation, and (2) the network simulation.

The research described in [5] focuses on simulating the AG node. A typical AG node contains four machines; video capture machine, video display machine, audio machine and control machine. Our simulation models the first three machines. The primary function of the control computer is audio control. Since we have a good environment for studying the audio control problem directly, we have decided not to model the control machine. The three-machine model is sufficient to simulate the video and audio applications (vic and rat). Vic and rat use the Real-time Transport Protocol (RTP) to facilitate QoS. OPNET does not provide an RTP model, so this effort primarily involved implementing RTP. In our AG node model, we embedded RTP within the audio and video application models.

The network modeling piece is straightforward, largely involving gathering the information about specific network nodes, topology, and configuration. For Abilene and several regional networks, this information is readily available. We are in the final stages of modeling Abilene. The simulation will be validated using some of the data collected as part of the measurement infrastructure.

5. SUMMARY

The AG provides an opportunity to learn about management issues in a large-scale networked system. Implementing an AG node at IIT has been beneficial in many different ways. One key benefit is the collection of comprehensive, end-to-end labeled data. The goal of this project is to put the collected data into a public domain repository.

A second benefit is the operational experience. We have gained expertise installing the AG node and continue to face the challenges of operating the node. This first hand experience has helped us to identify several candidates for automation. Given the significant manual effort required to operate the nodes, automating pieces of configuration or troubleshooting can have big impact. The public domain node software enables us to deploy the mechanisms we develop. This is critical for evaluating the effectiveness of the mechanisms under realistic situations.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge support from NSF 013067 and NSF 9984811. This work was also supported in part by the U.S. Department of Energy, under Contract W-31-109-Eng-38.

REFERENCES

[1] http://www.abilene.iu.edu/
[2] http://www.accessgrid.org
[3] http://www.accessgrid.org/agdp
[4] http://www.caida.org/tools
[5] Chowdhury, Sadia, "Simulating the Access Grid," IIT MS Project, May 2002.
[6] http://dast.nlanr.net/Projects/Beacon/
[7] http://e2epi.internet2.edu/tools_list.shtml
[8] http://www-fp.mcs.anl.gov/fl/accessgrid/ag-nodes.htm
[9] http://www.opnet.com
[10] http://www.thequilt.net

AUTOMATING PLACEMENT OF INSTRUMENTATION IN APPLICATIONS

Seema Kaushal
Motorola

Hanan Lutfiyya
Department of Computer Science
The University of Western Ontario
London, Canada
hanan@csd.uwo.ca

Abstract:

In this paper, we present an architecture that provides the functionality to place customized and automated instrumentation. We determine the components that are needed for this purpose, services offered by each of these components and the algorithms showing the steps taken. Using this architecture, the time and effort needed to develop instrumentation toolsets can be reduced. Consequently, the time and effort needed to place instrumentation in distributed applications, to make them manageable, can be greatly reduced. We also describe the current state of the prototype, our conclusions and future directions.

Keywords: Instrumentation, Management, Distributed Applications, TXL

1. Introduction

Effective management of distributed applications requires the ability to monitor application-specific attributes e.g., the amount of time a specific remote procedure call took. This requires application instrumentation; that is, code inserted into the application at strategic locations so that the application process can maintain monitored information, respond to management requests and generate event reports based on the evaluation of a condition on the state of the monitored information.

The advantage of manual instrumentation (i.e., adding instrumentation by hand) is that applications can be instrumented to meet their specific needs i.e., the instrumentation is customized. A disadvantage of manual instrumentation is the extra effort, resources and time is required by developers. It is this additional developer time that is often cited as a criticizm of instrumentation [1].

With automation, instrumentation can be placed automatically This would, not only save time in the development process, but also minimize the potential for errors.

This paper discusses the components of a toolset needed to provide the functionality to place customized and automated instrumentation. The paper is organized as follows: Section 2 describes an instrumentation architecture. In Section 3, the requirements and a description of the components and the services offered by each of the components is described. Section 4 describes the prototype. Sections 5 and 6 describe related work and conclusions.

2. Instrumentation Architecture

The purpose of this section is to briefly discuss the instrumentation architecture described in [6].

An application attribute is associated with a sensor. Management requirements often place contraints on the values that an application attribute may take e.g., the time it takes to complete a remote procedure call should be one second. Sensors have variables representing information that includes threshold values and comparison operators that are used to compare monitored attribute values with the threshold values. The sensor's methods (*probes*) are used to initialize sensors with threshold values and collect values of attributes. The coordinator is the interface between the sensors and the management system. An actuator is similar to a sensor except it encapsulates functions that can exert control over the intrumented process to changes it behaviour.

As an example, let us assume that we have a communication statistics sensor, which is responsible for computing the communication statistics of a remote procedure call (RPC) to a server that we will call echo_server(...). This server is to be passed a string. This sensor, denoted by rpcSensor, may have the following methods: Process_rpcRequestBegin(...), which will record the RPC's start time and Process_rpcRequestEnd(...), which will record the RPC's end time. By inserting these probes before and after a RPC, the communication statistics sensor is able to compute the time taken by a particular RPC to complete and if a threshold is exceeded it passes a notification to the Distributed Application Management System through the management coordinator.

3. Toolset Architecture

Placing instrumentation code is the process of inserting probes at strategic points (Probe Points) within the application's source code. A probe point can be described using a *pattern*. A *pattern* is a string that describes source code that should be searched for in order to insert instrumentation code. An example of a pattern is fopen($1,$2). $1, $2 are placeholders representing strings for the file name and the mode used to open the file. Another example of a pattern is echo_server($1) where $1 is a placeholder for a string.

The overall approach to automatically place instrumentation code is to parse the source code to create a parse tree, search for patterns in the parse tree, and inserting the instrumentation code at the points where the pattern was found. This results in modifying the original parse tree. The leaves of the modified parse tree are, then, written back to the source code form.

Several repositories are needed to support this approach. Information about patterns is stored in a Pattern Repository. Part of the information associated with a pattern record is the type of source code construct that the pattern is associated with e.g., IPC, OS, middleware (e.g., DCE), etc. This functionality allows for the categorization of patterns (e.g., IPC, OS). An example of the usefulness of this is the following: The user (through the GUI) can specify classes of patterns that are to be used in determining probe points. This way the user does not have to specify each individual pattern; rather they specify the class that the pattern belongs to. The user is also able to add patterns to the repository.

The Pattern Recognizer traverses the source code's parse tree to find a pattern. For each match found, the source code strings are passed to the Probe Writer component, which is is used for writing the corresponding probes i.e., the actual probe strings. This processing needs the instrumentation repository that relates patterns and sensors/actuators as well as a string with variable placeholders (*probe patterns*) that represents a probe for each probe of the sensor/actuator. The Probe Writer analyses the source code to get all the variables that are needed to replace the variable placeholders in the probe pattern. The result is an instantiated probe pattern which is the actual probe to be inserted in the source code. This is put into the Probes Location repository. The following example illustrates this need: A sensor is used to count the number of times each file is accessed. The name of the file needs to be known. This name is passed to the sensor using a probe that is placed just before the fopen(...) call or fread(...) call etc; The general form of the probe code may be (...).CountFileAccesses($1) where $1 is a string that represents any file name. Thus, the actual probe code will differ for different fread(...) calls. After the tree has been traversed, the Probe Inserter component retrieves the probes from the Probe Location component to insert the probes in the parse tree of the source code at predetermined locations. The source code is then written back to the appropriate file from its corresponding transformed parse tree.

An editor is provided that allows the user to selectively choose points to add instrumentation. It provides the user with basic editing facilities such as *Open*, *Close* or *Save* a file and *Cut* or *Paste* text. It displays the list of sensors and actuators so that any desired sensor/actuator can be selected and its probes can be added at the desired location. This makes use of the Instrumentation Repository component.

The user interface allows the user to add patterns, sensors, associations between patterns and sensors, and update the status if a pattern (i.e., is a pattern to be searched for automatically).

4. Prototype and Initial Evaluation

As a proof of concept, we developed a prototype tool based on the architecture described in the previous section. The developed prototype can automatically instrument DCE and socket *C/C++* applications and also provides the flexibility of adding customized instrumentation. The User Interface is currently implemented as one process in *Java 1.1*. The repositories are implemented as a set of flat files. Access to the repositories is through UNIX shell scripts. The processing components were implemented through the use of UNIX shell scripts. Parsing and pattern recognition services are implemented using *TXL* [3]. The Probe Writer and the Probe Inserter make use of the *Sed* and *Awk* facilities in Unix.

The developed toolset was used by several members of our research group to instrument socket and DCE applications including an MPEG player consisting of 5000 lines of code. The tool was found to be fairly easy to use and intuitive.

5. Related Work

Tools examined that automate the process of placing instrumentation to a varying extent included Pablo [2], AIMS [8], Paradyn [5]. Unlike our work, none of these tools provide the developer with the easy flexibility in adding their own set of source code constructs for automated instrumentation. The work closest to ours can be found in [4]. There were several differences in our approaches including their inability to add patterns and thus change the probe points where instrumentation should be placed.

6. Conclusions

This paper presented a toolset that can automatically instrument programs as well as allow manual instrumentation. Possible directions for future work include the following: (i) Currently, only those patterns that are associated with function calls are recognized. This should be extended to more complex patterns. (ii) In some cases, not all files or communications need to be monitored. The toolset should provide the ability to specify a constraint on where the instrumentation should take place. (iii) The current toolset prototype was implemented to demonstrate that the toolset design concepts were sound. More work can be done on strengthening the prototype to make it more robust and expand it to other environments such as *CORBA*. (iv) The toolset prototype should be further evaluated by instrumenting larger distributed applications. This will help us see how the current implementation, of our prototype, scales up when instrumenting large applications.

Acknowledgements

We would like to thank Gary Molenkamp, Michael Katchabaw and other members of the Distributed Systems Research Lab at the University of Western Ontario. This work is supported by the National Sciences and Engineering Research Council (NSERC) of Canada and the IBM Centre of Advanced Studies in Toronto, Canada.

References

[1] U. Blumenthal, G. Kar, and A. Keller. Classification and computation of dependencies for distributed management. *IEEE Symposium on Computers and Communications (ISCC 2000)*, July 2000.

[2] Y. Chang. Pablo MPI Instrumentation User's Guide. *Technical report, Univ. of Ill*, 1999.

[3] James R. Cordy and Ian H. Carmichael. *The TXL Programming Language Syntax and Semantics*. Software Technology Laboratory, Department of Computing and Information Sciences, Queen's University at Kingston, Kingston, Canada, June 1993. Version 7.

[4] R. Hauck. Architectuer for an automated management instrumentation for component based applications. *Proceedings of the 12th International Workshop on Distributed Systems: Operations and Management DSOM'2001*, Nancy France, October 2001.

[5] J. Hollingsworth, B. Miller, M. Goncalves, O. Naim, Z. Xu, and L. Zheng. MDL: A Language and a Complier for Dynamic Program Instrumentation. *International Conference on Parallel Architectures and Compilation Techniques*, 1997.

[6] M. Katchabaw, S. Howard, H. Lutfiyya, A. Marshall, and M. Bauer. Making Distributed Applications Manageable through Instrumentation. *The Journal of Systems and Software*, 45:81–97, 1999.

[7] W. Rosenberry, D. Kenney, and G. Fisher. *Understanding DCE*. O'Reilly and Associates, Inc., 1993.

[8] J. Yan, S. Sarvkkai, and P. Mehra. Performance Measurement, Visualization and Modeling of Parallel and Distributed Programs using the AIMS Toolkit. *Software Practice and Experience*, 25(4):429–461, April 1995.

SESSION 3

Provisioning and Service Management

Chair: Marcus Brunner
NEC Europe Ltd., Germany

GENERIC ON-LINE DISCOVERY OF QUANTITATIVE MODELS FOR SERVICE LEVEL MANAGEMENT

Yixin Diao[1], Frank Eskesen[1], Steven Froehlich[1], Joseph L. Hellerstein[1] Alexander Keller[1], Lisa F. Spainhower[2], Maheswaran Surendra[1]

[1]*IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, USA*

{diao|eskesen|stevefro|hellers|alexk|suren} @us.ibm.com

[2]*IBM Server Group, 2455 South Rd., Poughkeepsie, NY 12601, USA*

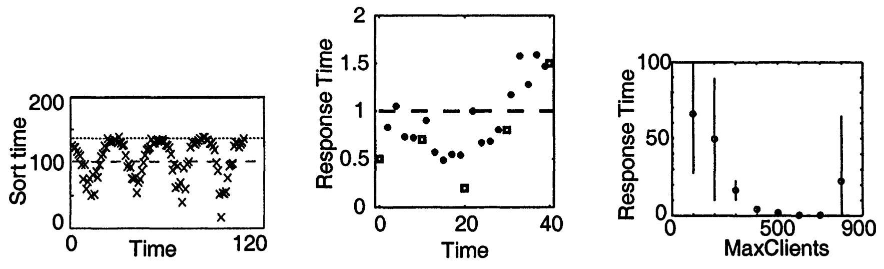
lisa@us.ibm.com

Abstract: Quantitative models are needed for a variety of management tasks, including (a) identification of critical variables to use for health monitoring, (b) anticipating service level violations by using predictive models, and (c) on-going optimization of configurations. Unfortunately, constructing quantitative models requires specialized skills that are in short supply. Even worse, rapid changes in provider configurations and the evolution of business demands mean that quantitative models must be updated on an on-going basis. This paper describes an architecture and algorithms for on-line discovery of quantitative models without prior knowledge of the managed elements. The architecture makes use of an element schema that describes managed elements using the common information model (CIM). Algorithms are presented for selecting a subset of the element metrics to use as explanatory variables in a quantitative model and for constructing the quantitative model itself. We further describe a prototype system based on this architecture that incorporates these algorithms. We apply the prototype to on-line estimation of response times for DB2 Universal Database under a TPC-W workload. Of the approximately 500 metrics available from the DB2 performance monitor, our system chooses 3 to construct a model that explains 72% of the variability of response time.

Keywords: Quantitative Model, Metric Discovery, Database System Instrumentation, Common Information Model, Service Level Management

1. Introduction

Central to service level management are tasks such as health monitoring to determine if the system is in a safe operating region, early detection of service level violations, and on-going optimization of configurations to ensure good performance. All of these tasks require quantitative insights, preferably quantitative models that predict service level metrics such as response time. Unfortunately, constructing such models requires specialized skills that are in short supply. Even worse, rapid changes in provider configurations and the evolution of business demands mean that quantitative models must be updated on an on-going basis. This paper proposes an approach to on-line discovery of quantitative models for service level management. The approach provides a way to construct quantitative models without prior knowledge of managed elements.



(a) Monitor health (b) Warn early (c) Optimize configuration

Figure 1. Some situations in which quantitative models is of value to service level management. (a) Identifying safe operating regions by using process control charts to track warning (horizontal line with long dashes) and critical (horizontal line with small dashes) limits on key variables, such as sort times in data bases. (b) Estimation of service level metrics such as response times (circles) provides early indications of service level violations compared to measured response times (squares). (c) Quantifying the impact of configuration parameters on response times provides a way to optimize configurations as illustrated by the parameter `MaxClients` for the Apache Web Server.

A variety of quantitative models are used in practice. For example, we relate DB2 performance metrics to response times using a model of the form $y = b_1x_1 + b_2x_2 + \cdots + b_nx_n$. Here, y is response time, the x_i are DB2 resource metrics (e.g., sort time, total buffer pool read time), and the b_i are constants estimated from the data using least-squares regression. We refer to y as the **response variable** and the x_i as the **explanatory variables**. Other examples of quantitative models include queueing network models (e.g., [12]), neural network models (e.g., [9]), and nearest neighbors approaches (e.g., [1]).

Figure 1 describes several situations in which quantitative models aid service level management. Considered first is monitoring system health by tracking key variables that characterize performance. For example, Figure 1(a) plots sort time, which turns out to be critical to the performance of eCommerce workloads such as those characterized by TPC-W [18]. A second situation is motivated by wanting an early warning of service level violations. This is illustrated in Figure 1(b) in which the squares are response times as measured by a response time probe. Such measurements are typically done at a low frequency because of cost and overheads. Augmenting these measurements with model-based estimates (the dots) provides a way to detect service level violations early (i.e., exceeding the dashed line). Last, optimizing configuration parameters requires having a quantitative understanding of how the parameters affect response times, which is greatly aided by having accurate quantitative models (especially for inter-related configuration parameters). Figure 1(c) illustrates this by showing that the Apache Web Server parameter `MaxClients` has a "U" shaped effect on response time, which permits using hill climbing to find a value of `MaxClients` that minimizes response time. Note that this method is applicable to any service level metric that is similarly affected by one or more configuration parameters.

While quantitative models can provide considerable value, their construction is difficult. Typically, a skilled analyst is required who understands the measurement data, configuration, and workloads. Changes in any of these or the relationships between them may mean that the model has to be reconstructed. This motivates a desire to automate model construction. Further, the approach should be *generic* in that it discovers the explanatory variables to use.

There are a number of areas of related work. Many researchers have investigated the detection of service degradations. Central to this is modeling normal behavior as in [13] which uses ad hoc models to estimate weekly patterns, [10] which employs more formal time series methods, and [19] which describes techniques for detecting changes in networks that are leading indicators of service interruptions. Further, statistical process control (SPC) charts are widely used for quality control in manufacturing [15] to detect shifts in a process as determined by an appropriate metric(s). These techniques have been applied to computing systems to track critical metrics (e.g., [14]). However, none of these approaches employ on-line model construction. More closely related to our work is [11], which uses knowledge of the functional relationship between inputs and outputs to detect changes in system operation. However, this work does not address how to identify a small set of explanatory variables. Several companies market products that aid in constructing performance policies (e.g., http://www.bmc.com). For the most part, the techniques employed are based on the distribution of individual variables, not relationships to response variables. One exception is correlation analysis (e.g., http://www.fortel.com), which uses cross correlation to identify the most important explanatory variables. However, this approach does not model the response variable. Thus, many redundant variables may be included. Further, all of the existing work assumes that the set of potential explanatory variables is known a priori rather than discovered on-line.

The remainder of this paper is organized as follows. Section 2 describes the architecture and information model used in our system for on-line metric discovery. Section 3 discusses the algorithms employed. Section 4 provides details of the prototype we built and illustrates its operation. Our conclusions are contained in Section 5.

2. System Overview

This section describes the architecture and information models used to support on-line discovery of quantitative models. Throughout, we assume that widely used supporting services are present, such as reliable message communication, persistence, registration, and location services.

2.1 Architecture

Figure 2 displays our architecture. The large rectangles identify key roles: the **Manager** and **Managed Element**. Although we treat all components in the architecture as objects, some can more naturally be thought of as data and others as procedures. The former are presented by ovals and the latter by small rectangles. Cascaded components (e.g., quantitative model) indicate that there may be several instances of them. The arrows indicate the flow of control or data, depending on whether the arrow connects two procedures or a procedure and data.

We begin by discussing the Managed Element. This is an encapsulation of a resource. A resource corresponds to functional entities such as a database, operating system, or web server. Our architecture augments the resource with an **Element Schema** that describes the resource (e.g., a database has tables and tablespaces), especially associated metrics (e.g., rows read, sort times) and configuration parameters (e.g., buffer pool sizes). In particular, we make use of the Common Information Model (CIM) [4] as a way to express the Element Schema. The **Agent Interface** (e.g., a

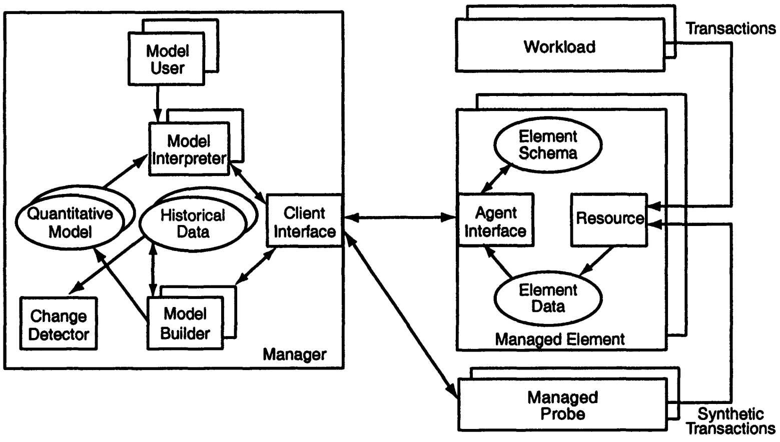


Figure 2. Architecture to support on-line discovery of quantitative models.

CIM Object Manager) is responsible for maintaining current values of element data as well as responding to requests to query the Element Schema. The use of an Element Schema is key to providing generic on-line discovery of quantitative models. Note that for our discussion, *schema* refers to the management information model describing the capabilities of a managed element, whereas the term *model* denotes the quantitative model used by the manager. Certain Managed Elements are used in special ways. A **Managed Probe** is a Managed Element that provides response time information by sending synthetic transactions to the resource and recording the transaction's start and completion times. Here, the Element Schema describes how to operate the probe (e.g., what synthetic transactions can be sent, the resource to which these transactions are sent) and the response times reported (e.g., by transaction type and resource).

Next we consider the Manager. The Manager communicates with Managed Elements through the **Client Interface**. The Manager has one or more **Model Users**, which are management applications that make use of quantitative models. Two examples of management applications are: (1) identifying key resource metrics for health monitoring and (2) predicting client response times for early detection of violations of service level agreements. For each type of model, there is a **Model Builder**. For example, we implemented a Model Builder that does real time construction of regression models [8]. The Model Builder typically requires historical data (e.g., to estimate the b_i in a regression model). These data are accumulated by using the Client Interface to find relevant data for the element (by querying the Element Schema) and then subscribing to updates of element data that are then placed in the Manager's historical data repository. Note that the same mechanism is used to acquire data from the managed elements and the managed probes. It may also be used to control workload generators if the Model Builder is conducting offline experiments to obtain a more diverse set of workload and system data. There are separate historical data for each model under construction. The **Model Interpreter** makes use of previously constructed quantitative models to estimate metrics of interest, such as estimating response times based on resource internal metrics. The **Change Detector** component uses statistical techniques (e.g., [2]) or rule based policies to determine if model predictions deviate too much from actual values and therefore the model must be revised (possibly by re-invoking the Model Builder).

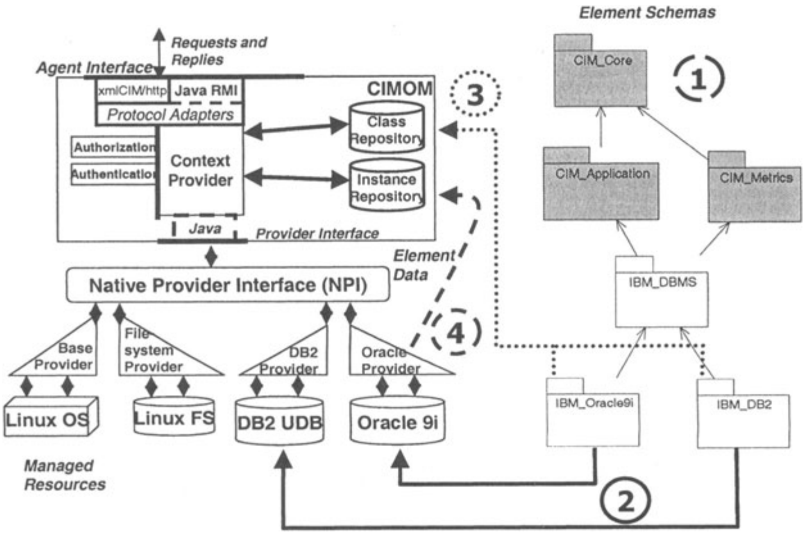


Figure 3. CIM Instrumentation of a Managed Element.

2.2 CIM-Based Managed Element

As mentioned earlier in Section 2.1, information about the Managed Element is surfaced by means of an agent based on the Web based Enterprise Management (WBEM) framework, whose core component is the Common Information Model (CIM) [4], a set of generic object oriented management models from which more specific resource models can be derived. The steps of designing and implementing a CIM based agent for retrieving metric data from the Managed Elements (in our case the database systems IBM DB2 UDB and Oracle9i) are depicted in Figure 3 and described below:

1 We identified the manageability information (e.g., descriptive and capability information, configuration parameters, statistical data such as counters and gauges etc.) available from our two target systems IBM DB2 UDB and Oracle9i. Then, we isolated the information common to both systems in an IBM_DBMS schema. The remaining data was placed into product-specific element schemas, which are named IBM_DB2 and IBM_Oracle9i, respectively (right side of figure).

2 CIM providers were implemented for the databases in accordance with the element schemas (solid lines at the bottom of the figure).

3 The standard CIM schema and the element schemas are loaded into the class repository of the CIM Object Manager (CIMOM). Then, the providers are registered with the CIMOM (as indicated by the dotted lines).

4 At runtime, clients request data from the Managed Element through the Agent Interface. These data are retrieved by the CIMOM, either from its instance repository or directly from the CIM providers (dashed line).

Designing CIM Extension Schemas for Database Systems. In accordance with CIM principles, every class of an extension schema is prefixed with a string identifying the creator of the management model (vs. the manufacturer of the resource); consequently, the classes we have introduced carry the prefix "IBM_". The hierarchy between the various CIM schemas and our extension schemas can be easily described by means of the UML package concept, which also captures dependency relationships (arrows between the various packages on the right side of Figure 3). It can be seen that we have used the CIM Core, Application and Metrics schemas as a basis to create our database models, i.e., we have derived the classes of our database models from classes defined in these schemas.

Designing the appropriate classes was fairly straightforward, given the fact that the basic architecture of a database system is common across multiple products: The overall `IBM_DBMS` (a subclass of `CIM_ApplicationSystem`) consists of a variety of subsystems, modeled either as subclasses of `CIM_SoftwareElement` or `CIM_SoftwareFeature`. In particular, we need classes to represent tables, tablespaces, buffer pools (aka caches), applications, and agents. The CIM schemas are designed according to the principle that all descriptive and capability-related information of a managed element is modeled as properties of the class representing the resource itself, while its statistical data is put in an associated class, subclassed from `CIM_StatisticalData`. This reflects the fact that the number of counters and gauges available for a managed element is in general fairly large, compared with the aforementioned descriptive resource information. Placing all statistical information into the resource class itself would lead to classes with many dozens of properties, which would result in an overly large amount of data that needs to be transferred if a class instance is to be retrieved. During our design, we were able to validate this assumption (some database parts have more than 30 different counters associated with them) and thus followed the recommended approach of encapsulating statistical information in separate classes. Overall, this leads to roughly a dozen leaf classes (plus a few association classes) that need to be implemented as CIM providers. In total, the classes of the IBM DB2 schema contain about 80 counters and gauges; for the sample database system we used for our experiments, this leads to about 500 metric instances that can be retrieved through the Agent Interface.

It should be noted that the DMTF Database Working Group [7] has recently published the first draft of a CIM Database Schema, which is likely to be included in the upcoming version 2.7 of the CIM Schemas. This schema contains a set of classes that describe a database in a very general way, which needs more refinement to be suitable for our purposes. More specifically, it defines three base classes `CIM_DatabaseSystem`, `CIM_CommonDatabase` and `CIM_DatabaseService` and three additional classes to capture statistical information related to the database as a whole and thus does not take the architectural details of a database system into account. Since our IBM_Database schema isolates the common elements of the two database systems with the largest market share, it is a possible candidate for submission to the DMTF Database WG for standardization.

Architecture of the CIM based Agent. The left part of Figure 3 depicts the architecture of our CIM agent. The part of the agent responsible for handling incoming requests and dispatching them to our providers is the publicly available SNIA (Storage Networking Industry Association) CIM Object Manager, which is implemented in Java. During our implementation, we encountered two important interoperability issues:

1 Currently, CIM does not specify the provider-facing part of a CIMOM; it is therefore not guaranteed that providers written for a specific CIMOM implementation are able to run with another CIMOM implementation.

2 Since the database systems expose their manageability information through interfaces in the C programming language, our providers are written in C, too. This brings up the need to interface between Java and C code.

The Native Provider Interface (NPI), implemented by the IBM Böblingen Laboratory as part of the SBLIM (**S**tandards **b**ased **L**inux **I**nstrumentation for **M**anageability) project [17], provides a convenient solution for both problems because it decouples the CIMOM from the various providers and the programming languages in which they are written. We use the NPI as glue code to interface between the Java based CIMOM and our database providers written in C. The detailed description of our prototype in Section 4 provides further details on our implementation experiences.

An object-oriented framework such as CIM permits the retrieval of all the statistical data in a single operation irrespective of the resource type (overall database system, tablespaces, tables, buffer pools, etc.). In particular, the CIM Operations over HTTP [5] protocol provides a means to enumerate all the instances of a given class. If the "deep" flag is set by a CIM client, this also applies to the instances of all the subclasses. Such a mechanism for retrieving only a selected subset of the available data is clearly superior to, e.g., a recursive "snmp-walk", which retrieves all of the data within a MIB (sub)tree. In our case, we can take advantage of this mechanism by retrieving all the instances of `CIM_StatisticalData` and its subclasses by means of a single operation. The CIMOM provides this level of abstraction by dispatching the requests to the appropriate providers, gathering the returned information and sending it back together, so that it is transparent to the client if the data surfaced by the CIMOM comes from one or many different CIM providers.

3. The Manager: Constructing Quantitative Models

This section describes the algorithms used by the Model Builder and Model Interpreter in Figure 2 for on-line construction and exploitation of quantitative models. To aid in this discussion, we use a running example in which DB2 UDB is the resource, **Element Data** are obtained from the DB2 performance monitor, and the response variable is response time (as measured by an external probe).

The **Quantitative Model** and algorithms employed by the Model Builder and Model Interpreter are specific to the modeling techniques employed. Thus far, we have found linear least-squares regression (e.g., [8]) to be very effective. The general form of a linear regression model is

$$y = b_0 + b_1x_1 + b_2x_2 + \cdots + b_nx_n.$$

The model relates the explanatory variables x_i to the response variable y through model parameters b_i. Linear models tend to be more robust than more elaborate nonlinear models such as neural network models. Also, linear least squares regression is widely applicable in solving real world problems. Moreover, building the linear models is less computational intensive so that it is more suitable for on-line model operation.

We begin by describing the content of the Quantitative Model object in Figure 2. For linear regression, it is sufficient to know the set of explanatory variables x_i and their associated constants b_i. For the running example, the challenge is determining a small number of metrics to use as explanatory variables out of the almost 500 metrics that can be obtained from the DB2 performance monitor.

It is the responsibility of the Model Builder to construct the Quantitative Model. Its inputs are `responseVariable`, the response variable (e.g., response time), and `candidateMetrics`, the set of candidate metrics. Its output is a Quantitative Model. Model Builder operates in two steps. The first is metric identification, which constructs `explanatoryVariables`, the set of explanatory variables x_i. The second step estimates the b_i.

3.1 Identifying Important Metrics

Metric identification determines which of the available metrics should be used as explanatory variables. Intuitively, it seems that using more explanatory variables results in a better model. However, this is often not the case. Indeed, extraneous variables may impair model performance when applied to new data.

Several approaches exist for selecting explanatory variables for quantitative models. Exhaustive search (e.g., false nearest neighbors [16]) examines all possible combinations of explanatory variables, but this is usually tractable only if the number of metrics is small. Ordered search explores the input possibilities according to certain importance measures, either in the incremented order or in the decremented order. One ordered search method is called stepwise regression. However, it may not necessarily produce the best model if there are redundant explanatory variables and may fail when applied to new data sets [8]. Moreover, it is computational intensive and may not be suitable for on-line discovery.

Metric identification takes as input `responseVariable` and `candidateMetrics` and produces `explanatoryVariables`, the set of explanatory variables. Metric identification begins once **Historical Data** have been collected for the candidate metrics. The details are described in Figure 4.

Note that the above algorithm does not consider all $2^m - 1$ possible models (where m is the number of metrics provided by the Managed Element). Rather, it incrementally selects the best metric based on what the current model does not explain. This simplifies the computation and makes it suitable for on-line discovery.

Also note that with more explanatory variables, the model tends to overfit the modeling data. To solve this problem, we use cross validation, a technique that employs testing data to assess model accuracy [20].

```
1 Initialization.
  (a) Set candidateMetrics to: list of metrics obtained from the Managed Element.
  (b) Set residualVariable to the response variable for the model.
  (c) Set explanatoryVariables to null.
2 Find the metric that best explains residualVariable.
  (a) Compute the cross correlation of each metric in candidateMetrics with residualVariable.
  (b) Set bestMetric to the metric with largest absolute value of the cross correlation.
  (c) Append bestMetric to explanatoryVariables.
3 Update variables.
  (a) Build a regression model of responseVariable on explanatoryVariables and set
      residualVariable to the residual of this model (the difference between the actual and estimated
      values of the responseVariable).
  (b) Remove bestMetric from candidateMetrics.
4 Check for termination.
  (a) Use cross validation to see if the testing error is increasing. If so
      i Remove bestMetric from explanatoryVariables.
      ii Return.
  (b) If candidateMetrics is not empty, then goto (2). Otherwise, return.
```

Figure 4. Algorithm for metric identification.

3.2 Estimating Parameters with a Quantitative Model

A wide range of standard techniques exist for estimating the regression model parameters b_i. In our case, the model is built initially using batch least squares. By "least squares", we mean that the unknown model parameters b_i are estimated by minimizing the sum of the squared deviations between the measured response data and the values estimated by the model. By "batch", we mean that all data are collected and *then* the b_i are estimated. The batch approach is well suited for off-line analysis, but can cause substantial computational overhead for on-line parameter estimation. As a result, we use recursive least squares [8], a technique that allows parameter estimates to be updated as new data are obtained. Not only does this reduce the computational overhead, it also provides a way to adapt to changes in workloads and configuration (although doing so requires another parameter, a forgetting factor, that determines the relative "weight" given to more recent data). To assess model accuracy, we employ the widely used R^2 metric.

$$R^2 = 1 - \frac{var(y - \hat{y})}{var(y)}$$

where y is the response variable, $\hat{y}$ is the estimated response variable, and $var(.)$ is the variance. The R^2 metric quantifies model accuracy by computing the variability explained by the model. R^2 ranges from 0 to 1. A value of 0 means the response data variability is not captured at all. A value of 1 may suggest a perfect fit.

4. Prototype Implementation

This section describes a proof-of-concept prototype constructed to demonstrate a generic approach to metric discovery. The prototype is depicted in Figure 5 and follows the architecture described in section 2.1. The Managed Element in the prototype is the IBM DB2 UDB Version 8.1 database management system running on a Linux

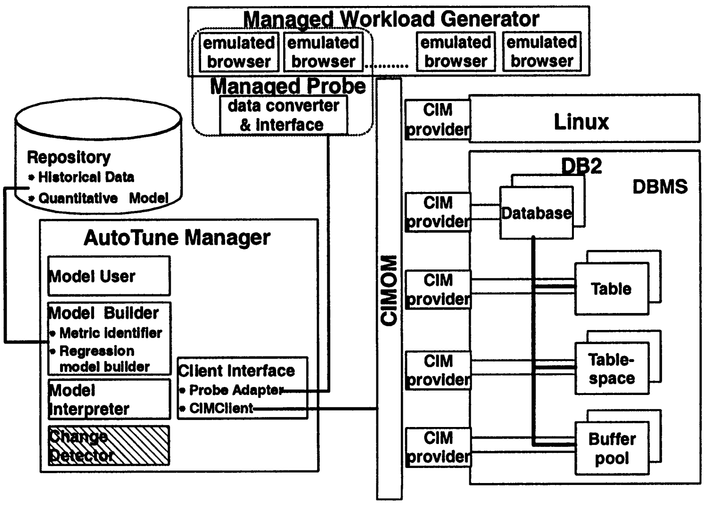


Figure 5. Architecture of the prototype

platform. As before, cascaded elements indicate multiple instances of an object (e.g. tablespaces, bufferpools).

4.1 CIM Providers for DB2 Performance Metrics

In order to dynamically obtain internal DB2 metrics, we developed CIM providers for DB2 Database, Bufferpool, Table and Tablespace information. Our providers use the `db2GetSnapshot` interface to obtain the data from DB2 and use the native provider interface to access the SNIA CIMOM.

This required some special code since the CIMOM environment requires that all functions operate in a thread-safe and thread-independent manner while the DB2 snapshot interface requires that all calls come from the same thread. The CIMOM can and often does create a new thread for each different request. Since each new CIMOM request would be driven from a separate thread and DB2 considers each thread a separate entity, without some sort of thread switching protocol within our provider, only meaningless (mostly zero) data could be extracted.

This led us to generate a separate thread for our DB2 application calls. All requests to DB2 are generated under control of this single separate thread. We atomically create this thread when we load the first provider and atomically delete it when we unload the last provider. When we receive a CIMOM request, we create a request element and atomically queue it to the DB2 thread. This request element contains a chain pointer, a synchronization mutex and a context descriptor. If, when queueing, the caller "atomically" recognizes that it is the first queuer, it also unlocks the DB2 accessor thread's mutex. After queueing the request, the caller waits for the DB2 thread to unlock the mutex contained in the queued request element. The DB2 thread unlocks the queue element mutex after the associated function completes.

The DB2 thread's logic is relatively simple. It has an input queue and a control mutex. The input queue is accessed atomically, using a locked compare and exchange instruction sequence. The first queuer, who changes the request queue from the empty state to the non-empty state, also unlocks the mutex after successfully changing the state. When the thread gains control, it atomically removes *all* elements from its input queue, thus changing the state back to empty. It then processes each removed element, one by one. After processing an element, it unlocks the mutex in the associated request element thus reactivating the calling thread. After processing all requests, the DB2 thread (again) waits for the mutex to be unlocked.

4.2 Workload Generation and Response Time Probes

The workload generator we use is TPC-W [18]. The characteristics of the workload are modulated by varying the number of emulated browsers (EB) and also the workload mix (buy vs browse). For the results shown here three types of workload mixes are used. The number of EBs is varied from 15 to 30 in a periodic fashion to approximately mimic time-of-day variations that are typical in an e-business environment.

For convenience, and also to take advantage of the measurement instrumentation available with the TPC-W kit, a subset of the EBs were also instrumented to provide client side response times (RT). Typically several transaction types are available from TPC-W, and for this work, the RT for the BestSellers transaction is used. On the same Linux system as the DB2 database, the workload was generated using emulated browsers each executing transactions according to the TPC-W benchmark specifications. A four hour workload cycle was created by continually increasing the number of emulated browsers from fifteen to thirty and then decreasing the number back down to fifteen. Additional variation was also introduced to the workload because of the probability associated with each emulated browser executing any one of the possible fourteen different transaction types. So analysis could be done later against the predicted response times, each active emulated browser logged the transaction type, the start time and completion time of that transaction.

4.3 Implementation of the Manager

The third major part of the prototype is the Manager. The Manager is built using the Agent Building and Learning Environment (ABLE) [3]. ABLE is a Java-based toolkit for developing and deploying hybrid intelligent agent applications. It provides a comprehensive library of intelligent reasoning and learning components packaged as JavaBeans (known as AbleBeans) and a lightweight Java agent framework to construct intelligent agents (known as AbleAgents). Built on top of ABLE is a general AutoTune Agent framework that facilitates the construction of on-line modeling/control agents. This general and extensible ABLE/AutoTune based implementation allows us to easily build the model, that is, the modules of Model User, Model Builder, and Model Interpreter. It also provides an interface to the Managed Element (for example, DB2) through an "adaptor" component that corresponds to the Client Interface in our architecture. Two adaptors are built: One communicates with Managed Element for DB2 and the other with the Managed Element for the response time probe. As an alternative to the latter, an adapter receiving SLA violation notifications from an SLA monitoring system (described in [6]) can be used to determine how a user experiences the quality of service.

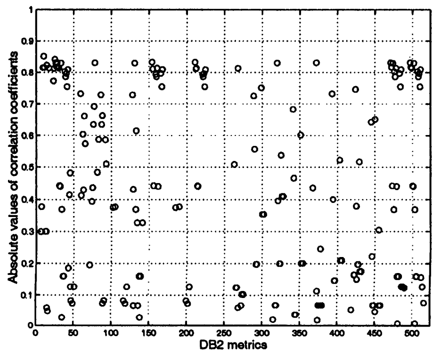
4.4 Interactions between Prototype Components

The operation of the prototype is as follows: First, the Model User instructs the Client Interface to obtain an enumeration of the available metrics from the CIMOM. In this implementation about 500 database metrics are made available. These metrics describe the operational status of the database across the multiple tables, tablespaces, and bufferpools. Although they provide a detailed view, most of them are either not directly related to predicting response time (e.g., `LockTimeouts` is a constant value which cannot be used to explain response time variations), or are essentially redundant. The metric identification procedure we described previously is able to discern the database metrics which are most effective as explanatory variables for response time.

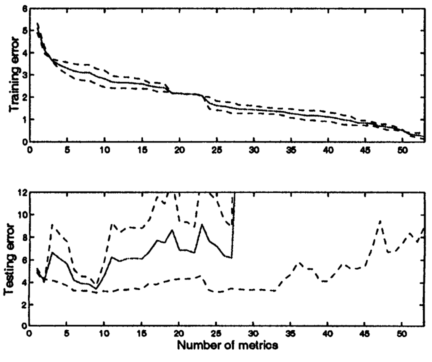
Similarly, data from the response time probe are collected. The data are sent to the Historical Data repository. In our prototype, both database metrics and response time are averaged over 30 minute intervals.

Next, the Model Builder is invoked to do metric identification. To illustrate the effectiveness of the algorithm in Figure 4, Figure 6(a) displays the absolute values of the correlation coefficients between the DB2 metrics (about 500) and the probed client response times. The high values indicate there exists strong correlation between the DB2 metrics and the client response time, which argues for building a linear model for response time prediction.

Figure 6(b) displays how the root mean square of the residual changes with the number of explanatory variables used. The top plot is a result of metric identification on all data in Historical Data. We see that having more variables almost always improves the model. The bottom plot uses cross validation, which drops observations in order to have separate testing data. The plot indicates that only the first three metrics should be used as explanatory variables. The resulting model is



(a) Correlation coefficients



(b) Determining key metrics

Figure 6. (a) Correlation coefficients between DB2 metrics and client response time. (b) Determine key metrics through cross validation.

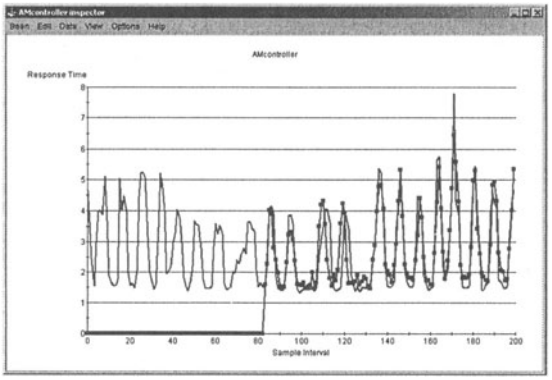


Figure 7. Screen shot showing extended time history of measured (light trace) and predicted (heavy trace) response time.

$$\begin{aligned} RT &= 1.44\texttt{ApplsExecutingInDBCurrently} \\ &+ 8.94 \times 10^{-5}\texttt{TotalBufferpoolReadTime} \\ &+ 9.69 \times 10^{-7}\texttt{TotalBufferpoolWriteTime} \end{aligned}$$

which has $R^2 = 0.72$ (i.e., explains 72% of the variability in the data). Since the workload variation is mainly caused by varying the number of emulated browsers, the ApplsExecutingInDBCurrently metric is identified as most important. The other two are also relevant because reading/writing to bufferpools is often where most of the delay in the database occurs when processing queries. The relative importance of these metrics, which are identified here without any in-depth knowledge of DB2, is consistent with the expectations of experienced database administrators.

Once the model is built, the Model User instructs the Model Interpreter to use the model to predict RT based on incoming DB metrics. The effectiveness of the model is apparent from Figure 7, where the light trace is RT as reported by the probe, and the heavy trace is the predicted RT. Note that in the left part of the plot, the heavy trace is flat since this is the period when the data are being collected for model building.

5. Conclusions and Outlook

Quantitative models have considerable value in performance management. Unfortunately, constructing quantitative models requires specialized skills that are in short supply. Even worse, rapid changes in provider configurations and the evolution of business demands mean that quantitative models must be continuously updated.

This paper proposes an approach to on-line discovery of quantitative models that operates without prior knowledge of managed elements. In particular, the Common Information Model (CIM) is used to discover metrics exposed by managed elements. These metrics are input to an algorithm that selects a subset to use as explanatory variables and then builds a quantitative model. Our approach employs an architecture in which the Managed Element includes components that describe the resource (Element Schema), contain data collected from the resource (Element Data), and an interface to the manager (Agent Interface). The manager has a matching Client Interface used by the Model Builder, which constructs quantitative models, and the Model Interpreter, which runs the models. The approach is demonstrated for estimating response times for DB2 UDB with a TPC-W workload. We show that of the approximately 500 metrics (counters and gauges) available from the DB2 performance monitor, our system chooses 3 to construct a model that provides very good estimates of response times.

While our initial results are encouraging, much work remains. Currently, the response variable (e.g., response time, throughput) must be known when the Model Builder is invoked. We are extending our architecture to include extracting response variables from a service level agreement specification. Another direction is to adapt the model on-line, such as when there are changes in workloads and/or configuration (which may require change-point detection). Last, we want to scale our techniques to address multi-tiered eCommerce systems.

References

[1] J. Aman, C. K. Eilert, D. Emmes, P. Yocom, and D. Dillenberger. Adaptive algorithms for managing a distributed data processing workload. *IBM Systems Journal*, 36(2), 1997.

[2] M. Basseville and I. Nikiforov. *Detection of Abrupt Changes: Theory and Applications*. Prentice Hall, 1993.

[3] J. P. Bigus, D. A. Schlosnagle, J. R. Pilgrim, W. N. Mills III, and Y. Diao. ABLE: A toolkit for building multiagent autonomic systems. *IBM Systems Journal*, 41(3), 2002.

[4] Common Information Model (CIM) Version 2.2. Specification, Distributed Management Task Force, June 1999. http://www.dmtf.org/standards/cim_spec_v22/.

[5] Specification for CIM Operations over HTTP, Version 1.0. Specification, Distributed Management Task Force, August 1999. http://www.dmtf.org/download/spec/xmls/CIM_HTTP_Mapping10.php.

[6] M. Debusmann and A. Keller. SLA-driven Management of Distributed Systems using the Common Information Model. In G.S. Goldszmidt and J. Schönwälder, editors, *Proceedings of the 8th IFIP/IEEE International Symposium on Integrated Network Management*. Kluwer Academic Publishers, March 2003.

[7] *DMTF Database Working Group*. http://www.dmtf.org/about/working/database.php.

[8] Frank E. Harrell. *Regression Modeling Strategies: With Applications to Linear Models, Logistic Regression, and Survival Analysis (Springer Series in Statistics)*. Springer Verlag, 2001.

[9] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Macmilan College Publishing Company, 1994.

[10] P. Hoogenboom and J. Lepreau. Computer system performance problem detection using time series models. In *Proceedings of the Summer USENIX Conference*, 1993.

[11] R. Isermann and B. Freyermuth. Process fault diagnosis based on process model knowledge. In *Proceedings of 1989 ASME International Computers In Engineering Conference and Exposition*, July 1989.

[12] Leonard Kleinrock. *Queueing Systems Volume I*. Wiley-Interscience, 2nd edition, 1975.

[13] Roy A. Maxion. Anomaly detection for diagnosis. In *Proceedings of the 20th International Annual Symposium on Fault Tolerance (FTCS)*, June 1990.

[14] J. McConnell, D. Helsper, L. Lewis, and S. Joyce. Predictive analysis: How many problems can we avoid? In *Networld+Interop, Las Vegas*, 2002.

[15] D.C. Montgomery. *Introduction to Statistical Quality Control*. Wiley, 3rd edition, 1997.

[16] Carl Rhodes and Manfred Morari. Determining the model order of nonlinear input/output systems. *AIChE Journal*, pages 151–163, 1998.

[17] *Standards Based Linux Instrumentation for Manageability Project*. http://oss.software.ibm.com/developerworks/projects/sblim/.

[18] Wayne D Smith. TPC-W: Benchmarking an ecommerce solution. In *http://www.tpc.org/tpcw*.

[19] Marina Thottan and Chuanyi Ji. Adapative thresholding for proactive network problem detection. In *IEEE Third International Workshop on Systems Management*, April 1998.

[20] S. Wold. Cross-validatory estimation of the number of components in factor and principal components model. *Technometrics*, 20(4):397–405, 1978.

A GENERIC MODEL FOR IT SERVICES AND SERVICE MANAGEMENT

Gabi Dreo Rodosek
Leibniz Supercomputing Center
Barer Str. 21, 80333 Munich, Germany
dreo@lrz.de

Abstract: Whereas network and system components were in the focus of management research in previous years, nowadays *management of services* dominates management activities. We are witnessing a paradigm shift from *device-oriented* to *service-oriented* management, and with this the need to deal with new challenging management issues. The management of the underlying infrastructure with respect to the delivered services and agreed service level agreements is certainly the fundamental challenge. It is easy to see that all research questions center around the new managed object *service* and its integration with existing device-oriented managed objects (network devices, end systems, applications). Thus, the development of a common definition of a service in terms of a *common generic service model* is essential.

Keywords: Service models, Information Modeling, IT Service Management

1. Introduction

Due to the significant increase in the complexity of enterprise applications and the need to offer distributed *IT services,* we are witnessing that the management focus has turned away from device-oriented to service-oriented management. This does not mean that device-oriented management is not of importance any more. On the contrary, an efficient device-oriented management is a precondition for an efficient IT service management. However, device-oriented management was exclusively in the domain of the provider and it was driven by the objectives of the provider. Nowadays, it is necessary to manage the underlying infrastructure with respect to services offered to customers and agreed service levels agreements. The complexity of IT service management becomes evident with the necessity to cope with service dependencies and the distributed provision of a service upon several resources.

The fundamental issue of IT service management is the development of a common definition of a service in terms of a *common service model.* Despite of an amount of existing service definitions (e.g., 9, 3, 5, 13, 6) a common understanding of a service that provides a unified approach to support the concepts of service management is lacking. A first step towards a *generic service model* has been proposed by the Service Management Task Force (SMTF), a group of researchers of the Munich Network Management team, in 4. We take this service model as a basis for the development of the *common generic service model* that provides the information basis upon which the deployment of service management applications can be approached.

The paper proceeds as follows: Section 2 gives an overview of the methodology for service modeling. The service-centric aspect is addressed with the service template model in section 3. Section 4 addresses the provider-centric aspect by introducing the provider-centric service template model, and section 5 proposes the customer-centric service template model. An example of the applicability of the proposed models is given in section 6. Finally, section 7 concludes the paper.

2. Methodology for Service Modeling

The business reference model, which provides the basis of our discussion, is visualized in Fig. 1, and identifies three aspects that need to be addressed: (i) the service-centric, (ii) provider-centric and (iii) customer-centric aspect of service modeling. The role of a customer refers hereby to an organization whereas the role of a user refers to a individual user. The further discussion refers only to the customer role.

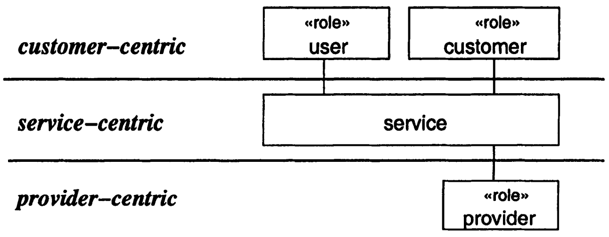


Figure 1. Business reference model (UML notation)

In the following, the identified aspects are addressed in more detail:

Service-centric part. This part refers to the elements of a service which are independent of any provider- or customer-centric issues. The main elements of the service-centric part are as follows. Firstly, the specification of the "abstract" *service functionality* needs to be approached. The service functionality with respect to the service hierarchy consists of two elements: (i) its own functionality and (ii) the functionality of its sub-services. Secondly, the specification of the "abstract" *quality* of the provided service functionality needs to be addressed. Quality of the provided functionality is measured and expressed with *service-centric QoS parameters.*

Provider-centric part. This part addresses the point that services, respectively the service functionality, can be provided in different ways by different providers (e.g., specific policies, specific service provisioning). Primarily, this part addresses the aspect of service provisioning and service operation. Elements of the provider-centric part are: (i) *steps* how to provide, operate, and withdraw services, (ii) the *quality* of services as offered by providers and (iii) the *policies* how to operate services. The workflow aspect is associated with the steps because several persons need to work together on service planning, provisioning, operation, change, and withdrawal. Quality of the provided services is measured with *provider-centric QoS parameters.* These parameters refine service-centric QoS parameters with value ranges that are defined by providers. In general, the specification of values for provider-centric QoS parameters is a trade-off between quality, cost and market demands.

Customer-centric part. A service as provided by a service provider can be offered to various customers. One of the most important elements of a customer-centric part are *customer-centric QoS parameters.* A customer has the possibility to select specific values from the offered provider-centric QoS parameters, respectively the value ranges.

We will proceed with a detailed specification of the attributes for the three identified parts of a service. A systematical identification of the attributes and their refinement requires to follow a *methodology for service modeling.*

The methodology for service modeling specifies the steps necessary for the development of the service model to address the service-, provider- and customer-centric aspects of a service. As depicted in Fig. 2, these steps are as follows:

1 The specification of the **service-centric** part of a service which is addressed by the development of the **service template model**. Often, the abbreviation **STM (service template model)** will be used.

2 The specification of the **provider-centric** part which is addressed by the development of the **provider-centric service template model**. Often, the abbreviation **provider-centric STM** will be used.

3 The specification of the **customer-centric** part of a service which is addressed by the development of the **customer-centric service template model**. Often, the abbreviation **customer-centric STM** will be used.

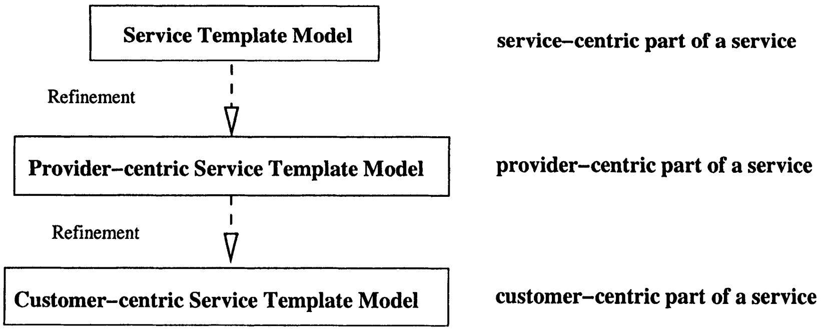

Service Template Model
service–centric part of a service
Refinement
Provider–centric Service Template Model
provider–centric part of a service
Refinement
Customer–centric Service Template Model
customer–centric part of a service

Figure 2. Methodology for service modeling

The order of the steps is defined by the fact that a service is an "abstract" functionality which needs to be realized (i.e., provided) by a provider. Different providers may provide the same services differently because of for example environmental specifics, policies, type of customers. Different customers may subscribe to the offered services. This is visualized in Fig. 2 with the introduction of the three models. The relation between the service models is defined such that (i) the *service template model* is refined to a *provider-centric service template model*, and (ii) that the *provider-centric service template model* is further refined to a *customer-centric service template model.*

Covering various aspects of a service (service-, provider- and customer-centric) with various models gives us flexibility and addresses with this the requirement of providing the same service by different providers to different customers.

To clarify the meaning of the introduced service models, we extend the model layer with the instance layer: a service template model is instantiated to a *service template instance,* a provider-centric service template model to a *provider-centric service*

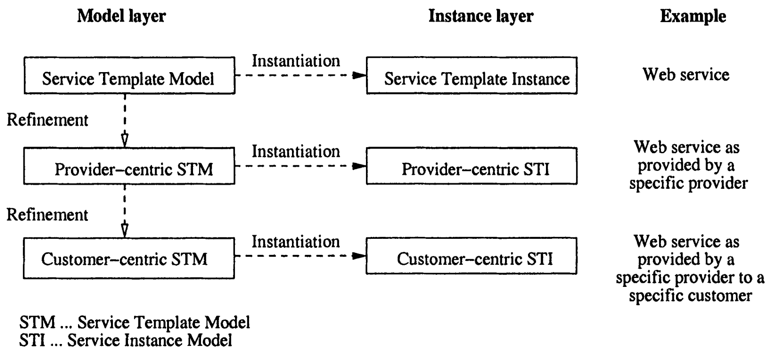


Figure 3. Model and instance layer explained on the Web service example

template instance, and the customer-centric service template model to the *customer-centric service template instance*. This is visualized in Fig. 3, where horizontally the instantiation of models to instances is visualized, and vertically the refinement of models. Let us explain the meaning of the model and instance layer on an example. An example of the service template instance is the Web service. An example of a provider-centric service template instance is if the "Web service is provided by a specific provider (e.g., Leibniz Supercomputing Center)". Furthermore, the "Web service customer-centric STI" layer represents in such a case a Web service as provided by a specific provider (e.g., Leibniz Supercomputing Center) to a specific customer (e.g., University of Munich).

As identified by the methodology, the first model to deploy is the *service template model.*

3. Service Template Model

The **service template model** addresses the elements of the service-centric part of a service (i.e., the service part which is independent of any customer- or provider-centric issues). To identify this service-centric part, the starting point of our discussion is the definition of a **service**.

We define a *service as a* **functionality** *that is provided with a certain* **quality** *and cost at a Service Access Point* (**SAP**). Elements of the service-centric part are discussed in the following.

Functionality of a Service. The functionality of a service consists of two parts: (i) the functionality of the service itself, and (ii) the functionality of sub-services that are involved in the service provisioning with respect to the service hierarchy. As a consequence, *service dependencies* are another element of the service-centric part of a service which are addressed later on. A quite common approach to describe the service functionality is with service functional building blocks. A precise specification of the service functionality, respectively the service functional building blocks, requires to include *functional parameters* into the service template model as well. Functional parameters are associated with service functional building blocks. For example, in order to send an email, it is necessary at least to specify the email address of the recipient (i.e., *send_email(EmailAddress)*). Functional parameters can be in general of all types as used in programming languages. An example is a character string or a sequence (e.g., Email address).

Service Access Point. In the customer-provider scenario a SAP is the point at which service functionality is accessed by a customer and provided by a service provider. The SAP definition of the OSI reference model cannot be used directly for our purposes, because such a strict layering of services is not existent in a service hierarchy. Services on the same layer can use other services on the same layer. Therefore, we define a SAP as *a point, where a service requests the functionality of another service or vice-versa provides its functionality to another service (regardless of any layering of services).* From the implementational point of view, a SAP can vary from a simple router interface to a complex application call. A router interface is an example of a SAP that is used by an ISP (Internet Service Provider) whereas an application call is an example for a SAP that is used typically by an ASP (Application Service Provider). To distinguish between these kinds of SAPs, we introduce the term *type* of a SAP. Another issue is also the necessity to identify the *location*, respectively the resource that realize the SAP.

Quality of Service (QoS) Parameters. These parameters are used to measure the quality of the provided service functionality at the SAP. Service functionality can be measured with one or more QoS parameters, and vice-versa a QoS parameter may measure the quality of various services (e.g., availability). There exists a many-to-many relationship between a QoS parameter and a service.

QoS parameters can be classified in general into (i) qualitative and (ii) quantitative parameters. Qualitative parameters express the quality in ranges such as gold, silver and bronze or yes and no, aggregating several quantitative or qualitative parameters under each term. Quantitative parameters measure the quality of parameters in concrete quantities and values (e.g., availability of 99,98%). A precise specification of QoS parameters is one of the most important issues of quality management and includes several elements, as depicted in Fig. 4. First, it is necessary to identify relevant QoS parameters for a service and specify the semantics of these parameters. Additionally, it is necessary to specify also the value type and the possible value ranges of the parameters. The value type specifies that a parameter is measured for example in percentage and that the value range may be from 0% till 100%. In the case a service depends on other sub-services, a QoS parameter (e.g., QoS3 in Fig. 4) is an aggregated parameter of the basic parameters QoS1 (i.e., a QoS parameter of sub-service i) and QoS2 (i.e., a QoS parameter of sub-service j). To distinguish between basic and aggregated QoS parameters, we introduce the term *parameter type.* Basic QoS parameters are aggregations of QoD parameters (i.e., MIB variables).

The description of QoS parameters includes the following elements (as visualized in Fig. 4): (i) specification of the semantics of a QoS parameter, (ii) specification of the value type and value range, (iii) specification of the parameter type (basic or aggregated), (iv) definition of the calculation metrics (i.e., how QoS parameters are aggregated or deduced from other QoS parameters or from QoD parameters), and (v) identification of Quality of Device parameters (QoD). The identified elements are analyzed in the following in more details:

Semantics of QoS parameters. An explicit and precise description of the semantics of QoS parameters from the customer's and provider's perspective is essential to omit possible misunderstandings. In most cases, such a description is per-

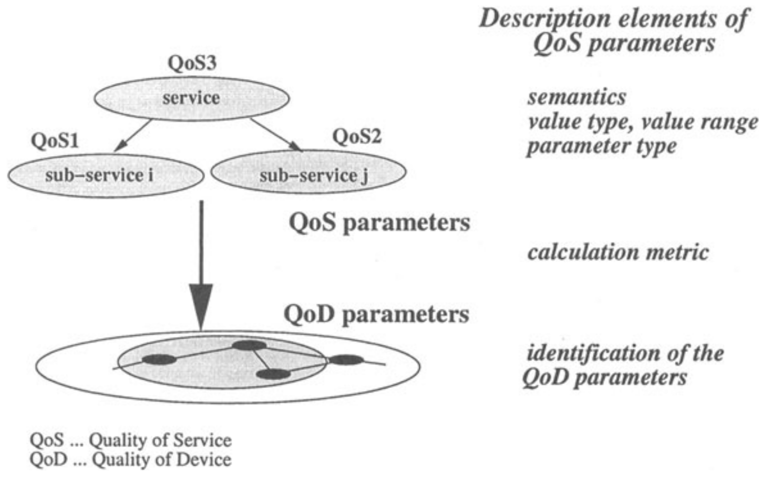


Figure 4. Description elements of QoS parameters

formed in a free-form text although a more formal description would be more appropriate.

An alternative approach to describe the semantics in more details is to express the service quality with several QoS parameters. An example should clarify this statement. Assume that a customer and a provider agree on the QoS parameter *availability* with a specific value. This value can be met if a service is unavailable once for a long time period or the service is unavailable very often for short time intervals. In the first case, such a long unavailability of a service may cause more serious problems than the short outages. Thus, it would be appropriate to specify additionally to the *availability* also that the parameter *MTTR* (Mean-Time-To-Repair) should not exceed a certain time interval.

Value type and value range. A further point of discussion is the description of the value type and with this associated the value range of a QoS parameter. Similar, as with the definition of the types of functional parameters, types of QoS parameters can be those used in programming languages. In some cases, the value range can be derived from the value type. For example, a common value type for the QoS parameter *availability* is to express this parameter as a percentage value. The derived value range is from [0% - 100%]. In some cases, however, such an automatic derivation is not possible, and therefore it is necessary to separate between the value type and the value ranges of QoS parameters. For example, the value range for the QoS parameter *Customer Satisfaction Index (CSI)*, measuring the quality of the hotline support from the customer's perspective, can be specified in various ranges.

Parameter type. QoS parameters can be of the following types with respect to the previous discussion: (i) basic or (ii) aggregated. Basic QoS parameters are calculated directly from QoD parameters (i.e., MIB variables), whereas aggregated QoS parameters are an aggregation of several basic QoS parameters.

Calculation metric. The calculation metric specifies how QoD parameters are combined, respectively aggregated, together to obtain the requested QoS parameters. Calculation metrics are used to aggregate (i) QoD parameters $\rightarrow$ basic QoS parameters and (ii) basic QoS parameters $\rightarrow$ aggregated QoS parameters. A necessity hereby is the specification of a *calculation language* in order to describe the calculation metrics on the service layer appropriately, as proposed in 2.

Identification of QoD parameters. QoD parameters are relevant MIB variables or other data (e.g., log files) which can be gathered by device-oriented management tools. If following a top-down approach, the identification of the relevant QoD parameters, in case they exist, is certainly a challenging issue and depends on the application scenario.

Service Dependencies. As known services depend on sub-services in terms of a service hierarchy. The goal of this discussion is to address the description of the *depends on* relation between services and sub-services. It should be noted that we refer to dependencies which are visible from the outside (i.e., so-called inter-dependencies), and not to intra-dependencies within a service (e.g., dependencies between software components of a service). Service dependencies can be presented in terms of a directed acyclic layered graph. We refer to this graph as a service dependency graph. The nodes of the graph represent the services and sub-services whereas the directed edges represent the *depends on* relation between services and sub-services. It should be stressed that the dependency relation does not only exist between services on the higher layer and sub-services on the lower layer, but also between services on the same layer. For example, the proxy and the database services depend on the DNS (name service). Weights or other attributes may be assigned to edges of the service dependency graph if the graph is used for certain purposes such as root cause analysis, configuration issues, calculation of QoS parameters or service provisioning. For our purposes it is enough to model the dependency relation as a directed relation between a service and its sub-services and having the ability to assign attributes to the edges of the service dependency graph.

Service Cost. The service template model needs to address, additionally to the service quality, also the cost of the provided services. The service cost is measured with cost parameters, and is in relation with the provided service functionality and the quality. Similarly as with QoS parameters, we distinguish between (i) the cost for a whole service and (ii) the cost of separate service functional building blocks. Furthermore, it is necessary to distinguish between *one-time* cost for the service provisioning and the *running* costs. In order to measure the running costs, it is necessary to identify (i) accountable units of a service and the (ii) cost for a unit. The following tasks need to be performed: (i) the identification of service-related accountable units, (ii) specification of the measurement methodology and reference point, (iii) specification of tariff models to determine the service cost. The tariff model is, however, not part of the service model but part of the SLA.

The summarized view of the service template model is shown in Fig. 5.

4. Provider-Centric Service Template Model

The next step in the methodology of service modeling is the development of the **provider-centric service template model**. The provider-centric STM addresses the

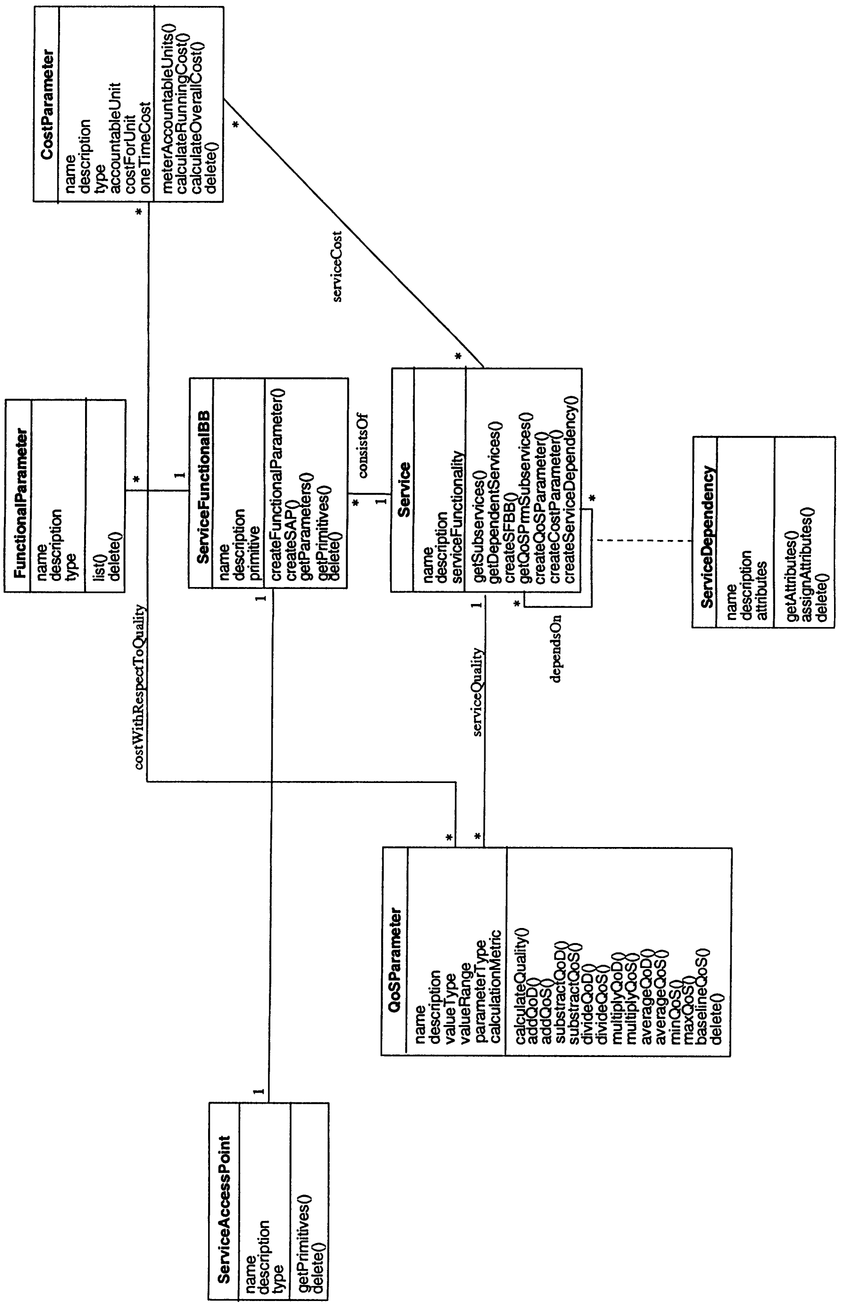


Figure 5. Design of the service template model

provider-centric issues which refer primarily to service provisioning and operation. The goal of this discussion is to identify the elements of the provider-centric part of a service.

Based on the provider's primarily objectives of service provision and operation, the following elements need to be addressed additionally in the provider-centric service template model: (i) specification of the steps for service provisioning, operation, change and withdrawal of a service, (ii) specification of the quality of the provided services, (iii) specification of policies to define the operation of services,(iv) specification of a *Customer Service Management (CSM)* to give customers a transparent view of the quality of the provided services. These elements are analyzed in more details in the following discussion.

Steps. Each phase of the service life cycle consists of several steps in order to realize the goal of the phase. For example, planning steps are necessary to plan the provision of a service. This includes mainly the identification of the resources to provide the service and the development of an operational concept for this service. However, planning steps are not part of the provider-centric service template model.

Provisioning steps are necessary for the configuration of resources involved in the service provision. A step or an action is thus associated with a resource, and it can be a simple execution of a script (e.g., `traceroute www.lrz.de`) or a complex process (e.g., *configure_database(database_server)*). *Operational steps* refer to steps necessary for the configuration of device-oriented management tools to monitor and control resources which are involved in the service provision. The goal is to gain a *service view* based on the device-oriented information as provided by management tools. One of the challenges hereby is the identification of the involved resources and the appropriate configuration of the device-oriented management tools. Another issue is the dynamic changing environment. For example, it is necessary to cope with services that are "moved" around the infrastructure, with changes in the functionality of management tools or new services that are introduced. Current management tools almost give no support to cope with the mentioned change dynamics. There is a lot of manual work associated with the tracking of changes. *Withdrawal steps* refer to the reconfiguration of resources involved in service provisioning as well as the reconfiguration of management tools involved in service monitoring and control.

Problems associated with the specification of steps are as follows: (i) it is necessary to specify the steps with an adequate granularity, and (ii) to assure the up-to-dateness of the steps. To cope with the high change dynamics, an alternative approach to maintain the steps is as follows: firstly, to specify the steps in a generic, parameterized way, and secondly, to improve the granularity and up-to-dateness of the steps with real experience during the application of the steps. Obviously, an ideal solution would be to have steps defined as executable scripts which could be executed automatically by existing tools. Unfortunately, such granularity is difficult to achieve and also to maintain due to the enormous effort. Such a detailed specification has already proved their weakness in the fault management area.

Provider-centric QoS parameters. The value range of service-centric QoS parameters is changed in the provider-centric service template model with respect to provider specifics.

SLAs. Another element of the provider-centric service template model which needs to be addressed are Service Level Agreements (SLAs), respectively the technical part of an SLA. A provider needs to think of SLAs that he can offer to customers.

Policies. A provider has certain provider-centric constraints that have an influence on the operation of services. Operational rules of service operation are specified as *policies.* Examples of policies are to make a service available 24 hours a day, 7 days a week or to deny everything what is not explicitely allowed. An adequate description of policies requires the development of an appropriate specification language, as proposed for example by Sloman et al. in 1. Besides, topics such as policy conflict resolution (10) need to be addressed as well. Our focus lies solely on the application of existing work done in policies to our problem area.

Customer Service Management (CSM). A requirement from the customer's side is to have a customer-centric and transparent view on the provided quality of the subscribed services. A provider needs to offer a set of reports about service quality that a customer can configure. Beside accessing reports about the provided service quality, customers want to interact with the provider for example to report problems, order new services, select new service qualities and service costs, track the resolution of problems. In other words, customers want to have access to a CSM. The objective of a provider is to develop a CSM application with the functionality to cover all phases of a service life cycle where a customer is involved in.

Customer Service Management Access Point (CSMAP). A CSMAP is the interface between a customer and a provider over which a customer can access the CSM application and the requested information. From the conceptual point of view, the meaning of a CSMAP is similar to a SAP.

Beside the description of the provider-centric service template, another important provider issue needs to be mentioned as well. A provider needs in some cases a more "global" view of the provided services to customers due to operational issues (e.g., availability of the whole IP backbone versus the availability of the individual router interfaces). We have referred to this view as the "all customers" view, and to the services as management services. Such a global view is important to monitor or achieve a good utilization of resources or to act appropriately in case of failures. Again, assume that a provider offers connectivity services to his customers, and that a SAP to a customer is a router interface. From the customer's perspective, only the availability of his SAP (e.g., the router interface) as well as the provided quality of service at the SAP is of importance. It is irrelevant for him whether other router interfaces or even routers are unavailable so far his quality of service is not affected. However, from the provider's perspective, the availability of the whole IP backbone is of relevance as well to act appropriately in the case of performance degradations or failures. Therefore, an example of such a management service is *IP backbone* and its availability.

As already emphasized, management services are treated in the same way as application-oriented services with respect to monitoring QoS parameters and costs. The only difference is that internal services have, in general, no penalties, in the case of violating the thresholds, associated with them (in most cases), and that in general the reporting and escalation mechanisms may be different.

A service provider uses the provider-centric service template model, or precisely the provider-centric service template instances, to build a **library of services** (e.g., Web, Mail, UHD, backup, remote printing) that he provides to his customers with the specified provisioning and operational steps. Several services can be combined together to *service packages* (e.g., Internet access including name service, access service, connectivity service etc.). A customer may now select between these packages

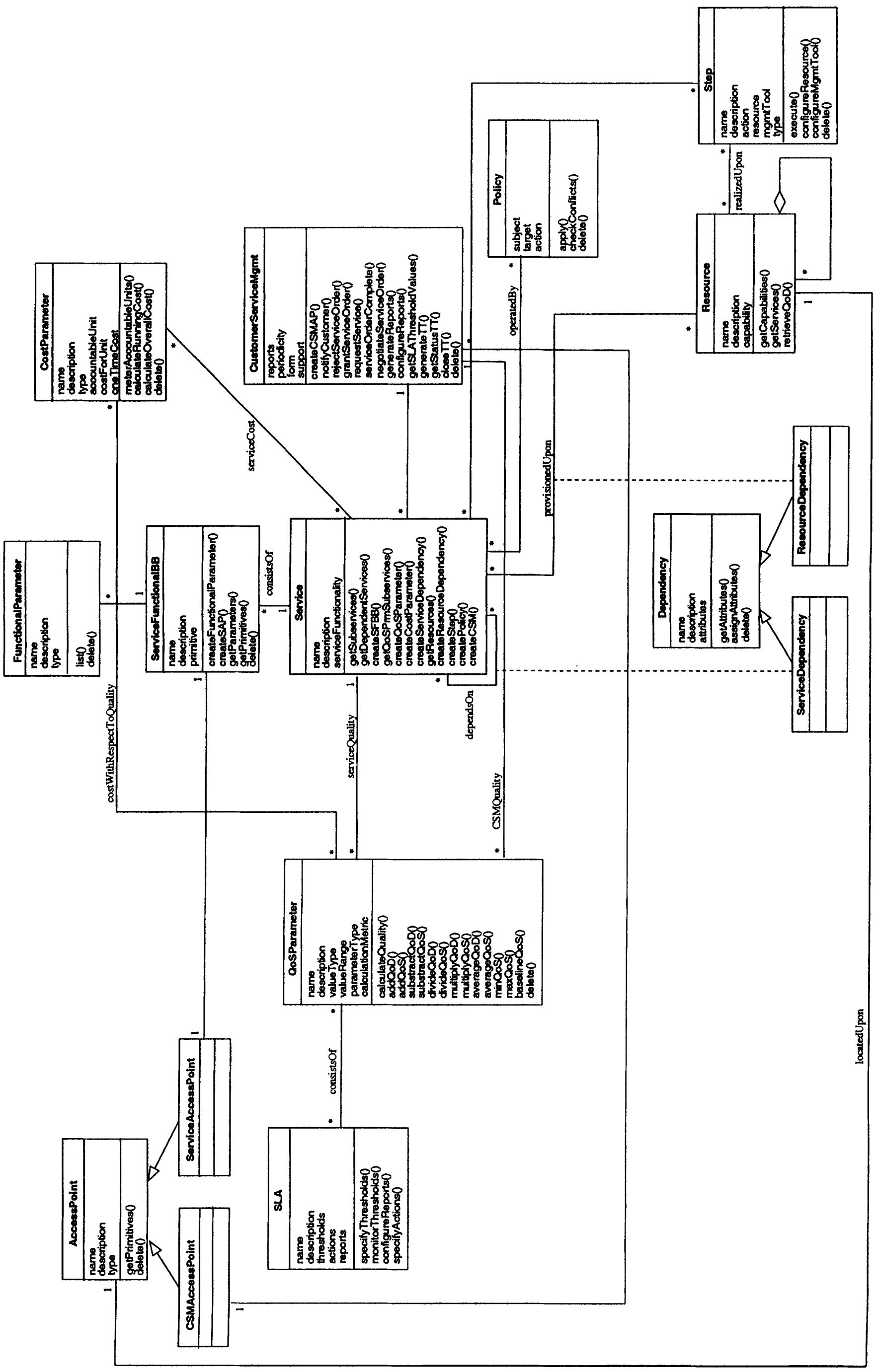


Figure 6. Design of the provider-centric service template model

or customize certain quality parameters (e.g., select the availability of a service greater than 99,98%).

The summarized view of the provider-centric service template model is shown in Fig. 6.

5. Customer-Centric Service Template Model

The last step of the methodology is the development of the **customer-centric service template model** to address the customer-centric issues. The customer-centric service template model is obtained by refining the provider-centric service template model and by adding attributes that represent customer-centric characteristics. This means that we add a class `Customer` to the provider-centric service template model as visualized in Fig. 6.

Customer-centric QoS parameters. They are a refinement of the provider-centric QoS parameters. A customer can select specific values from value ranges of QoS parameters which are offered by a provider. He may request also other values. In such a case, the negotiation about these values is started. The selected value is in relation with the cost. Another issue is that customer-centric quality parameters should express the quality of a service in a customer-centric way. Current practice is that QoS parameters are expressed as provider-centric parameters, such as packet or cell loss, reachability etc. The reason is obvious. Such QoS parameters are device-oriented and can be measured with existing device-oriented management tools.

Customer type. The customer type is used to distinguish between an intra- or inter-organizational provision of a service. This means to distinguish whether a service is provided to an external or to an internal (within an organization) customer. This is necessary because different processes (e.g., escalations, billing) may be used for an internal or external service provision.

Customer-centric CSM. A customer-centric CSM represents those reports about the provided quality of service that a customer has individually selected or configured from the available reports of the provider. A customer has the possibility to configure (i) what reports to access (e.g., specific QoS parameters, service levels), (ii) the periodicity of generating reports (monthly, daily, weekly etc.), (iii) the way, respectively the form, of providing reports (e.g., via email, Web, fax, paper). Beside the configuration of such reports, a customer may order new services, report problems with respect to his SLA, track resolution of problems etc. The customer-centric CSM can be considered as the customer's individual management interface to the provider.

6. Example

A simplified example of a problem management service should clarify the applicability of the proposed models. The class diagram in Fig. 6 extended with the class `Customer` should be consulted for that.

Service: Problem Management (PM) service with the functionality to support the resolution of customer-reported problems.

Dependency: The PM service depends on sub-services such as DNS, IP, database sub-service, file sub-service;

Service Functional Building Blocks: Examples are (i) documentation of problem resolution in trouble tickets, and (ii) workflow support. Examples of primitives are *submit(Problem), accessStatusProblem(TT-Number).*

Functional parameters: Examples are `Problem` which is a description of the reported problems or `TT-Number` to identify the trouble ticket which documents the reported problem.

QoSParameter: Examples are *first response time* and *average resolution time.* Value type for both parameters is real whereas the value range for both parameters is defined in the SLAs. The *first response time* is an example of a basic QoS parameter and the parameter *average resolution time* is an example of an aggregated QoS parameter.

SLA: An SLA consists of various QoS parameters for which thresholds, actions and reports are defined. An example of a threshold for the QoS parameter *first response time* is less than 4 hours in the case the reported problem is a critical one.

CostParameter: Accountable units for the PM service could be the number of problems (documented in trouble tickets) that can be reported by a customer in a certain time period. In addition a `costForUnit` could be associated to these reported problems.

Step: An example of a provisioning step is *configure_database(database_server). configure_NetworkMgmtPlatform(database_server);* is an example of an operational step. Withdrawal steps are the same steps as in the case of provisioning and operation, however, to withdraw the PM service.

CSM: CSM and the associated interface CSMAccessPoint define the attributes and operations that a customer can initiative over the CSM. Examples of operations are:
getProblemStatus(TT-Number);
getServiceQualityReport(average_resolution_time);
getServiceCostReport(month).

Customer: The type of a customer can be either internal or external.

Resource: Example of a resource is the database server as well as other network devices, providing IP sub-service, or end systems, providing DNS.

7. Assessment and Conclusions

The relevance of the developed service models can be summarized as follows: (i) definition of a **modeling approach** → object-oriented approach; (ii) definition of a **syntax for the description** of service-related management information → UML; (iii) specification of **service-related management information** and **service information** (e.g., what is a service, what is a SAP) in a customer-provider environment; (iv) specification of **relations** between the identified managed objects of IT service management (e.g., service and resource dependencies, cost and quality associations). An important contribution of the service models in addition is that they define elements which should be included in an SLA.

The proposed service model for IT service management represents the first attempt to provide a generic customer- and quality-oriented model for services which can be used as a common information basis for various applications. The proposed model defines commonly needed service-related terms, concepts and structuring rules in a general and unambiguous way. According to the methodology, a *service template model* is proposed to model service-centric issues of a service. A *provider-centric service template model* is further introduced to address provider-centric requirements mainly with respect to service provisioning and management. Lastly, a *customer-*

centric service template model is proposed to cover customer-centric aspects of service usage.

Areas for future work include: (i) further refinement of the specifications of service models (e.g., adding arguments to operations, specifying diagrams on the implementation level); (ii) using the proposed approach to specify a **library of services**. Such a library would include a set of *service templates* describing various services. Providers could access such library, select service templates and customize them into provider-centric service templates, and into customer-centric service templates respectively; (iii) building appropriate service management development tools (e.g., editors, consistency checkers) for the description of service templates, provider-centric and customer-centric service templates. Providers could use these tools to describe their specific services.

Acknowledgments

The author wishes to thank the members of the Munich Network Management (MNM) Team for helpful discussions and valuable comments on previous versions of the paper. The MNM Team is a group of researchers of the Munich Universities and the Leibniz Supercomputing Center of the Bavarian Academy of Sciences. It is directed by Prof. Dr. Heinz-Gerd Hegering.

References

[1] N. Damianou, E. Dulay, N. Lupu, and M. Sloman. The ponder policy specification language" *in: proceedings of the workshop on policies for distributed systems and networks, bristol, springer-verlag, lncs 1995.* pages 18–39, January 2001.

[2] G. Dreo Rodosek. *A Framework for IT Service Management.* habilitation thesis, University of Munich, June 2002.

[3] P. Ferguson and G. Huston. *Quality of Service - Delivering QoS on the Internet and in Cooperate Networks.* John Wiley and Sons, ISBN 0-471-24358-2, 1998.

[4] M. Garschhammer, R. Hauck, H.-G. Hegering, B. Kempter, I. Radisic, H. Rølle, H. Schmidt, M. Langer, and M. Nerb. Towards generic service management concepts — a service model based approach. In Pavlou et al. [11], pages 719–732.

[5] R. Gopal. Unifying Network Configuration and Service Assurance with a Service Modeling Language. In Stadler and Ulema [12], pages 711–725.

[6] Open Distributed Processing – Reference Model – Part 2: Descriptive Model. IS 10746-2, International Organization for Standardization and International Electrotechnical Committee, 1993.

[7] M. Langer, S. Loidl, and M. Nerb. Customer service management: A more transparent view to your subscribed services. pages 195–206, October 1998.

[8] A. Lazar, R. Saracco, and R. Stadler, editors. *Proceedings of the Fifth IFIP/IEEE International Symposium on Integrated Network Management V (IM'97)*, San Diego, USA, May 1997. Chapman & Hall.

[9] L. Lewis. *Service Level Management for Enterprise Networks.* Artech House Inc., ISBN 1-58053-016-8, 1999.

[10] E. Lupu and M. Sloman. Conflict Analysis for Management Policies. In Lazar et al. [8], pages 430–443.

[11] G. Pavlou, N. Anerousis, and A. Liotta, editors. *Proceedings of the Seventh IFIP/IEEE Integrated Network Management VII (IM'01)*, Seatle, WA, May 2001. IEEE Publishing.

[12] R. Stadler and M. Ulema, editors. *NOMS 2002 IEEE/IFIP Network Operations and Management Symposium — Management Solutions for the New Communications World*, Florence, Italy, April 2002. IEEE/IFIP.

[13] Service Architecture. Tina baseline, TINA Consortium, June 1997.

A REVENUE-BASED MODEL FOR MAKING RESOURCE INVESTMENT DECISIONS IN IP NETWORKS

Srinivasan Jagannathan,[1] Jörn Altmann,[2] and Lee Rhodes[3]
[1] *University of California Santa Barbara,* [2] *University of California Berkeley,* [3] *Hewlett-Packard Company*
[1]jsrini@cs.ucsb.edu,[2]jorn_altmann@acm.org, [3]lee_rhodes@hp.com

Abstract: Capacity planning is a critical task in network management. It identifies how much capacity is needed to match future traffic demand. It directly affects customer satisfaction and revenues. In this work we present a network usage analysis tool called *Dynamic Netvalue Analyzer (DNA)*, which helps alleviate a big problem that network engineers and marketing executives face– making optimal resource investment decisions. Marketing executives have to project customer growth while network engineers have to project traffic volume based on the entire customer population. DNA helps the prediction process by presenting actual network usage data from a business perspective, in a form that is useful to both network engineers and marketing executives. Using these projections, decisions on how to upgrade resources can be made. We show that information from DNA can be used to: (1) quantify revenue earned on each link, (2) quantify return-on-investment on performing a link upgrade, and (3) quantify the loss due to customer dissatisfaction when a link is not upgraded. We also illustrate how these formulations based on business information can be used to improve capacity planning decisions.

Keywords: Network and Systems Monitoring, Investment Cycle, Business Process, Network and Service Management

1. Introduction

The explosive increase in the number of Internet users as well as in volume of usage poses significant challenges to the network infrastructure and, by extension, to the network service providers. Network service providers are faced with two challenges today. On the one hand, they want larger number of customers in order to increase revenues. On the other hand, they want to manage the data volume efficiently. Capacity planning plays a crucial role in helping network providers tackle these challenges. Capacity planning is the process of predicting tomorrow's needs and preparing for them today. Network capacity planning involves combining marketing information and traffic analysis results to predict future resource requirements. Intelligent capacity planning can result in enormous cost savings and increased customer satisfaction. At the other extreme, poor capacity planning can result in enormous expenditures, poor customer service, and loss of revenues. The importance of capacity planning cannot be over-emphasized.

Capacity planning for Internet infrastructure requires good understanding of network traffic growth. The overall traffic growth depends on the number of new subscribers on the network as well as the usage increase per subscriber. Predicting these growth factors determines the decision on the investment size for upgrading network capacity. Currently, there is no standardized process to combine both growth factors.

Prediction of traffic growth and intelligent decision-making can be greatly facilitated by correlating network data with business-relevant information. Such information can be classified into:

1 Subscriber usage information: Considering usage of individual subscribers (or of subscriber segments) reveals more information than considering the aggregated usage of all subscribers.

2 Value of the subscriber to the business: The revenue and costs of an individual subscriber are important in evaluating the value of an investment. Clearly, an investment is worthwhile only if the revenue per subscriber outweighs the costs incurred per subscriber. Therefore analyzing the value of customers can help make educated investment decisions.

3 Competition in the market: Since competition in different geographical regions can impact subscriber loyalty, it is important to consider its impact before making costly investment decisions.

By associating such business information with raw traffic data, the network service provider can make better decisions about investments. However, today's capacity planning for the Internet does not incorporate this kind of information. The dominant reason is that the data collection and analysis process is not in place. The current process suffers from the problem that the data volume overwhelms conventional database management systems. Moreover, marketing and engineering disciplines of the service provider business lack a common vision. Marketing managers concentrate only on customer numbers and ignore the traffic-volume aspects of the business, while network engineers concentrate only on traffic volumes and ignore the customer aspect of the business. Each discipline views data in isolation, which results in myopic decision making. A holistic view of the data is necessary for informed investment decisions. In this paper, we present an innovative tool called *Dynamic Netvalue Analyzer (DNA)* that overcomes these lacunae. DNA aggregates, analyzes and models network data streams on the fly [1]. We show how this tool can be used to combine marketing, revenue, and engineering aspects of a service provider's business in order to make efficient capacity planning decisions.

In this paper, we only focus on capacity planning for regional networks. Our solutions are not directly applicable to capacity planning in backbone networks because of the following differences. First, the cost structures in these two categories of networks are dissimilar. For instance, laying optical fiber from coast to coast imposes very different costs than laying cable in an urban neighborhood. Second, the ratio between the traffic of an individual subscriber and the total traffic is significantly higher in case of a regional network. On the Internet backbone however, a huge number of individual flows is aggregated, thereby decreasing the impact of individual flows on overall traffic. Conversely, traffic of one individual subscriber in a regional network has much more impact on the overall traffic. Third, the size of the network and the volume of traffic in regional networks is much smaller. Therefore, capacity planning solutions for regional networks are not limited by scalability issues.

We now briefly discuss related work. Some of the previous work on capacity planning has focused on IT issues. Diao et al. propose an approach to maximize profits in service level agreements by designing feedback loops at application level [3]. Menasce and Almeida discuss performance issues and capacity planning for client-server systems [8]. On the network side, Robertazzi presents practical aspects of planning telecommunication and telephone networks [12]. Keshav discusses network capacity planning from the perspective of traffic management [7]. In addition, there are many software products that perform network capacity planning [11, 9, 6, 10].

The rest of the paper is organized as follows. In Section 2, we describe the capacity planning problem in greater detail. We describe current approaches to capacity planning and illustrate how a combination of marketing, revenue, and engineering analysis can greatly improve the planning process. In Section 3, we describe the DNA tool in more detail, and its versatility in collecting varied kinds of network data. In Section 4, we describe new algorithms to improve capacity planning decisions. We conclude the paper with a discussion on architecture and implementation issues in Section 5.

2. Capacity Planning

Network Capacity planning has three phases– (1) predicting future growth in customers, (2) predicting future volume of traffic, and (3) planning resource upgrades for the future. In the first phase, the marketing team estimates how many new customers will join the service and how many old customers will leave the service. The marketing team can use historical growth patterns, advertising budget, channel strength and customer satisfaction reviews, etc. to determine future growth and churn. This allows prediction of total number of users in the network. In the second phase, network engineers translate the number of customers into possible network traffic. This helps identify hot-spots in the network. Once the hot-spots are identified, in the third phase, the service provider must decide where investments are necessary in order to provide a good network service to customers.

For example, one simple approach to make investment decisions could be the following. The service provider sets a policy that all links should have overall utilization less than a threshold, $\tau_{suggested}$. Consider link l with capacity β_l that has a projected volume of traffic $b_{l,future}$ bits over the future time period T_{future}. Therefore, overall utilization of l, denoted by $\tau_{l,future}$ can be computed as $\frac{b_{l,future}}{T_{future}\,\beta_l}$. If this exceeds $\tau_{suggested}$, then the service provider marks the link to be upgraded. The quantum of upgrade, χ_l, should be greater than $\frac{1}{\tau_{suggested}} \times \frac{b_{l,future}}{T_{future}} - \beta_l$. This is obtained by solving the following inequality representing the service provider's policy decision:

$$\tau_{suggested} \geq \frac{b_{l,future}}{T_{future}\,(\beta_l + \chi_l)} \tag{1}$$

Using some such mechanism, the service provider can construct a set of resources S, that need to be upgraded/purchased, and also determine the capacities of these resources. In this paper, we assume that the service provider has done some analysis to determine this initial set S of resources that need to be upgraded/purchased.

In the rest of this section, we examine the state-of-the-art in capacity planning and ways to improve it using an illustrative example.

2.1 State-of-the-Art

The example we present to describe the state-of-the-art is contrived and simplified, but it serves to illustrate the inefficiencies of capacity planning as practised today. Consider the example network shown in Figure 1

Figure 1 shows a small network with two edge routers, A, and B, and a border router C. There are two main links in the network– AC and BC. A back-up link AB

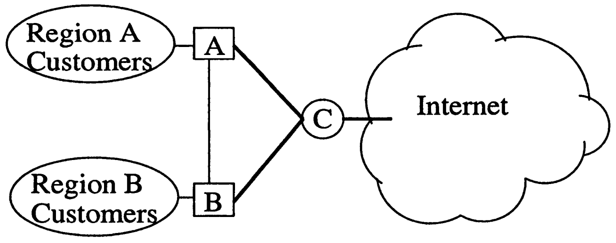


Figure 1. A Capacity Planning Example

is used for fault-tolerance. Customers are connected either to A or to B. Further, let us assume that there are two kinds of customers– residential and business. Residential customers pay a flat fee of $20 every month, while business customers pay a flat fee of $1000 every month. Currently, there are 20 business customers, 6 at A and 14 at B, and 200 residential customers, 150 at A and 50 at B. The problem of capacity planning involves analyzing existing traffic, predicting growth and making intelligent decisions on: (1) scheduling maintenance of AC and BC, (2) upgrading links AC, BC, AB, and routers A, B, and C, and (3) changing physical topology with new nodes and links.

Currently, service providers monitor the traffic using protocols like Simple Network Management Protocol (SNMP) [2]. A huge amount of data is collected, usually at 5 minute intervals. Let us assume the example network also uses SNMP and that a network engineer has analyzed SNMP data collected over the past 6 months (sampled in 5 minute intervals). The analysis result is that link AC has overall utilization of 60% and a peak utilization of 70% (over a 5 minute interval). Similarly BC has overall utilization of 51% and a peak utilization of 65%. The engineer also observes that link AB has less than 5% utilization.

The marketing team believes that it will acquire 5 residential customers each at A and B every month over the next 6 months. In addition, it estimates that it will acquire 1 business customer each at A and B every month over the next 6 months. Further, the marketing team believes that there will be no loss of customers in the next 6 months. This information is presented to the network engineer.

Currently, there is no industry-wide standard for interpreting marketing projections. Different service providers use different metrics. To the best of our knowledge (based on direct inquiries to network service providers), network engineers use some rule-of-thumb to convert marketing data to traffic volume predictions. For instance the rule-of-thumb may translate every new residential customer into 64kbps network capacity and every business customer into 512kbps network capacity. In addition they may assume that current customers' usage will increase 50% every 6 months. Some service providers use the traffic volume predictions obtained from such rules-of-thumb in sophisticated network simulation tools [11, 9, 6, 10] to analyze points of failure and then decide on investments and expenditures. But most service providers make decisions using simpler analysis. For instance, in this example, the network engineer can compute that after 6 months the requirement would be as follows. Current customers in A would require 0.6*10240*1.50 = 9216kbps. New residential customers will require 30*64 = 192kbps, and new business customers will require 512*6 = 3072kbps. This means that traffic on link AC will be 12480kbps which exceeds the link capacity. Similar analysis reveals that traffic on link BC will be 11098kbps which also exceeds the link capacity. Based on these results, the engineer may conclude that the capacity of AC and BC needs to be doubled. Furthermore, this capacity increase may require router C also to be upgraded. Suppose that upgrading AC and BC costs \$20000 and \$10000 respectively. And let a high-capacity router cost \$20000. Thus, using this analysis, capacity planning expenditures total \$50000.

In summary, today network capacity planning is an art and not a science. We now present our vision for capacity planning.

2.2 Improving Capacity Planning

Our vision of capacity planning uses information that is available, yet unused. For example, the service provider can ascertain how much data on each link belongs to each customer segment. This can be done by observing the source or destination IP address of network flows and then correlating this address to the customer segment assigned that address. The network engineer generates a histogram to study how usage by residential customers has grown over the past 6 months. Using this information, the network engineer can use standard mathematical techniques of extrapolation to estimate how much volume of traffic new customers will generate. By performing a similar analysis, the engineer can also estimate how much data the business customers will generate. Thus the network engineer can estimate overall traffic growth in a more objective manner.

For the example presented in the previous subsection, let us assume that the engineer performs subscriber-specific usage analysis and diagnoses that usage per subscriber has not grown over the past 6 months for both residential as well as business customers. Based on this result, he predicts that the same trend will continue. Since the marketing team predicts negligible customer churn, and that there will be many new customers over the next 6 months, the service provider needs to analyze if current capacity can sustain future traffic. Based on past statistics, the overall utilization on both the links AC, and BC will exceed 50%. Therefore if more customers are added, the service provider may decide on upgrading one or both the links.

The monetary value generated by the customers can also provide critical information for capacity planning. In this example, customers at A generate 6*1000 + 150*20 = \$9000 each month, and almost all the data they generate traverses AC. Similarly, customers at B generate 14*1000 + 50*20 = \$15000 each month, and almost all the data they generate traverses link BC. Clearly, link BC has been more lucrative over the past 6 months. Furthermore, based on the marketing projections, adding new customers to region A will increase revenues by (1000 + 5*20)*(1 + 2 + 3 + 4 + 5 + 6) = \$23100 over the next 6 months. Similarly, adding new customers to B will increase revenues by (1000 + 5*20)*(1 + 2 + 3 + 4 + 5 + 6) = \$23100 over the next 6 months. For simplicity, let us assume that link capacities can only be doubled. Since doubling capacity of link BC will cost only \$10000, while gain in revenue is \$23100, it is worthwhile to double the capacity of link BC. On the other hand, in case of link AC, the gain in revenue almost matches the cost of doubling the capacity. Furthermore, if capacity of AC is also doubled, then the router will also need to be replaced. There will therefore be a loss if one invests in link AC and BC together. Based on this analysis, the service provider can decide to: (1) double capacity of only link BC, (2) ask marketing team to not campaign aggressively in region A, and (3) ask marketing team to intensify campaign in region B.

The example illustrates the power of combining marketing, revenue and network usage data in devising better capacity planning solutions. A tool that collects, aggregates, and visualizes network usage and revenue data is a prerequisite for such a capacity planning scenario. In the next section, we describe one such tool.

3. Dynamic Netvalue Analyzer

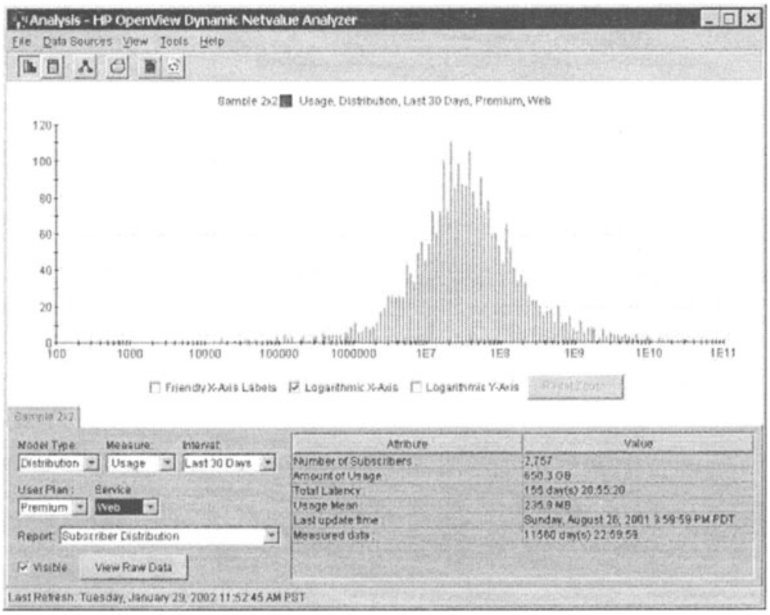
HP OpenView Dynamic Netvalue Analyzer (DNA) is a business intelligence and decision support tool targeted for network service providers [1]. It transforms raw customer usage data into business information, supporting business managers and network engineers to model revenues and profitability for new and existing services as well as network capacity upgrades. It helps network service providers to understand the usage behavior of their subscribers in real-time. In general, it enables new revenue and return on investment paradigms.

DNA uses statistical models to analyze customer usage data. A statistical model can be thought of as a histogram-based distribution of observed data. By converting raw usage data into statistical models (streaming data analysis), instead of storing all the raw data as in a warehouse approach, DNA frees storage space. It thus enables the analysis of more historical data in a shorter time period. Also, a benefit of this approach is that statistical models are small in size and do not grow with increased traffic volume - instead they get more accurate. The statistical models are based on business dimensions such as pricing plans, services, or geography. The only drawback of using such statistical models is that it is static; the business dimensions have to be decided well in advance. If new business dimensions are selected then new data has to be collected to populate the statistical models along these dimensions.

DNA is comprised of a backend server, which aggregates usage data, and clients to view and model business decisions. The backend server is built on HP's Internet Usage Manager (IUM) mediation platform. At the data collection stage, three modules of IUM are used to aggregate customer information, session information, and the actual usage data of all end-users. The second level correlates the output of the previous stage and transforms the data into an internal format, which allows rapid access of individual usage data. The third stage is comprised of statistical models, which are specified by the user of the tool. The models are populated by the second stage's output. The third stage also interacts with clients, which request model data. DNA clients allow network usage data to be viewed in numerous formats with varying levels of detail. Data is presented in statistical histograms, tables, and summaries. For an in-depth analysis, it allows the business manager to view individual subscriber usage.

How DNA can improve communication between network engineers and marketing managers, especially with respect to the capacity planning process, can be demonstrated using the analysis example shown in Figure 2. Instead of aggregating all usage data across all subscribers to one number, DNA splits up the data into those business dimensions (e.g. pricing plan, link, subscriber, etc) that are important for the decision making process. Figure 2 illustrates the usage of individual subscribers over a time period of 30 days for one customer segment under a flat-rated pricing plan. By looking at this distribution and the historical trends (such as mean and variance), the network engineers can predict the usage of new subscribers in this customer segment. The marketing manager, who analyzes this figure, notices the imbalance between light users and heavy users. As a consequence, he can work on introducing pricing plans that resolve this problem.

In detail, DNA allows an on-the-fly analysis of raw data coming from a NetFlow-enabled router. Considering the example of Figure 1, DNA would analyze the NetFlow data coming from router A and B. It would aggregate usage data in units of bytes with


Analysis - HP OpenView Dynamic Netvalue Analyzer
File Data Sources View Tools Help
Sample 2x2 Usage, Distribution, Last 30 Days, Premium, Web
120
100
80
60
40
20
0
100
1000
10000
100000
1000000
1E7
1E8
1E9
1E10
1E11
Friendly X-Axis Labels
Logarithmic X-Axis
Logarithmic Y-Axis
Sample 2x2
View Raw Data

Model Type: Distribution
Measure: Usage
Interval: Last 30 Days
User Plan: Premium
Service: Web
Report: Subscriber Distribution
Visible [✓]

Attribute	Value
Number of Subscribers	2,757
Amount of Usage	650.3 GB
Total Latency	155 day(s) 20:55:20
Usage Mean	235.9 MB
Last update time	Sunday, August 26, 2001 3:59:59 PM PDT
Measured data	11560 day(s) 22:59:59

Last Refresh: Tuesday, January 29, 2002 11:52:45 AM PST

Figure 2. DNA analysis result of subscriber usage behavior

respect to the customer segment (based on the pricing plan), the destination IP address, and link congestion. The result of the aggregation would be used to make the decision for resource investments for all links as described in the following section.

4. DNA Decision Making Process for Resource Investments

We now present an approach to make smarter resource investment decisions. The approach assumes that there is an initial set of resources S that need to be upgraded and that the quantum of upgrade for each resource is known. Our main contribution is to associate a number called *investment gain* with each resource upgrade in S. The investment gain associated with a resource upgrade indicates how much benefit is accrued from the upgrade per unit money spent on that upgrade. The problem of making the right upgrade decisions then reduces to solving an optimization problem where the total investment gain is maximized subject to budgetary constraints on the investment costs. The investment gain $\Phi(l)$ for each link l in the network is based on two business criteria:

1 The size of the return-on-investment (RoI). The RoI depends on the quantum of upgrade, growth in customer base as well as on projected volume of traffic per customer segment.

2 The loss of customer satisfaction. The loss is directly correlated with the loss caused by not upgrading. Customers who are unhappy with the service may choose a different service provider thereby decreasing revenue.

The investment gain $\Phi(l)$ is calculated according to the following formula:

$$\Phi(l) = \frac{\gamma_{l,T_{future}} + \zeta_{l,T_{past}}}{FC_l + M_{l,T_{future}}} \tag{2}$$

where $\gamma_{l,T_{future}}$ is defined as the revenue earned on link l in future time period T_{future}, $\zeta_{l,T_{past}}$ represents the monetary value of customer unhappiness with link l during the past time T_{past}, FC_l is the fixed cost for upgrading link l by a quantum χ_l, and $M_{l,T_{future}}$ is the additional maintenance cost of l in time period T_{future}.

The numerator represents the sum of revenue earned and the monetary value of gain in customer satisfaction as a result of the upgrade. We assume that the quantum of upgrade is such that all the customers who were discontent with service before the upgrade are no longer discontent. The denominator represents the cost of the investment. The ratio reflects what is gained per unit of investment. The higher the ratio, the greater the priority for upgrade. Suppose that a network provider has to choose the right set of resource upgrades from set S, given a budget constraint B. For each resource r_i in S, we ascertain $\langle\ FC_i,\ \Phi(r_i)\ \rangle$ where FC_i is the cost of the upgrade and $\Phi(i)$ the investment gain. Let δ_i denote the decision to upgrade r_i, i.e., $\delta_i = 1$ represents a decision to upgrade and $\delta_i = 0$ otherwise. Then the network provider can choose the right set by solving the following optimization problem: choose $\delta_i \in \{0, 1\}$ to maximize $\sum \Phi(r_i)\delta_i$ subject to: $\sum FC_i\ \delta_i \leq B$. This optimization problem is a version of the 0-1 Knapsack problem which is known to be NP-Complete. If the cardinality of the set S is large, then any of the well known polynomial time approximation algorithms for 0-1 Knapsack problem can be used to solve the resource investment problem.

In our model, even though we focus on link upgrades, the approach is easily extended to include other resources like routers. Router investment costs can be subsumed[1] into the costs of the first link that necessitates a router investment.

In order to quantify return-on-investment and loss of customer satisfaction, we introduce two algorithms to compute $\gamma_{l,T_{future}}$ and $\zeta_{l,T_{past}}$ in the following two subsections.

4.1 Quantifying Loss of Customer Satisfaction

Rational customers are discontent with the service if the service provider does not meet QoS requirements. This occurs if one of the links of the data route is congested. By observing the degree of congestion at each link, we can compute the degree of customer dissatisfaction at a particular link.

For each link, we assume that the business manager sets a threshold. If the data transmitted per unit time on that link exceeds the threshold, then we shall assume that the link is congested. For example, the network provider can follow a policy that if the data transmitted on a link in a 5 minute interval exceeds 70% of the total link capacity then that link is considered to have been congested in that interval.

Since all customers who are using a link when it is congested are equally affected, we shall quantify customer dissatisfaction as follows. Compute the monetary value of bytes transmitted during congested intervals on a link. This represents revenue earned when customers are discontent with the service. Now, let us assume that the business manager weights the importance of customer satisfaction to his business on a scale

[1] In this case FC_i is not known initially, and will vary with the candidate solutions of the knapsack problem.

between 0 and 1. Suppose this weight is θ. Then, the monetary value of customer dissatisfaction ζ can be computed as described in Figure 3.

```
Input:
   Network topology $N$, and link maintenance costs
   Routing tables for time period $T_{past}$
   Customer flows $F_{T_{past}}$ for time period $T_{past}$
   Customer billing information for time period $T_{past}$
   Link utilization congestion threshold $\tau_{max}$
   Relative importance of customer satisfaction $\theta$ (a
   number in the range [0,1])

Output:
   Customer dissatisfaction $\zeta_{l,T_{past}}$ for each link $l$
   within time $T_{past}$

Algorithm:
   For each link $l$
      Initialize customer dissatisfaction, $\zeta_{l,T_{past}} = 0$
   Using flow information and topology:
      For each flow $f \in F_{T_{past}}$ and small time intervals $t$
      Let $n_{f,t}$ be the number of links
      in path taken by $f$ over time interval $t$
      Let $b_{f,t}$ the number of bytes of $f$ transmitted
       in interval $t$
      Let $p_f$ be the cost of transmitting a single byte
       in flow $f$
      Let $\beta_l$ the capacity of link $l$
        Let $v(f,t) = \frac{b_{f,t} \times p_f}{n_{f,t}}$
        For each link $l$
          For each small interval $t$
            If $\frac{\sum_{f \in F_{T_{past}}} b_{f,t}}{\beta_l \times t} \geq \tau_{max}$
            $\zeta_{l,T_{past}} = \zeta_{l,T_{past}} + \theta \times \sum_{f \in F_{T_{past}}} v_{f,t}$
End.
```

Figure 3. Computing customer dissatisfaction

4.2 Quantifying Return-on-Investment

Return-on-Investment depends on future revenues that the service provider hopes to earn. Therefore, to quantify RoI, we also need a mechanism to estimate future revenues. Suppose that there exists some mechanism to project growth of data volume as well as growth in the number of customers. Let us call this algorithm $Pred - Algo$. Then the return-on-investment can be computed as shown in Figure 4. The algorithm takes as input, the network topology, future routing information, current maintenance costs, quantum of upgrade for different links, the cost of these upgrades, and future maintenance costs. It produces as output, the projected revenue on all links for a future time period T_{future}.

The algorithm for quantifying return-on-investment assumes that we know how to quantify revenue earned on a link. To this end, we now develop an algorithm to distribute revenues among the links in the network. Our algorithm considers how much each link has been used by each customer. Then, using pricing plan information, the algorithm partitions the customer segment's payment to each of the links based on vol-

```
Input:
   Network topology, and link maintenance costs
   Amount of extra capacity to be added to each
   link $l$ and investment and maintenance costs
   Customer usage information for time period $T_{old}$
   Customer billing information for time period $T_{old}$
   An algorithm $Pred-Algo$ to project usage and
   revenue for the future time period $T_{future}$

Output:
   $RoI_{l,T_{future}}$, the return on investment
   for each link in $T_{future}$

Algorithm:
   For each customer $C$ (including future customers)
      Apply $Pred-Algo$ to project growth of usage over
      ⟨source, destination⟩ pairs and payment $P_C$
      in time period $T_{future}$.
   Let $\gamma_{l,T_{future}}$ be revenue earned on link $l$
   in interval $T_{future}$
   For each link $l$ compute $\gamma_{l,T_{future}}$
      Let $FC_l$ be fixed cost of upgrading $l$
      Let $M_{l,T_{future}}$ be the maintenance cost of $l$
      in interval $T_{future}$
         $RoI_{l,T_{future}} = \frac{\gamma_{l,T_{future}}}{FC_l+M_{l,T_{future}}}$.
   End.
```

Figure 4. Estimating Return-on-Investment on upgraded links

ume of usage. The value of each link then, is the sum total of the money earned by that link from all customers. By comparing the value of each link with the investment and maintenance costs, we get a reasonable estimate of the importance of each link to the NSP's business over a given time period. The algorithm is presented in Figure 5

The algorithm outlined in Figure 5 estimates the price of each flow and equally distributes the payment among all the links in the path taken by that flow. Notice that the way we choose to distribute revenues and estimate value of a link is just for "accounting simplicity" and by no means an exact measure. It however gives us adequate perspective about which links are on popular revenue-generating routes. The algorithm in Figure 5 makes an implicit assumption that there exist alternative paths to every destination, and that should a link in one of the paths fail, the alternate path can sustain the traffic. Otherwise, a bridge link will be worth as much as the revenue earned from traffic traversing that link. The path redundancy assumption is justified because of two reasons: fault tolerance of network and computational complexity. Path redundancy is highly desirable from the perspective of fault tolerance. Hence almost every real-world network can be expected to be a strongly connected graph with redundant paths to every destination. The computational complexity of estimating whether an alternate path can sustain the traffic load is prohibitive[2]. Assuming that alternate paths can sustain

[2]For each edge in a given path in an undirected graph, one can estimate the impact of removing an edge in that path, using a modified shortest path algorithm [4, 5]. But this approach cannot be used for directed graphs, and the problem is provably harder.

the traffic however allows us to perform a fast computation to estimate link value at the cost of some loss in accuracy.

```
Input:
  Network topology $N$ and link maintenance costs
  Routing tables for time period $T$
  Customer flow information for time period $T$
  Customer billing information for time period $T$

Output:
  The revenue $\gamma_{l,T}$ generated by each link

Algorithm:
  For each link $l$
    Initialize revenue earned by that link, $\gamma_{l,T} = 0$
  Using flow information and topology:
    For each flow $f$
    Let $n_{f,t}$ be the number of links in path
    taken by $f$ over time interval $t$
    Let $b_{f,t}$ be the number of bytes of $f$
    transmitted in interval $t$
    Let $p_f$ be the price for transmitting one byte of $f$
    For small time intervals $t$
      For each link $l$ in path
        $\gamma_{l,T} = \gamma_{l,T} + \frac{p_f b_{f,t}}{n_{f,t}}$
  End.
```

Figure 5. Distributing revenues among links

We now illustrate our algorithm using an example. Figure 6 shows a network service provider (NSP) with two customers– XYZ and ABC. XYZ pays a monthly rate of \$20000 for time critical data that it transmits to other locations. ABC pays a monthly rate of \$5000 as data transmission costs. The NSP pays \$5000 a month to the backbone provider for data transmission. The maintenance costs for each link are indicated in the figure. The figure also shows how much data has been transmitted by each customer (say in the past month) and which links the data traverses on its way to destinations. XYZ sends/receives 50MB data on link A-B and A-C and 200 MB data on link B-D. ABC sends/receives 300 MB data on link A-C. For simplicity, we shall ignore the links connecting ABC and XYZ to nodes A and B respectively.

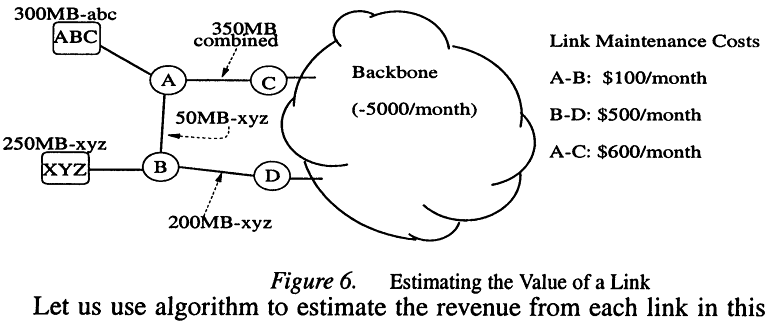


Figure 6. Estimating the Value of a Link

Let us use algorithm to estimate the revenue from each link in this network. We assume that the entire volume of data over a link was generated in a single flow. Since payment of customer ABC is \$5000, the price of each Megabyte is $p_{ABC} = \frac{5000}{300} = \$16.67$. For customer XYZ, the payment is \$20000. Hence $p_{XYZ} = \frac{20000}{250} = \$80$. ABC has only one flow of volume $300MB$ over path A-C which contributes \$5000 to $\gamma_{A-C,T}$. Customer XYZ has two flows, one over B-A-C and the other over B-D. The former has a data volume of 50MB and contributes $\frac{1}{2} \times \frac{50}{250} \times 20000 = \$2000$ each to $\gamma_{A-B,T}$ and $\gamma_{A-C,T}$. Similarly, the second flow contributes $\frac{200}{250} \times 20000 = \$16000$ to $\gamma_{B-D,T}$. Thus we have, $\gamma_{A-C,T} = \$7000$, $\gamma_{A-B,T} = \$2000$, and $\gamma_{B-D,T} = \$16000$.

5. Discussion

In this paper, we have described a new approach to capacity planning. We illustrated the power of combining marketing, revenue, and customer usage information. We then showed that the Dynamic Netvalue Analyzer (DNA), a tool that can aggregate and analyze raw network data on-the-fly, can be used for this new approach to capacity planning. In more detail, we outlined algorithms that use DNA-aggregated data for making investment decisions.

We now discuss some architectural issues and tradeoffs associated with some possible implementations. An important issue that arises is the location of the data collecting agents within the network. Network managers are very reluctant to make changes to the network or install monitoring equipment on important or busy routers. Our architecture does not require DNA at any of the internal or border routers. DNA runs at the network edge. Because all traffic has to enter or exit through one of these edge-routers, information collected from these routers is sufficient to capture the entire network scenario. Our algorithms do require routing information from the network. One possible approach is to get a daily update of routing tables from all routers in the network. A daily snapshot will suffice if internal routes are relatively stable over time. Even in case of route instability over short intervals of time, our analytical results will not be significantly affected because capacity planning is a long-term process and uses data over long intervals of time (often using data gathered over months).

The algorithms we have described are implementation independent. They can be implemented over a sophisticated network simulation tool, or they can be implemented inside the monitoring tool itself, thus performing all the analysis on-the-fly. A simulation based approach has significant data storage and transportation overheads. To perform a realistic simulation of observed traffic, we need detailed information about network flows to be collected, transported and stored at a centralized location. Furthermore, the simulation will be computation intensive and could be time consuming. However, since capacity planning is a long-term process, a dedicated machine can accomplish the task. A possible optimization is to use "aggregated network data" or traffic models instead of detailed flow information. This will reduce the accuracy of the analysis, but will also significantly reduce the data transportation and storage costs. DNA can be used to perform such traffic modelling [1].

If the second approach is adopted, our algorithms that compute monetary value of links and customer dissatisfaction are implemented within DNA. This may require approximations of the price of transmitted data because the actual price charged for transmission may not be known in real-time (e.g. tiered-pricing plans). Moreover, some coordination may be required with a network management protocol like SNMP. Specifically, the network management protocol should trigger alarms whenever link utilization over observation intervals exceed pre-configured thresholds. This is required to compute the customer dissatisfaction with links. But at the cost of administrative overhead, this approach saves significant data transportation and storage costs.

An important issue concerning our capacity planning vision is in evaluating the trade-off between information utility and data collection overhead. Data collection and analysis, if done efficiently, can deliver business information, whose value outweighs the cost of the data collection. We believe that DNA and our algorithms for investment decisions deliver this efficiency.

References

[1] J. Altmann and L. Rhodes, "Dynamic Netvalue Analyzer - A Pricing Plan Modeling Tool for ISPs Using Actual Network Usage Data", IEEE WECWIS2002, International Workshop on Advance Issues of E-Commerce and Web-Based Information Systems, 2002.

[2] J. Case, K. McCloghrie, M. Rose and S. Waldbusser, "Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1905, 1996.

[3] Y. Diao, J. L. Hellerstein, and S. Parekh, "A Business-Oriented Approach to the Design of Feedback Loops for Performance Management", Distributed Operations and Management, 2001.

[4] J. Hershberger and S. Suri, "Vickrey Prices and Shortest Paths: What is an Edge Worth?", IEEE Symposium on Foundations of Computer Science, 2001.

[5] J.Hershberger, S. Suri and A. Bhosle., "On the Difficulty of some Shortest Path Problems," STACS, Berlin, 2003.

[6] IPAT, http://www.wandl.com/html/ipat/IPAT_new.cfm

[7] S. Keshav, "An Engineering Approach to Computer Networking: ATM Networks, the Internet, and the Telephone Network", Addison-Wesley, 1997.

[8] D. A. Menasce and V. A. F. Almeida, "Capacity Planning for Web Services: Metrics, Models, and Methods", Prentice Hall PTR, 2002.

[9] NetQuad, http://www.anitetelecoms.com/products/NetQuad.html

[10] NetRule, http://www.analyticalengines.com/

[11] Opnet, http://www.opnet.com .

[12] T. Robertazzi, "Planning Telecommunication Networks", Wiley-IEEE Press, February 1999.

SESSION 4

Policy-Based Management

Chair: Morris Sloman
Imperial College, UK

POLICY PROVISIONING PERFORMANCE EVALUATION USING COPS-PR IN A POLICY BASED NETWORK

A. Corrente, M. De Bernardi, R. Rinaldi
Cefriel, Getronics

Abstract: This paper presents a study carried out to evaluate policy provisioning performance, by COPS-PR protocol, in a policy based environment. First of all the prototypes developed are presented. These simulate the component behavior of the two lower layers of a policy based management system. Then the various measurements, done on the prototypes, are shown. All measures are carried out for several scenarios, described through the use appropriate configuration policies, in an increasing complexity order. It is demonstrated that the architecture of COPS-PR is scalable, though the PDP can become a bottleneck.

Key words: *Configuration Management and Policy-Driven Management, Policy Provisioning, COPS-PR, PRC*

1. INTRODUCTION

Configuring networks is becoming an increasingly difficulty task. The problem exacerbates as networks grow in dimensions (number and type of devices. To face this growth, administrators discovered that they have new requirements for the configuration of their networks. One of these new requirements is "configuration provisioning" to a specific set of devices. An example of this type of configuration would be instructing all routers, along a path in the network, to provide a specific quality of service to a particular set of customers. Historically, network operators used mechanisms and protocols such as SNMP (Simple Network Management Protocol) [1] and CLI (Command Line Interface) to provision configurations to network devices. Today these protocols have some difficulties due to their low level granularity and their device specific nature. Today it's clear that network administrators and the existing software of configuration management can't face the complexity growth of networks. Some IETF working groups (Policy Framework WG) suggest solving these problems by moving configuration management from device to network layer. This high level management can be obtained with the specification of management policies and their mapping into proper configurations, which have to be distributed to the "right" network devices to be enforced. This abstraction mechanism allows to have simplified management data and to exploit commonality across devices. These two features, simplified management data and re-use of configurations, can lead to the automation of some configuration management tasks allowing the network to operate with minimum human intervention. This work presents a study carried out to evaluate the performance of configuration provisioning by COPS-PR protocol [3], in a policy based network. As case study we have considered the provisioning of DiffServ (Differentiated Services) configurations to a DiffServ enabled device. Section two presents the architecture of a policy based network and the interaction between its components for the configuration provisioning. Section three presents the developed prototypes. Section four presents the performance measurements and, finally, section five the results. The presented work is innovative since it investigates policy provisioning and, evaluates the efficiency of policy based configuration management.

2. BACKGROUND

Policies are set of rules that allow defining the desired behavior for network resources or for distributed systems. As defined in [4], policies are a mean to translate high level objectives into proper configuration of network devices. To be implemented in an automated environment as a network, policies must be transformed from abstract rules to functional ones. At this level policies are ECA (event-condition-action) rules [4]. IETF Policy Framework WG works especially on the "condition action" part to define a UML information model for the representation of policy information [5] [6]. These models can be extended to represent policy for a specific management area (e.g. QoS policy information model). To control a network via policies, an administrator needs a set of software tools that allow the definition and deployment of policies to network devices. With "policy based network" we mean the set of tools and systems that are at administrator's disposal and are policy compliant. The major components of this architecture are PDP, PEP and COPS-PR protocol. In this work we have studied the distribution of DiffServ configuration [9].

2.1 PDP/PEP "configuration provisioning" interaction

This subsection describes the interaction between a PDP and a PEP in the "configuration provisioning" scenario [3].

When a device boots, it opens a COPS-PR persistent connection to its primary (default) PDP. After the connection is established the PEP sends, information about itself to the PDP, as a configuration request. This information includes the controlled device capabilities and PEP specific information used by the PDP to determine relevant policies for that PEP. In response to this request, the PDP sends policies/configurations to the PEP for the enforcement. In the paper we will refer to this behavior as *"start up"* interaction. If the PDP detects any events that require a configuration change, then it sends the changes to the PEP. The events that cause the PDP to reconfigure policies could be: the deployment request of a new configuration made by the network administrator, the activation of an higher level policy, the configuration request to the PDP made by other sources besides the PEP. We have called this behavior *"external event"* interaction.

3. THE DEVELOPED PROTOTYPES

During the work have been developed two pairs of PDP/PEP, called "start up" and "decision", as well as two PIB for the representation of policy information. All the prototypes have been developed in C++ language on Linux OS (RedHat distribution, version 6.2). For the development of PDP and PEP we have used a free set of COPS-PR API (Application Program Interface) developed by the Vovida Organization.

3.1 The two PIBs

A PIB is a logical database used for policy information which allows the PDP and PEP to share information in a standard way, using a common information syntax and semantic. Policy information is represented by PRC (Provisioning Classes) and related instances PRI (Provisioning Instance). We have developed part of two different PIBs: the framework [7] and the DiffServ [8] PIB. The implementation of the first one allows the organization of network device capabilities into an "interface set", as well as, the representation of the support information, needed by the PDP, to determine the relevant configurations for a PEP (e.g. role combination). The second permit to represent policy for the configuration of a DiffServ enabled device interfaces. The DiffServ PIB knows the concept of "data path" configuration [9]. This is the sequence of processing elements that must elaborate an IP packet so that it can obtain the correct DiffServ treatment. In our DiffServ PIB implementation the IP traffic can be classified (by a filter or by an access list); the classified traffic can be marked with a DSCP (Differentiated Service Code Point) code (by a packet marker); the marked traffic can be queued as input to a scheduler. Each element (the filter, the packet marker, the queue and the scheduler), according to standard DiffServ PIB, is modeled by one or more PRCs and is configured in compliance with the QoS policy we want to express. We have also implemented some PRCs of the DiffServ PIB that allow representing the DiffServ capabilities of an interface (e.g. classification capabilities, queuing and scheduling).

3.2 PDP and PEP

The first pair of PDP/PEP is called "start up". These two prototypes simulate the interaction between a PDP and a PEP that request its initial configuration ("start up" interaction, see 2.1). The second pair of PDP/PEP is called "decision". These prototypes simulate the interaction between a PDP and N PEP, when the PDP detects an "external event" that involves a (re)configuration of PEPs ("external event" interaction, 2.1).

Start up PDP and PEP

The start up PDP is a multi threaded server that can manage N start up PEP. Figure 1 shows the interaction between PDP and PEP.

The PEP connects to the PDP and sends, as COPS-PR request message, the DiffServ capability, of the interface that it controls, and the information used by the PDP to determine the relevant configuration for it. The PDP installs the capability in its own PIBs and, then sends the DiffServ configuration to the PEP in a COPS-PR decision message. The DiffServ configurations are contained into the DS PIB, previously loaded in memory by the PDP. The implemented PDP is state-full. To maintain the correct association between a single PEP and its capability, the PDP inserts the PEP identifier (PEPid object) and the capability received into appropriate associative maps. To build the decision message with the DiffServ configuration, the PDP exploits the DiffServ "data path" concept. The sequence of the constituent elements of the data path is fixed, while their number and their interconnection pattern is variable. The PDP obtains the configuration by browsing along the data path and exploring it completely, using the interconnection information included in every PRC. The "correct" data path is determined by the PDP using the "role combination" information received from the PEP. When receiving the decision message the PEP parses and installs it, copying, the configuration in its own DiffServ PIB.

Decision PDP and PEP

Our implemented "decision PDP" is a multi threaded server that takes an operative event and, as response to this, configures N connected "decision PEPs". The decision messages with the configuration are built as previous. The PEP parses the received message and installs the configuration in its DiffServ PIB. The operative event we selected is the Nth COPS OPEN registration message received by server. This event is simple to implement, to monitor and, from measurement perspective, it is logically equivalent to an external request or a higher-level policy activation. Figure 2 shows the interaction between a decision PDP and its client.

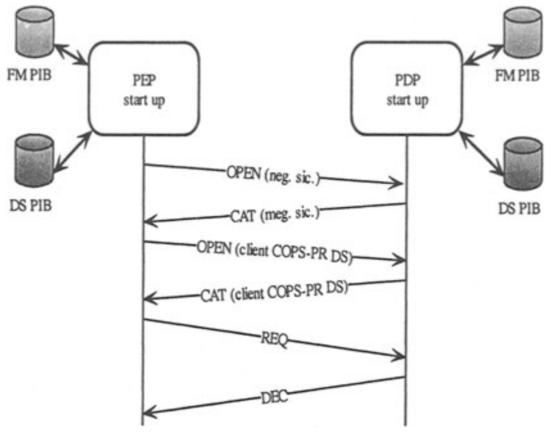


Figure 1. **PDP/PEP start up interaction.**

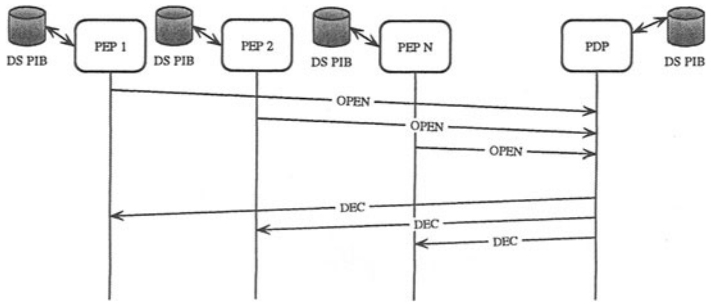


Figure 2. **Decision PDP/PEP interaction**

4. EXPERIMENTAL ENVIRONMENT

Based on the implemented PDP, PEP, and PIB prototypes a set performance measurement has been carried out. The executed measures have been designed to cover, in an exhaustive way, the two lower levels of the architecture of a policy based network (PDP and PEP). All measures have been also repeated with four different DiffServ configuration in order to evaluate the impact of the policy complexity on the recorded times. We have used two different test bed. In the first test bed (called "start up" test bed) a computer is dedicated to the start up PDP process and a computer is dedicated to the corresponding PEP. In the second test bed (called "decision" test bed) a computer is dedicated to the PDP process and two separated machines host up to N PEP running in parallel, with a maximum of 500 processes. In both test beds the computers are connected by a 100 Mbit switched LAN.

4.1 Measurements

Start up interaction time (T*start-up*).

With this measurement we want to estimate the time for a start up interaction on the server side. It is done (in the "start up" test bed) with one "start up PDP" and one "start up PEP", since is improbable that more than one PEP starts up simultaneously, or that a PEP starts before that the PDP ends to serve the previous one. We measure the time elapsed between the arrival at the PDP of the first "OPEN" message and the return of the "send" function of the decision message (Figure 1). T*start-up* will depend on: the number of the messages exchanged during the interaction, their dimensions and the processing related to every message (e.g. the necessary time for the construction of the DEC message). The following table 1 shows the exchanged messages and their dimensions. One can see how the dimensions of the DEC message changes according to the contained policy.

Table 1. Message dimension

	PEP → PDP			PEP → PDP	
# MSG	MSG	Dim. (Byte)	# MSG	MSG	Dim. (Byte)
1	OPEN(sic)	40			
			2	CAT(sic)	
3	OPEN (Client)	40			
			4	CAT(Client)	
5	REQ	904			
			6	DEC	Pol. A= 528
					Pol. B= 936
					Pol. C=1256
					Pol. D=1616

Decision time (Tdecision).

With this measurement we want to estimate the time, needed by a "decision PDP", to provision N "decision PEPs" with a DiffServ configuration. The measure is done with the "decision" test bed. We measure the time elapsed between the recording of the "operative event" and the return of the "send" function of the last decision message to the last served PEP (Figure 2). In this interaction the PDP process N decision messages, each sent to a PEP. Tdecision will depend on: the number of the PEPs that must be configured by the PDP and the policy complexity. The dimensions of the DEC message (see previous table) and the time necessary to it construction will reflect on the last contributing factor.

Provisioning time (Tprovisioning).

With this measurement we want to estimate the entire provisioning time, including the installation time of configurations by the last "decision PEP" (client side). We measure (in the "decision" test bed) the time elapsed between the sending of the last PEP registration message (COPS-PR OPEN message) and the installation of the DiffServ configuration made by the last PEP served (Figure 2). Also in this measure, as in the previous one, N decision messages are exchanged and processed.

Policy information base update time, (TPIB).

With this measurement we want to estimate the updating time of the DiffServ PIB on PEP side. We measure the time elapsed between the return of the "receive" function of the decision message and the end of the PIB update function. We have used the "time stamp counter" of the Pentium processor for time measurements. The counter is incremented by one every clock cycle and can be used to record the execution time of a set of instructions. This time is the difference between the value of the counter just after and just before the instruction block, divided for the processor frequency. The value of the counter can be read using the assembler (Intel) instruction RDTSC (read time stamp counter). This method has been chosen for its high precision (theoretically only one clock cycle) and for its low overhead. This method is applicable to the proposed measurements with a few other considerations due to the multi-threaded execution of the PDP.

5. THE PROVISIONED POLICIES, THE RESULTS

This section presents the results. Initially we discuss the policy and the data path, while the last section deals with the results done on these policies.

5.1 Policy A, basic complexity

This case study considers the scenario of a financial department that is the source of FTP (File Transfer Protocol) traffic. This traffic must be aggregated and marked with a DSCP code equal to AF11 (Assured Forwarding 11) for its treatment by nodes of the DiffServ domain. The aggregate flow must be granted 10%- 15% of the total available bandwidth. Figure 3 shows the data path representing the policy and, thus, the configurations sent from PDP to PEP for the policy enforcement. Each rectangle is an entry of a specific PRC of the DiffServ PIB of the PDP. Note how the condition part of the policy ("IF" part) is translated into an appropriate access list of the PIB (classifier element and IP filter), while the action part is rendered through specific PRCs (DSCPMarkAction, Queue, Scheduler) with appropriate parameters.

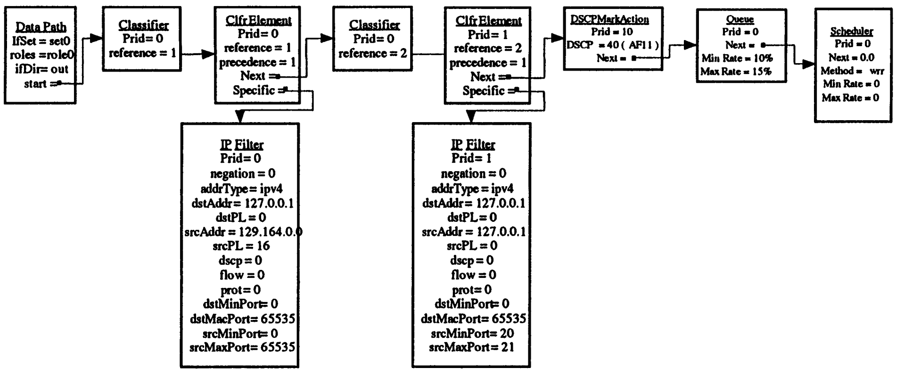


Figure 3. Policy A data path

5.2 Policy B, medium complexity

This case study considers the scenario of a design department with a remote projects database. Thus the main traffic is a FTP flow to the DB to which a bandwidth between 15% and 50% (of the total) must be granted, to reflect its strategic relevance. The FTP flow must be marked as AF11 traffic. The department generates both Telnet and SMTP (Simple Mail Transfer Protocol) traffic in addition to FTP. Telnet Traffic is granted 10 - 30% of the bandwidth and is marked as AF12, while to the SMTP is given 5 - 10% and is marked as AF13.

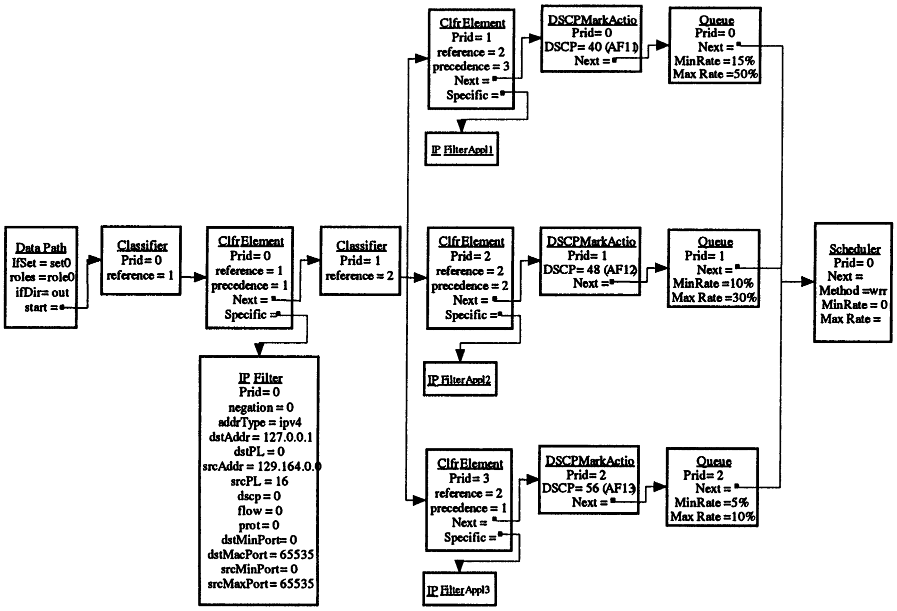


Figure 4. Policy B data path

5.3 Policy C, high complexity

This case study considers the scenario of a sub-net source of various traffic flows, from five different applications. These, however, can be divided into three classes of traffic: valuable, medium interest and low interest traffic. A single application (e.g. VoIP) generates valuable traffic which must be marked as EF (expedited forwarding) and granted 10 - 15% of the total bandwidth. Two different applications, (e.g. web application and FTP) generate medium traffic which must be granted 5-10% of the bandwidth with packets marked as AF11 and AF12 respectively. Another two applications belong to the low interest traffic class (e.g. telnet and SMTP) so are allocated 25 - 5% of the bandwidth with packets marked as AF21 and AF22 respectively.

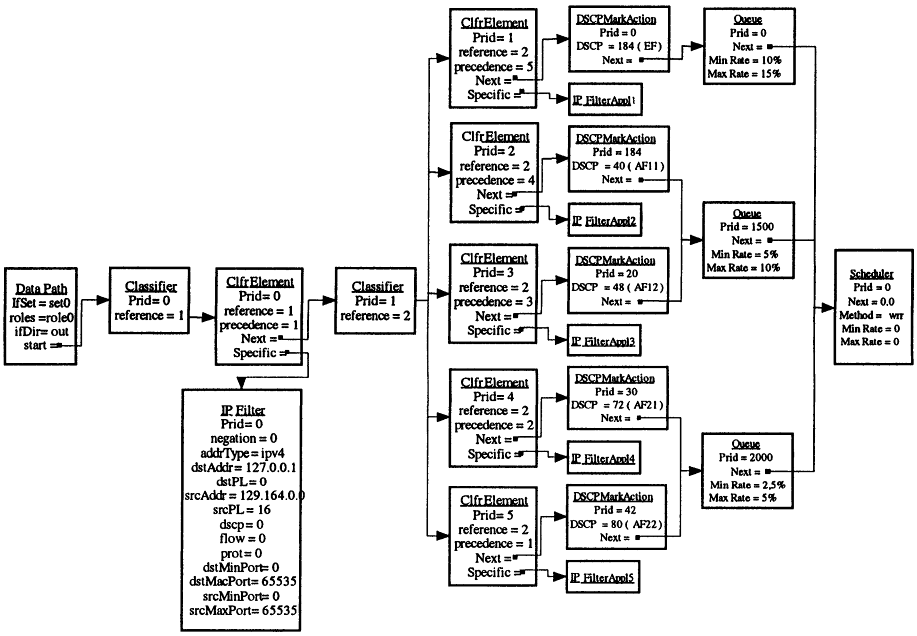


Figure 5. Policy C data path

5.4 Policy D, very high complexity

This scenario is similar to the previous one but with traffic from seven different applications. Three classes of traffic are distinguished. The first one is constituted by a single EF application with 20 - 25% of the total bandwidth. The other two classes consist of three applications. Application packets of the first family are marked as AF1i (i = 1,2,3), while application packets of the second as AF2i (i = 1,2,3). To these two classes is granted a bandwidth between 10% and 20% of the total. The granted band is divided into the two families as follows: 40 - 60% (of the granted 10% or 20%) goes to family AF1, while 20 30% (of the granted 10% or 20%) goes to family AF2.

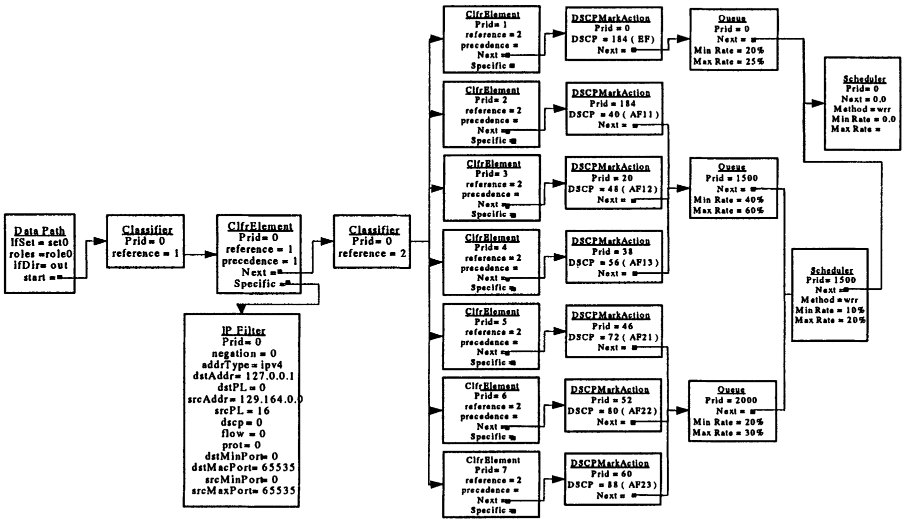


Figure 6. Policy D data path

5.5 Results

This subsection reports the measurement results obtained for the various policies described above. To compare policies we use, as ordering criteria, the number of the elements in the corresponding data path and their size of the policy (in bytes) when inserted into decision message sent by PDP (the fixed overhead of the COPS-PR protocol is not counted). The following table 2 shows the provisioned configurations, ordered according to the criteria just expressed.

Start up Time (Tstart up).

Table 3 shows the results for this measure. We have a mean time of 0.365 seconds, which is an acceptable time considered the poor performance of the computer and the length of the interaction. If we compare the mean value of the Tstart up we note a substantial independence of this time for the different types of provisioned policy. This is perhaps because the measured interaction between "start up PDP" and "start up PEP" is so long that it "masks" the dependency of the PDP time with the configuration.

Table 2. Policy element number and dimension

	Policy A		*Policy B*		*Policy C*		*Policy D*	
	Num	Byte	Num	byte	Num	Byte	Num	byte
DATA PATH	1	48	1	48	1	48	1	48
CLASSIFIER	2	56	2	56	2	56	2	56
CLASSIFIER ELEMENT	2	96	4	192	6	288	8	384
IP FILTER	2	144	4	288	6	432	8	576
DSCP MARKER	1	40	3	120	5	200	7	280
QUEUE	1	44	3	132	3	132	3	132
SCHEDULER	1	40	1	40	1	40	2	80
TOTAL	10	468	18	876	24	1196	31	1556

Table 3. Mean and Standard deviation

Policy	Mean (sec)	Standard deviation
A	0.371377	0.000658
B	0.357130	0.028821
C	0.370961	0.002597
D	0.361734	0.053850

PDP Decision Time (Tdecision)

Figure 7 shows the tendency of the mean PDP decision time with respect to the function of the number of PEPs that must be provisioned. The trend of this function are clearly linear with the number of PDP clients (i.e. PEPs). This function indicates a good architecture scalability with respect to the classical exponential trend of response time, especially for a high number of PEPs (devices) to configure. The decision time is variable with policy complexity in a sensible way. Higher complexity policies grow more quickly, but always linearly.

Total provisioning time (Tprovisioning)

The total provisioning time also has a linear trend with the number of the connected PEPs (figure 8). The provisioning time is also depends on the configuration considered. If we calculate the time difference between the function of figure 8 and that of figure 9 we obtain the function shown in figure 11. From the trend of this figure we can argue that the total provisioning time (Tprovisioning) is the sum of the total PDP decision time (Tdecision) plus a constant. It's interesting to note that the value of this constant term is slightly than Tdecision. In other words, the PDP time (Tdecision) is the much heavier component of the total provisioning time of a configuration (Tprovisioning). Thus the PDP is the bottleneck of the architecture.

Policy Information base update time, TPIB

The PIB updating time obviously depends on the DiffServ configuration that we considered. This time, however, is so short that it's negligible with respect to the

PDP time and its high variability is not visible. Figure 10 shows the mean updating time on the X axis while the empiric frequencies of the observed time are shown on the Y axis.

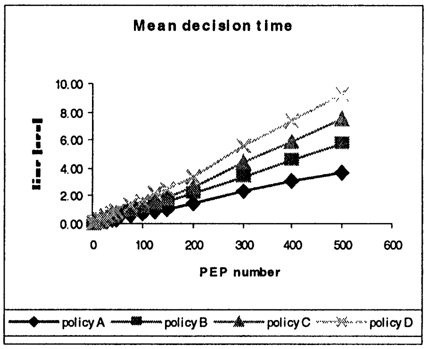


Figure 7. PDP decision time

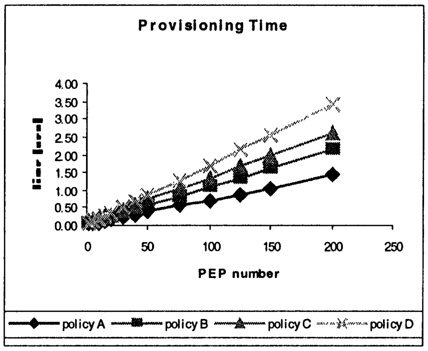


Figure 8. Total Provisioning time

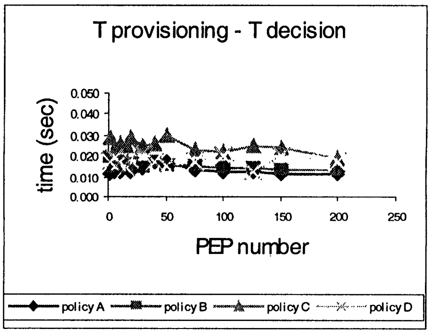


Figure 9. Provisining time minus decision time

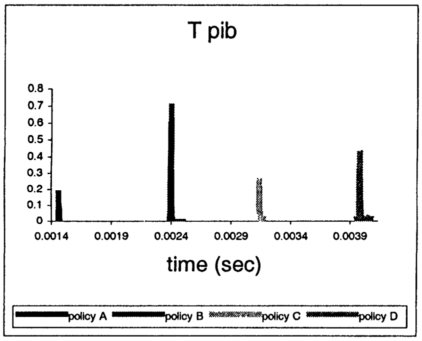


Figure 10. PIB installation time

6. CONCLUSIONS

In this paper we presented a range of COPS-PR measures, simulating the behavior of a policy based network, when acting in the configuration provisioning scenario. The prototypes are the "start up PDP" and the "decision PDP" and the client (PEP) developed as part of two different PIBs. We also proposed a simple way to translate a generic PRC into an object oriented language. The realized prototypes permit us to evaluate performance of COPS-PR protocol. The estimated times cover the two lower level of the architecture of a policy based network. In particular we have determined the linear trends of the PDP decision time for the provisioning of N PEP, underlining good architecture scalability and of our PDP prototypes, also for a big number of client (PEP). Further we have determined the bottleneck of the enquired architecture: the PDP. Finally, we have evaluated the impact of policy complexity on the overall processing time, pointing out that it is negligible in a start up scenario while it's influence is much greater in an "external event" scenario. It seems that the policy based approach can efficiently be used, in fact the policy interaction protocol has a linear growth with the number of PEPs, so the architecture and the protocols proposed by IETF are scalable. One more issue that can be further investigated is how simple to use are policies and how user friendly are policy based applications. In fact other specific technology as like as SNMP, CLI and SCRIPT, able to support "configuration management" were affected by usability problem.

7. REFERENCES

[1] J. Case, D. Harrington, R. Presuhna, B. Wijen, "Message Processing and Dispatching for the Simple Network Management Protocol", RFC 2572, April 1999.

[2] D. Durham, J. Boyle, R. Cohen, S. Herzog, R. Rajan, A. Sastry, "The COPS (Common Open Policy Service) Protocol", RFC 2748, January 2000.

[3] K. Chan, J. Seligson, D. Durham, S. Gai, K. McCloghrie, S. Herzog, F. Reichmeyer, R. Yavatkar, A. Smith, "COPS Usage for Policy Provisioning (COPS-PR)", RFC 3084, March 2001

[4] R. Wies, "Policy in Network and Systems Management – Formal Definition and Architecture-", Journal of Network and System Management, volume 2, number 1, March 1994

[5] B. Moore, E. Ellesson, J. Strassner, A. Westerinen, "Policy Core Information Model", RFC 3060, February 2001

[6] B. Moore, L. Rafalow, Y. Ramberg, Y. Snir, A. Westerinen, R. Chadha, M. Brunner, R. Cohen, J. Strassner, "Policy Core Information Model Extensions", internet-draft, November 2001

[7] M. Fine, K. McCloghrie, J. Seligson, K. Chan, S. Hahn, R. Sahita, A, Smith, F. Reichmeyer, "Framework Policy Information Base", internet draft, November 2001

[8] M. Fine, K. McCloghrie, J. Seligson, K. Cha, S. Hahn, C. Bell, A. Smith, F. Reichmeyer, " Differentiated services Quality of Service Policy Information Base", internet draft, November 2001

[9] Y. Bernet, K. McCloghrie, J. Seligson, K. Chan, S. Hahn, R. Sahita, A. Smith, "An informal management model for Diffserv routers", internet draft, February 2001.

[10] Vovida organization, www.vovida.org

[11] L. Lewis, "Policy-Based Configuration Management: A Perspective from a Network Management Vendor", The Simple Times, volume 2, number 1, September 2000

[12] J. Roese, "Configuration Management Services for the Large Enterprise Network", The Simple Times, volume 2, number 1, September 2000

[13] L. Lewis, "Implementing Policy in Enterprise Networks", IEEE Communications Magazine, January 1996.

[14] R. Yavatkar, D. Pendarakis, R. Guerin, "A framework for policy-based admission control", RFC 2753, January 2000.

[15] Y. Nomura, A. Chugo, M. Adachi, M. Toriumi, "A policy based Networking Architecture for Enterprise Networks", IEEE Internation Conference on Communications, 1999.

[16] M. Damianou, N. Dulay, E. Lupu, M. Slaman, T.Tonouchi, "Tools for Domain-based Policy Management of Distributed Systems" NOMS2002, April 2002.

DESIGN AND IMPLEMENTATION OF A POLICY-BASED RESOURCE MANAGEMENT ARCHITECTURE

Paris Flegkas, Panos Trimintzios, George Pavlou, Antonio Liotta
Centre for Communication Systems Research, School of Electronics, Computing and Mathematics, University of Surrey, Guildford, Surrey GU2 7XH, UK, {P.Flegkas, P.Trimintzios, G.Pavlou, A.Liotta}@eim.surrey.ac.uk}

Abstract: Policy-based Management can guide the behavior of a network or distributed system through high-level declarative directives that are dynamically introduced, checked for consistency, refined and evaluated, resulting typically in a series of low-level actions. We actually view policies as a means of extending the functionality of management systems dynamically, in conjunction with pre-existing "hard-wired" management logic. In this paper, we first discuss the policy management aspects of a resource management architecture for IP Differentiated Services networks and we focus on the functionality of the network dimensioning component. We then present a detailed description of the design and implementation of the components of the policy management sub-system needed to be deployed in order to make our system policy-driven. Finally, we present examples of network dimensioning policies describing their transformation from their definition by the operator until their enforcement.

Key words: Policy-based Management, IP Differentiated Services, Network Dimensioning, Resource Management

1. INTRODUCTION

For years the Internet networking community has been struggling to develop ways to manage networks. Initial attempts brought mechanisms and protocols that focused on managing and configuring individual networking devices i.e. the Simple Network Management Protocol (SNMP). This model worked well in early deployments of IP management systems for local and metropolitan area networks but now, with the evolution of Quality of Service (QoS) models such as the Differentiated Services (DiffServ) framework, the complexity and overhead of operating and administrating networks increases enormously. As such, it is very difficult to build management systems that can cope with the growing network size, complexity and multi-service operation requirements. There is also a need to be able to program management systems and network components to adapt to emerging requirements and subsequently to be able to dynamically change the behavior of the whole system to support modified or additional functionality. The emerging Policy-based Network Management paradigm claims to be a solution to these requirements.

Policy-based Management has been the subject of extensive research over the last decade [1]. Policies are seen as a way to guide the behavior of a network or distributed system through high-level, declarative directives. The IETF has been investigating policies as a means for managing IP-based multi-service networks, focusing more on the specification of protocols (e.g. COPS) and the object-oriented information models for representing policies. Inconsistencies in policy-based systems are quite likely since management logic is dynamically being added, changed or removed without the rigid analysis, design, implementation, testing and deployment cycle of "hard-wired" long-term logic. Conflict detection and resolution is required in order to avoid or recover from such inconsistencies.

In the next section, we discuss the policy management aspects of a resource management system for IP Differentiated Services networks; we then focus on the functionality of the dimensioning component in section 3 and in section 4, we present a detailed description of the design and implementation of the components of the policy management sub-system needed to be deployed in order to make our system policy-driven. Finally, we present examples of network dimensioning policies, describing their transformation from their definition by the operator until their enforcement.

2. SYSTEM ARCHITECTURE

We have designed a system for supporting QoS in IP DiffServ Networks in the context of the European collaborative research project TEQUILA (Traffic Engineering for QUality of service in the Internet at LArge scale). This architecture can be seen as a detailed decomposition of the concept of an extended Bandwidth Broker (BB) realized as a hierarchical, logically and physically distributed system. A detailed description can be found in [2]. A classification of the policies applied to this system was presented in [3]. In Figure 1 we present only the resource management part of the architecture together with the components of the policy management sub-system i.e. Policy Management Tool, Policy Repository and Policy Consumer needed to make the system extensible through policies.

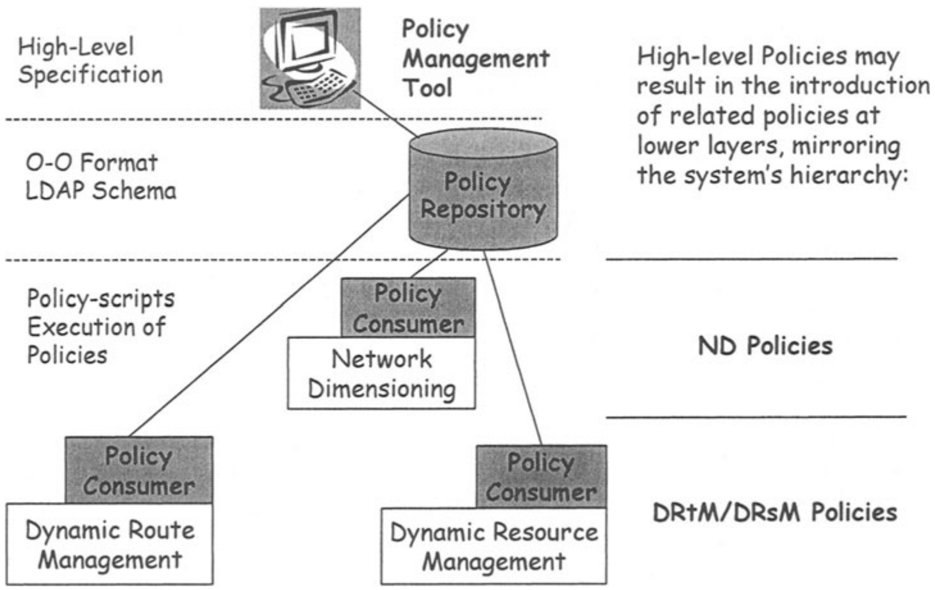


Figure 1. Policy-driven Resource Management System

The Network Dimensioning (ND) component is responsible for mapping traffic requirements to the physical network resources and for providing Network Dimensioning directives in order to accommodate the predicted traffic demands. We describe the functionality and algorithms of Network Dimensioning in more detail in Section 3. The lower level of the system intends to manage the resources allocated by Network Dimensioning during the system operation in real-time, in order to react to statistical traffic fluctuations and special arising conditions. This part is realized by the Dynamic Route (DRtM) and Dynamic Resource Management (DRsM), which both monitor the network resources and act to medium to short term fluctuations. DRtM operates at the edge nodes and is responsible for managing the routing processes in the network. It mainly influences the parameters based on which the selection of one of the established MPLS Labeled Switched Paths (LSPs) is effected at an edge node with the purpose of load balancing. An instance of DRsM operates at each router and aims to ensure that link capacity is appropriately distributed among the PHBs in that link.

A single Policy Management Tool exists for providing a policy creation environment to the administrator where policies are defined in a high-level declarative language and after validation and static conflict detection tests, they are translated into object-oriented (O-O) representation (information objects) and stored in a repository. The Policy Repository is a logically centralized component but may be physically distributed since the technology for implementing this component is the LDAP (Lightweight Directory Access Protocol) Directory. After the policies are stored, activation information may be passed to the responsible Policy Consumer in order to retrieve and enforce them. The Policy Consumer can be seen as a collocated Policy Decision Point (PDP) and Policy Enforcement Point (PEP) with regards to the IETF Policy Framework [4]. A detailed description of the design and implementation of these components is presented in Section 4.

In Figure 1 the representation of the policies at every level of the framework is also depicted showing that every policy is going through two stages of translation/refinement in its life-cycle in order to be enforced: first from the high-level specification to an object-oriented format (LDAP objects) and second from the LDAP objects to a script that is interpreted on the fly, complementing this way conceptually the management intelligence of the above layer in the hierarchy. For example, a policy enforced on the DRsM component is actually enhanced management logic that conceptually belongs to the ND layer of our system. Although policies may be introduced at every layer of our system, higher-level policies may possibly result in the introduction of related policies at lower levels, forming a policy hierarchy mirroring the management system's hierarchy. This means that a policy applied to a hierarchical system might pass through another stage of translation/refinement that will generate the policies that are enforced in the lower levels of the system. It is questionable if the automation of this process is feasible without human intervention. A more detailed discussion on policy-based hierarchical management systems can be found in [5].

In the next section, we present the algorithm by which the Network Dimensioning component calculates the configuration of the network in order for the reader to understand the examples of policies enforced on this component presented in the following sections.

3. NETWORK DIMENSIONING ALGORITHM

ND performs the provisioning activities of the system. It is responsible for the long to medium term configuration of the network resources. By configuration we mean the definition of LSPs as well as the anticipated loading for each PHB on all interfaces, which are subsequently being translated by DRsM into the appropriate scheduling parameters (e.g. priority, weight, rate limits) of the underlying PHB implementation. ND does not provide absolute values but they are in the form of ranges, constituting directives for the function of the PHBs, while for LSPs they are in the form of multiple paths to enable multi-path load balancing. The exact PHB configuration values and the load distribution on the multiple paths are determined by DRsM and DRtM respectively, based on the state of the network, but should always adhere to ND directives.

ND runs periodically, first requesting the predictions for the expected traffic per Ordered Aggregate [6] (OA) in order to be able to compute the provisioning directives. The dimensioning period is in the time scale of a week while the forecasting period is in the time scale of hours. The latter is a period in which we have considerably different predictions as a result of the time schedule of the subscribed Service Level Specifications (SLSs). For example, ND might run every Sunday evening and provide multiple configurations i.e. one for each period of each day of the week (morning, evening, night).

The objectives are both traffic and resource-oriented. The former relate to the obligation towards customers, through the SLSs. These obligations induce a number of restrictions about the treatment of traffic. The resource-oriented objectives are related to the network operation, more specifically they are results of the high-level business policy that dictates the network should be used in an optimal manner. The basic Network Dimensioning functionality is summarized in Table 1.

Table 1. Network Dimensioning Algorithm Overview

Input:
Network topology, link properties (capacity, propagation delay, PHBs)
Pre-processing:
Request traffic forecast, i.e. the potential traffic trunks (TT)
Obtain statistics for the performance of each PHB at each link
Determine the maximum allowable hop count K per TT according to the PHB statistics
Optimisation phase:
Start with an initial allocation (e.g. using the shortest path for each TT)
Iteratively improve the solution such that for each TT find a set of paths:
The minimum bandwidth requirements of the TT are met
The hop-count constraints K is met (delay/ loss requirements are met)
The overall cost function is minimized
Post-processing:
Allocate any extra capacity to the resulted paths of each OA according to resource allocation policies
Sum the path requirements per link per OA, give minimum (optimisation phase) and maximum (post-processing phase) allocation directives to DRsM
Give the appropriate paths calculated in the optimisation phase to DRtM
Store the configuration into the Network Repository

The network is modeled as a directed graph $G = (V,E)$, where V is a set of nodes and E a set of links. With each link $l \in E$ we associate the following parameters: the link physical capacity C_l, the link propagation delay d_l^{prop}, the set of the physical queues K, i.e. Ordered Aggregates (OAs), supported by the link. For each OA, $k \in K$ we associate a bound d_l^k (deterministic or probabilistic depending on the OA) on the maximum delay incurred by traffic entering link l and belonging to the $k \in K$, and a loss probability p_l^k of the same traffic.

The basic traffic model of ND is the traffic trunk (TT). A traffic trunk is an aggregation of a set of traffic flows characterized by similar edge-to-edge performance requirements [7]. Also, each traffic trunk is associated with one ingress node and one egress node, and is unidirectional. The set of all traffic trunks is denoted by T.

The *primary objective* of such an allocation is to ensure that the requirements of each traffic trunk are met as long as the traffic carried by each trunk is at its specified minimum bandwidth. However, with the possible exception of heavily loaded conditions, there will generally be multiple feasible solutions. The design objectives are further refined to incorporate other requirements such as: a) avoid overloading parts of the network while other parts are under loaded, b) provide overall low network load (cost).

The last two requirements do not lead to the same optimization objective. In any case, in order to make the last two requirements more concrete, the notion of "load" has to be quantified. In general, the load (or cost) on a given link is an increasing function of the amount of traffic the link carries. This function may refer to link utilization or may express an average delay, or loss probability on the link. Let x_l^k denote the capacity demand for OA $k \in K$ satisfied by link l and $u_l^k = x_l^k / C_l$ the link utilisation. Then the link cost induced by the load on OA $k \in K$ is a convex function, $f_l^k(u_l^k)$, increasing in u_l^k. The total cost per link is defined as $F_l(\overline{u}_l) = \sum_{k \in K} f_l^k(u_l^k)$, where $\overline{u}_l = \{u_l^k\}_{k \in K}$ is the vector of demands for all OAs of link l. The total cost per link is an approximate function, e.g. $f_l^k(u_l^k) = a_l^k u_l^k$.

We provide an objective that compromises between the two a) and b), that is avoid overloading parts of the network and minimize overall network cost:

$$\text{minimize} \ \sum_{l \in E}(F_l(\overline{u}_l))^n = \sum_{l \in E}\left(\sum_{k \in K} f_l^k(u_l^k)\right)^n, \quad n \geq 1 \quad (1)$$

When $n = 1$, the objective (1) reduces to objective b), while when $n \to \infty$ it reduces to a).

Each traffic trunk is associated with an end-to-end delay and loss probability constraint of the traffic belonging to the trunk. Hence, the trunk routes must be designed so that these two constraints are satisfied. Both the constraints above are constraints on additive path costs under specific link costs. However the problem of finding routes satisfying these constraints is, in general, NP-complete [8]. Given that this is only part of the problem we need to address, the problem in its generality is rather complex.

Usually, loss probabilities and delay for the same PHB on different nodes are of similar order. We simplify the optimization problem by transforming the delay and loss requirements into constraints for the maximum hop count for each traffic trunk (TT). This transformation is possible by keeping statistics for the delay and loss rate of the PHBs per link, and by using the maximum, average or n-th quantile in order to derive the maximum hop count constraint.

For each traffic trunk $t \in T$ we denote as R_t the set of (explicit) routes defined to serve this trunk. For each $r_t \in R_t$ we denote as b_{r_t} the capacity we have assigned to this explicit route. We seek to minimize (1), such that the hop-count constraints are met and the bandwidth of the explicit routes per traffic trunk should be equal to the trunks' capacity requirements.

This is a network flow problem and considering the non-linear formulation, for the solution we use the general gradient projection method [9]. This is an iterative method, where we start from an initial feasible solution, and at each step we find the minimum first derivative of the cost function path and we shift part of the flow from the other paths to the new path, so that we improve our objective function (1). If the path flow becomes negative, the path flow simply becomes zero. This method is based on the classic unconstraint non-linear optimization theory, and the general point is that we try to decrease the cost function through incremental changes in the path flows. A more detailed description of the algorithm is presented in [10].

4. DESIGN AND IMPLEMENTATION OF THE POLICY COMPONENTS

In the following sections, we present and describe the design and implementation of the policy sub-system components. First in the Policy Management Tool section, the description of policy definition language as well as the capabilities of the graphical interface are presented; we then focus on the Policy Repository where the most important issue is how one models policies in an O-O format and then on the Policy Consumer where policies are translated to scripts and executed on the fly. Examples of policy rules are also presented demonstrating the different way of representation of the rules at every stage of their life cycle i.e. from high-level directives to LDAP objects and finally to interpreted scripts realising the new management logic added through these policies.

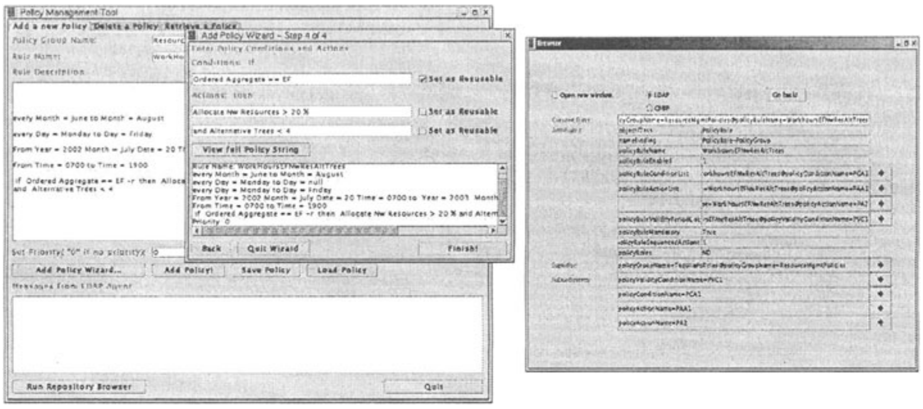
4.1 Policy Management Tool

A high-level definition language has been designed and implemented that provides to the administrator the ability to add, retrieve and update policies in the Policy Repository. The administrator enters a high-level specification of the policy, which is then passed to a translation function that maps this format to entries in an LDAP Directory realizing the Policy Repository (see next section) through LDAP *add* operations, according to an LDAP schema of our information model; the latter has been produced following the guidelines described in [11]. The format of a policy rule specification is shown below:

[Policy ID] [Group ID] [time period condition] **[if** {condition [and] [or]}] **then** {action [and]}

The first two fields define the name of the policy rule and the group that this policy belongs to so that the generated LDAP entries should be placed under the correct policy group entry. The time period condition field specifies the period that the policy rule is valid and supports a range of calendar dates, masks of days, months as well as range of times. The following {if then} clause represents the actual policy rule where the condition and action fields are based on the information model described earlier in this section. Compound Policy Conditions are also supported both in the Disjunctive Normal Form (DNF) (an ORed set of ANDed conditions) and in the Conjunctive Normal Form (CNF) (an ANDed set of ORed conditions) as well as Compound Policy Actions representing a sequence of actions to be applied. Our implementation also caters for the notion of rule-specific and reusable conditions and actions in a way that every time a new policy rule is added, it first checks if its conditions and actions are already stored in the repository as reusable entries. If such entries exist, an entry is added with a DN pointer to the reusable entry under the policy rule object while if not they are treated as rule-specific, placing the condition entry below the policy rule entry.

A compiler has been implemented in order to parse and translate the policy rules specified with the above syntax using SableCC [12]. This is an object-oriented framework that generates compilers by building a strictly-typed abstract syntax tree that matches the grammar of the language, automatically generates tree-walker classes and enables the implementation of the actions on the nodes of the tree using inheritance. Translation classes have been implemented that map the entered policy rules to LDAP *add* operations, according to an LDAP schema of our information model (see next section). Notifications to the corresponding Policy Consumers are also sent every time a policy rule is successfully added to the repository. The role attribute of every policy rule is used to distinguish which policy consumer to notify since this attribute represents the properties of the PEPs that each PDP manages according to Policy Core Information Model (PCIM)[13].



Policy Group Name: Resourc
Rule Name: WorkHo
Rule Description

every Month = June to Month = August
every Day = Monday to Day = Friday
From Year = 2002 Month = July Date = 20 T
From Time = 0700 to Time = 1900
if Ordered Aggregate == EF -r then Alloca
and Alternative Trees < 4

Set Priority("0" if no priority): 0
Messages from LDAP Agent

Enter Policy Conditions and Actions
Conditions: if
Ordered Aggregate == EF [✓ Set as Reusable]
Actions: then
Allocate Nw Resources > 20 % [ ] Set as Reusable
and Alternative Trees < 4 [ ] Set as Reusable

```
Rule Name: WorkHoursIfNwRetAltTrees
every Month = June to Month = August
every Day = Monday to Day = null
every Day = Monday to Day = Friday
From Year = 2002 Month = July Date = 20 Time = 0700 to Year = 2003 Month
From Time = 0700 to Time = 1900
if Ordered Aggregate == EF -r then Allocate Nw Resources > 20 % and Altern
Priority 0
```

*Figure 2.*a) Policy Management Tool Screenshot b) LDAP Browser Instance

A Graphical User Interface (GUI) has been implemented in order for the administrator to manage policies that are entered in the system (Fig. 2). It provides capabilities to add new policies, retrieve or delete existing policies. The addition of the policies can either be done directly by writing the whole policy rule in the format described previously or follow a wizard that guides the administrator through the process of creating and entering a new policy to the system step by step. A directory browser has also been implemented and integrated with the tool that enables the operator to browse the information stored in a tree-structured repository like the LDAP Directory. In the browser's design, the idea of a current entry was adopted for displaying the contents of a tree. Each browser instance has one current entry, which is a selected tree entry. Its distinguished name (DN), its attributes' types and values, its superior entry's distinguished name and its subordinate entries' relative distinguished names (RDNs) are displayed. The current entry's position in the tree can be identified by the current entry's distinguished name. The user may select any of the current entry's subordinate entries or its superior entry and make it the browser's current entry. This way, one can move up and down the hierarchy of the Directory Information Tree (DIT) that is accessed. If an attribute's entry is a pointer to another entry, the user can make it the current entry, in which case they still keep the option to go back. In addition, the user is allowed to fill in the DN of the entry to be displayed in the Current Entry field.

In order to demonstrate the results of the enforcement of policies we used a 10-node (nodes 0-9) 36-link random topology and a traffic load of 70 % of the total throughput of the network. Our first example (P1) concerns a policy rule that wants to create an explicit LSP following the nodes 4, 9, 7, 6 with the bandwidth of the TT being 2 Mbps that is associated with this LSP. The policy rule is entered with the following syntax:

```
If OA==EF and Ingress==4 and Egress==6 then Setup LSP 4-9-7-6 2
Mbps (P1)
```

The second example (P2) of a policy rule concerns the effect of the cost function exponent in the capacity allocation of the network. As we mentioned earlier by increasing the cost function exponent, the optimisation objective that avoids overloading parts of the network is favoured. So, if the administrator would like to keep the load of every link below a certain point then he/she should enter the following policy rule in our system using again our policy notation:

```
If maxLinkLoad > 80% then Increase Exponent by 1          (P2)
```

4.2 Policy Repository

An object-oriented information model has been designed for representing the network dimensioning policies, based on the IETF Policy Core Information Model (PCIM) and its extensions (PCIMe) specified in [13] and [14] respectively. One of the major objectives of such information models is to bridge the gap between the human policy administrator who enters the policies and the actual enforcement commands executed at the component in order to realize the business goal of the administrator. Another goal is to facilitate interoperability among different systems so that policy consumers that belong to different systems understand the same semantics of policy and they have a mutual knowledge of how policies are stored in the policy repository despite the fact that each policy consumer might interpret it differently. IETF has described a QoS Policy Information Model [15], representing QoS policies that result in configuring network elements to enforce the policies, while our information model describes policies that are applied at a higher level (Network Management Level). Some of these policies may possibly be refined into lower-level policies mirroring our architecture's hierarchy and finally result into policies configuring the Network Elements.

In our information model that represents Network Dimensioning policies that can be enforced in our system, most of the conditions are modeled by using the class SimplePolicyCondition with instances of the variable and value classes (IF <variable> matches <value>). Some of the actions are modeled by defining classes derived from the PolicyAction abstract class while others are modeled by using the class SimplePolicyAction with the appropriate aggregations, using instances of the variable and value classes ("SET <variable> TO <value>"). For example, the maximum number of alternative trees that the ND algorithm should calculate for every TT is represented by a pair of maxAltTree variable and IntegerValue classes as well as the definition of the constant used in the link cost function. In [3] the policyAction class hierarchy is described in more detail where the DimensioningPeriodAction class models the policy action that sets the period that ND is calculating a new configuration, while the NwBwAllocationAction and LinkBwAllocationAction classes represent the actions that indicate the amount of bandwidth to be allocated to every OA (depending on the policy condition) at a network wide level and in every link respectively. The SpareBwTreatmentAction and OverBwTreatment Action classes represent the policy actions that drive the post-processing stage of ND as explained in the previous section. The SetLSPAction class models the setup of an LSP that is defined by policy and the HopCountDerivationAction class represents the action that influences the way the derivation of the delay and loss requirements to an upper bound of number of hops is done.

In Figure 3 the policy rule P1 presented as an example in the previous section is depicted modeled according to PCIM and PCIMe. As it can be seen, it comprises a compound policy condition which represents a combination of 3 simple policy conditions in Disjunctive Normal Form (ORs of ANDs) each of them belonging to the same Group (GroupNumber =1) and a Policy Action represented by the SetLSPAction class derived form the PolicyAction abstract class defined in PCIM. The first simple policy condition uses an OA variable which takes integer values from 1 to 4 (EF is 1, AF1x is 2, etc), the second and third simple policy conditions use an Ingress node and Egress node variables which take integer values form 0 to 9 for our network topology (the egress node simple policy condition is not shown in Figure 3 for illustrative purposes). The aggregations used in order to define in order to define this rule are also depicted as defined in PCIMe.

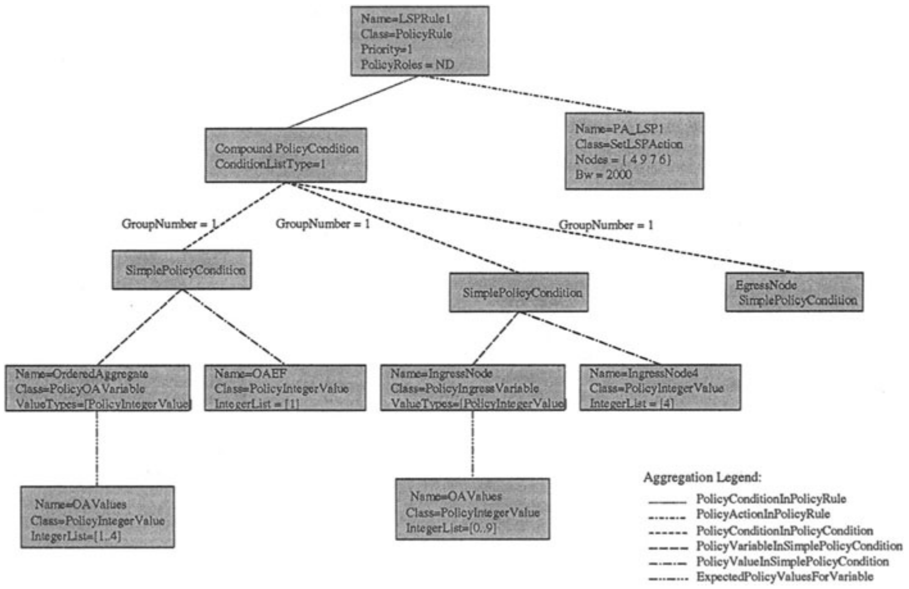


Figure 3. Policy Rule P1 according toPCIM/PCIMe

Using the same methodology, policy rule P2 is modelled according to PCIM/PCIMe using a Simple Policy Condition and ExponentAction class derived from the Policy Action abstract class. These classes are mapped to structural and auxiliary classes as defined in [11] in order to be stored to an LDAP Directory, which realizes our Policy Repository.

Since our system described in Section 2 is a large scale distributed system, it is valid to consider CORBA as the technology to support the remote interactions between the components. This was the key motivation for mapping the LDAP functionality to CORBA realizing the Policy Repository as an LDAP Directory offering a CORBA IDL interface, identical to the LDAP specifications [16], to the rest of the components. The following Code 1 caption shows the part of the specification of an LDAP server in Interface Definition Language (IDL) providing a LDAP search operation to every LDAP client in our system. Note that the CORBA implementation of the LDAP search operation returns the result in a single message while the LDAP protocol returns the multiple matching entries in a series of messages, one for each entry. The results are terminated with a result message, which contains an overall result for the search operation.

```
//...
typedef string LDAPDN_t;
enum Scope_t {
     sc_baseObject,
     sc_singleLevel,
     sc_wholeSubtree
  };
typedef string Filter_t; // filter for this implementation
struct SearchResultEntry_t_struct {
     LDAPDN_t          objectName;
     AttributeList_t   attributes;
  };
typedef SearchResultEntry_t_struct SearchResultEntry_t;
typedef sequence<SearchResultEntry_t> SearchResultEntryList_t;
interface LDAPServer {
  void Search (
     in LDAPDN_t               baseObject,
     in Scope_t                scope,
     in Filter_t               filter,
     in AttributeDescriptionList_t attributeTypes,
     out SearchResultEntryList_t   searchResultList
  ) raises (noSuchObject, invalidDNSyntax, invalidFilterSyntax, generalError);
//...
}; // interface LDAPServer
```

Code 1. LDAP Search operation in IDL

4.3 Policy Consumer

Policy Consumers may be considered as the most critical components of the policy management framework since they are responsible for enforcing the policies on the fly while the system is running. The key aspect of policies apart from their high-level declarative nature is that they can also be seen as a vehicle for "late binding" functionality to management systems, allowing for their graceful evolution as requirements change. So, a policy capable system should provide the flexibility to add, change or remove management intelligence while according to traditional management models, management logic is of static nature, parameterized only through Managed Objects (MOs) attributes and actions. In order to achieve such functionality, a policy is eventually translated to a script "evaluated" by an interpreter with actions resulting in management operations. In [3] a detailed design and decomposition of the policy consumer was presented where every policy consumer comprises a Repository Client which retrieves all the LDAP objects associated with a policy rule, a script generator which is responsible for creating the script that implements the policy, and a policy interpreter which provides the "glue" between the policy consumer and the policy-based component and interprets the script, which includes functions that perform management operations. In the following paragraphs, we present how the policy rule examples P1 and P2 are translated and enforced by the Policy Consumer.

After the P1 rule is correctly translated and stored in the repository, the Policy Management Tool notifies the Policy Consumer associated with ND that a new policy rule is added in the repository, which then goes and retrieves all the associated objects with this policy rule. From the policy objects the consumer generates code that is interpreted and executed on the fly representing the logic added in our system by the new policy rule. In our implementation, we have chosen TCL as the scripting language due to the ease with which it interfaces with C, since the ND component is implemented in C. The pseudo code of how the above policy is realised by the Policy Consumer is shown in caption Code 2.

```
$TT_{OA}$: the set of TTs belonging to OA
For each $tt_i \in TT_{OA}$ we get the following:
  $v_{ingress}, v_{egress}$ : ingress, egress nodes
  $b(tt_i)$: bandwidth requirement of $tt_i$
for each $tt_i \in TT_{EF}$ do
  if (($v_{ingress}$ == 4) and ($v_{egress}$ == 6))
    add_LSP ("4-9-7-6", 2000)
    $b(tt_i) = b(tt_i) - 2000$
  Else
    Policy not executed – TT not found
```

Code 2. Pseudocode produced for enforcing (P1)

As it can be seen from the above pseudo-code, it first searches for a TT in the traffic matrix that matches the criteria specified in the conditions of the policy rule regarding the OA, the ingress and egress node. If a TT is found then it executes the action that creates an LSP with the parameters specified and subtracts the bandwidth requirement of the new LSP from the TT in the traffic matrix file so that the ND algorithm will run for the remaining resources. Note that if the administrator had in mind a particular customer for this LSP then this policy should be refined into a lower level policy enforced on the DRtM component, mapping the address of this customer onto the LSP.

The same procedure explained in the previous example is followed again and the policy consumer enforces this policy by generating a script, which is shown in Caption Code 3.

```
maxLinkLoad: the maximum link load utilisation
after the end of the optimisation algorithm
n: the cost function exponent (initially = 1)
Optimisation_algorithm n
while (maxLinkLoad > 80 )
    n = n+1
    optimisation_algorithm n
```

Code 3. Pseudocode produced for enforcing (P2)

As it can be observed from Figure 4, the enforcement of the policy rule caused the optimization algorithm to run for 4 times until the maximum link load utilisation at the final step drops below 80%. The exponent value that achieved the policy objective was $n = 4$. There might be cases that rules like the one above will cause infinite recursion when the algorithm cannot drop the maximum link load below a certain threshold, so a maximum number of iterations should be defined to avoid these kind of problems.

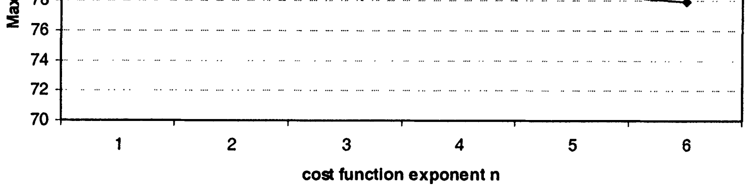


Figure 4. Effect of the cost function exponent on the maximum link load utilization

For the purpose of demonstrating the effects of the enforcement of policies in our system we implemented a TE-GUI shown in Fig. 5. It depicts the topology of the network that the ND component is calculating a new configuration. The GUI draws the links of the topology with different colours according to load utilisation and all the LSPs for every OA created. It has also the capabilities to display overall statistics for the load distribution for every link per OA as well as statistics for every step of the ND algorithm i.e. average link utilisation, link load standard deviation, max link load, running time etc. In the following figure, two snapshots of the TE-GUI are depicted one before and one after the enforcement of the above policies. As it can be seen, the enforcement of the policies caused the link load to fall under 80 % (before the enforcement of policies the link 5->6 was loaded over 90%) as well as the LSP created by the P1 is also drawn.

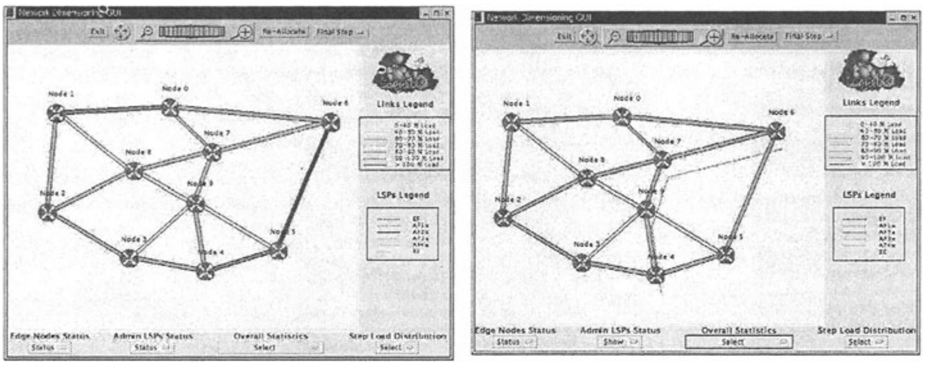


Figure 5. TE-GUI snapshots (a) before and (b) after the enforcement of policies P1 and P2

5. CONCLUSIONS

While most of the work on policies has focused in specifying rules for configuring network elements, our work addresses issues for defining higher level (network-wide) policies that apply to a hierarchical distributed management system. We view policies as a means for enhancing or modifying the functionality of policy influenced components reflecting high-level business decisions. When designing a policy-based system, it is very important to identify the parameters that are influenced by policies resulting in driving the behavior of a system to realize the administrator's business goals. This decision should take into account the inconsistencies caused by the coexistence of policies with "hard-wired" functionality.

In this paper, we presented a policy-driven resource management system and described the components of such a system, focusing on Network Dimensioning. We then presented a detailed description of the design and implementation of the components of the policy sub-system needed to be deployed in order to make our system policy-driven and finally, examples of network dimensioning policies are presented describing their transformation from their definition by the operator until their enforcement.

As a continuation of the work described in this paper, we will be focusing on defining policies for the rest of the components of the TE system and explore the issue of the refinement of policies entered at the Network Dimensioning to lower level policies that apply to Dynamic Resource and Route Management components forming a policy hierarchy. Also we intend to look at the specification of conflict detection and resolution mechanisms specific to our problem domain.

REFERENCES

[1] M. Sloman. Policy Driven Management For Distributed Systems, Journal of Network and Systems Management, Vol. 2, No. 4, pp. 333-360, Plenum Publishing, December 1994.

[2] P. Trimintzios et al.. A Management and Control Architecture for Providing IP Differentiated Services in MPLS-based Networks, IEEE Communications, special issue in IP Operations and Management, Vol. 39, No. 5, pp. 80-88, IEEE, May 2001.

[3] P. Flegkas, P. Trimintzios, G. Pavlou. A Policy-based Quality of Service Management Architecture for IP DiffServ Networks, IEEE Network Magazine, special issue on Policy Based Networking, vol. 16, no. 2, pp. 50-56, March/April 2002.

[4] R. Yavatkar, D. Pendarakis, R. Guerin. A Framework for Policy Based Admission Control, Informational RFC 2753, January 2000.

[5] P. Flegkas et al.. On Policy-based Extensible Hierarchical Network Management in QoS-enabled IP Networks, Proceedings of the IEEE Workshop on Policies for Distributed Systems and Networks (Policy '01), Bristol, UK, M. Sloman, J. Lobo, E. Lupu, eds., pp. 230-246, Springer, January 2001.

[6] D. Grossman. New Terminology and Clarifications for DiffServ, IETF IETF Informational RFC 3260, April, 2002.

[7] T. Li, and Y. Rekhter. Provider Architecture for Differentiated Services and Traffic Engineering (PASTE) IETF Informational RFC-2430, October 1998.

[8] Z. Wang, and J. Crowcroft. Quality of Service Routing for Supporting Multimedia Applications, IEEE Journal of Selected Areas in Communications, vol. 14, no. 7, pp. 1228-1234, September 1996.

[9] D. Bertsekas. Nonlinear Programming, (2nd ed.) Athena Scientific, 1999.

[10] P. Trimintzios et al.. Quality of Service Provisioning through Traffic Engineering with Applicability to IP-based Production Networks, to appear in Computer Communications, special issue on Performance Evaluation of IP Networks and Services, Elsevier Science Publishers, Vol. 26, No. 8, 2003.

[11] J. Strassner et al.. Policy Core LDAP Schema, draft-ietf-policy-core-schema-14.txt, January 2002.

[12] E. Gagnon. SableCC, An Object-Oriented Compiler Framework, Master of Science, School of Computer Science, McGill University, Montreal.

[13] B. Moore et al.. Policy Core Information Model – Version 1 Specification, IETF RFC-3060, February 2001.

[14] B. Moore et al.. Policy Core Information Model Extensions, draft-ietf-policy-pcim-ext-08.txt, May 2002.

[15] Y. Snir et al.. Policy QoS Information Model, draft-ietf-policy-qos-info-model-04.txt, November 2001.

[16] M. Wahl, T. Howes, S. Kille. Lightweight Directory Access Protocol (v3), IETF RFC-2251, Decemeber 1997.

BANDS: AN INTER-DOMAIN INTERNET SECURITY POLICY MANAGEMENT SYSTEM FOR IPSEC/VPN

Yanyan Yang[1], Zhi (Judy) Fu[2], and S. Felix Wu[1]
[1]Department of Computer Science, University of California, Davis, CA 95616, USA, {yyyang,sfwu}@ucdavis.edu, [2]Network and Infrastructure Research Lab (NIRL), Motorola Labs, USA, jfu@labs.mot.com

Abstract: IPSec/VPN is widely deployed for users to remotely access their corporate data. IPSec policies must be correctly set up for VPN to provide anticipated protection. Manual policy setup is unscalable, inefficient and error-prone. Automated policy generation to comply with and enforce high-level security policies is desired but difficult, especially in an inter-domain environment when a VPN traverse multiple domains. This paper presents a distributed framework and protocol, BANDS, for inter-domain policy negotiation and generation. The BANDS architecture consists of two phases: AS (Autonomous System) route path discovery and an inter-domain collaborative protocol for policy negotiation among the autonomous systems discovered in the first phase. Each AS conceptually has one security requirement server responsible for the task of inter-domain policy negotiation. Following this two-step process in BANDS, a set of distributed security policies (for the implementation of policy enforcement) will be automatically negotiated/generated based on decentralized and predefined security requirements.

Key words: Inter-domain Security Management, Security Policy Management, IPSec/VPN

1. INTRODUCTION

1.1 Security Management for a Remote/Mobile Layer-3 Network Node

We had encountered various difficulties when we plugged in our laptop computers in a foreign domain during a trip, as shown in Figure 1. We did not know the security policies along the route path from our laptops (then being connected to a remote layer-3 network) or we did not even know what the route path was. In our home layer-3 network, our own laptops might be protected by an intrusion detection system as well as some other security counter measures, but, in a foreign domain, we in general know very little. When our traffic caused conflicts with "some" security policies along the route path, we did not know what the root cause was (or even target security gateway). Maybe our destination server was down during that time. Or, possibly, our SSH requests to certain restricted sites have been dropped by a particular IPSec/Firewall/NAT gateway on the way. In [7], we have discussed how such conflicts can occur in the context of IPSec/VPN and firewall.

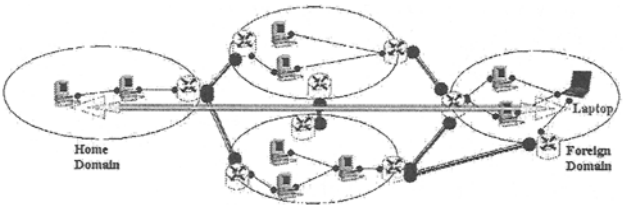


Figure 1. Security management for a remote network node

What we really want is a plug-and-play Internet security management solution. When we plug in a computer, a laptop, or any layer-3 addressable node under some Internet/VPN access points, within a couple minutes or shorter, our connections to some corresponding destinations will be established correctly and securely.

One short-term solution to this problem is to establish a bi-directional secure virtual tunnel from the remote laptop to a "home agent" (similar to MobileIP) in the home layer-3 network. This tunnel may be a L2VPN, a remote IPSec tunnel, a L2TP tunnel, a SSH tunnel, or their combination. This approach has two major shortcomings. First, the traffic will go through sub-optimal route paths, as they must travel through the home network in both directions. Second, with this approach, the traffic from the remote laptop will be properly protected by the home network security gateways because the security gateways will treat them as "trusted home traffic". But, if the laptop (even if it is at home) wants to connect to a "new" corresponding destination under a different administrative domain, it is not clear whether some new "policy conflicts" will arise as we don't know the security policies at the other end of the communication at that particular moment.

1.2 IPSec/VPN Security Policy Management

Internet has been more and more "dynamic" in many ways. In order to provide secure communications for end-to-end connections, IPSec (Internet Security Protocol Suite) [10] policies are widely deployed in firewalls/gateways to restrict access or selectively enforce security operations. Currently, most commercial IPSec implementations require a manual policy configuration process, which is inefficient and error-prone. Some IPSec management products (such as CISCO or NETSCREEN) provide a policy distribution system to allow a centralized policy server to distribute policies to all the IPSec devices in the same administrative domain as shown in Figure 2. However, for all the venders we have talked to, none of their products can provide any "correctness" assurance [7] about the IPSec/VPN policy rules stored in the policy repository in the first place. In other words, currently, even in an intra-domain environment, commercial IPSec/VPN policy tools cannot generate or validate a set of provably correct policy rules.

For the purpose of security policy formal analysis, in [1][7], we proposed a separation principle between security policy and requirement. Based on this principle, system administrators can unambiguously specify each individual security model (or individual security requirement) in our policy language. Then, the collection of such security requirements can be formally and efficiently analyzed for its correctness and completeness properties. Furthermore, a set of IPSec security policy rules will be automatically produced for all the PDP (Policy Decision Point) / PEP (Policy Enforcement Point) devices.

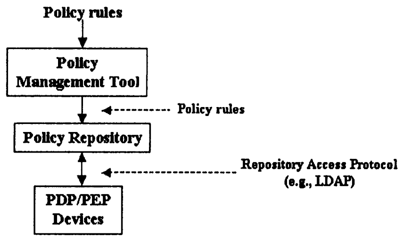


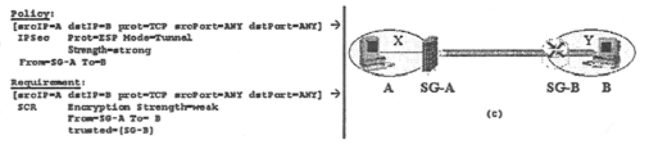
Figure 2. A sample policy distribution system

While our earlier work [1] provides a rigorous framework for security policy management in an intra-domain environment, we do not yet have a good solution for securing end-to-end connections across multiple administrative domains. Therefore, the key contribution in this paper is a distributed and yet scaleable architecture and protocol for inter-domain IPSec/VPN security policy management. The rest of the paper is structured as follows. Section 2 defines the problem of inter-domain security policy management. In Section 3, we briefly review the related work. Then, we present how to solve our target problem as well as different components under our solution in Section 4. Section 5 gives an example scenario to illustrate the cooperation and interoperation of each component under the BANDS architecture. We will also present some preliminary performance evaluation results in Section 6. Finally, in Section 7, we summarize the paper and outline some future works.

2. TERMINOLOGY AND PROBLEM DEFINITION

2.1 Security Policy versus Requirement

Traditionally there is no rigorous definition of security requirements and security policies. As a result, the relationship between them is so vague that the correctness of security policies cannot be formally and automatically substantiated. The needs to distinguish high-level security requirements and low-level policies were addressed in [4][5]. Once the high-level requirements are specified/modified, it should be possible to determine what kind of low-level policies must be created/changed.



```
    Policy:
    [srcIP=A dstIP=B prot=TCP srcPort=ANY dstPort=ANY] →
(a)  IPSec   Prot=ESP Mode=Tunnel
             Strength=strong
     From=SG-A To=B

    Requirement:
    [srcIP=A dstIP=B prot=TCP srcPort=ANY dstPort=ANY] →
(b)  SCR     Encryption Strength=weak
             From=SG-A To= B
             trusted={SG-B}
```

Figure 3. An example of policy (a), requirement (b) under certain network topology (c)

Therefore, under the BANDS architecture, a two-level security requirement/policy model is used. The word "policy" means "How should a network entity or a policy domain handle a particular flow of packets" in the context of

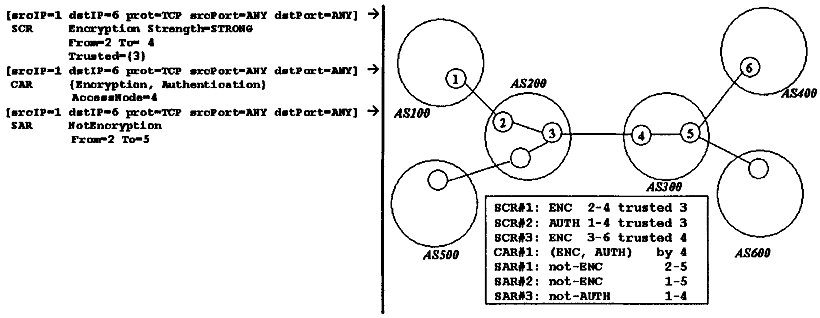
IPSec/VPN. In other words, policy is the command interface between a system administrator and a network device such that the human administrator can instruct the device to perform certain IPSec/VPN related operations. Once a "policy" rule is defined, a network device can unambiguously process the packet flow, including both packet headers and payloads. The left in Figure 3 is an example of policy. It specifies that the TCP traffic from A to B will be encrypted through an ESP Tunnel SA using the triple DES algorithm. And, the unidirectional SA starts at the security gateway SG-A and ends at B itself. This "low-level" policy specifies how SG-A and B should process the TCP packets from A to B.

On the other hand, in the context of this paper, the word "requirement" means "How should a sequence of information bits (the original payload) be handled from the source to the destination" regardless of any possible IP/IPSec header transformation on the route path (e.g., IPSec Transport or Tunnel mode, NAT or NAPT, IP fragmentation and de-fragmentation). In other words, "requirement" (sometimes we call it "high-level" policy) expresses the administrator's (or the user's) intentions about the security of some end-to-end information bits across different administrative domains without concerning low-level security operations. For instance, in the above SCR (Security Coverage Requirement) shown in Figure 3, the system administrator specifies that the TCP traffic between A and B must be encrypted between SG-A and B. And, furthermore, it might be OK to let SG-B to examine the content (for the purpose of intrusion detection, for example).

Policy and requirement are not one-to-one mapping. Usually, one requirement can be satisfied by a set of low-level security policy rules. As shown in [1][7], it is possible to find multiple security policy sets and any one of them can satisfy the target security requirement equally well.

In BANDS, IPSec/VPN security requirements have four different types:

- **Access Control Requirement (ACR):** ACR is related to a security gateway or firewall's access control function to some trusted traffic.
- **Security Coverage Requirement (SCR):** SCR applies security mechanism to prevent traffic from being compromised during the transmission across certain area. It requires the security protection to cover all links and nodes within the certain area. Various algorithms of authentication and encryption can be specified as the parameters in low-level policies.
- **Content Access Requirement (CAR):** Some network nodes may need to access the content of certain traffic, yet the content cannot be viewed if an encryption tunnel is built to across it. For example, a content access policy can be defined to deny all the encrypted traffic.
- **Security Associate Requirement (SAR):** Security Associations (SA) must be formed to perform desired security functions, thus there is a need to specify that some node desire to or not to set up SA with other nodes. Network peers are allowed to build SA unless explicitly disallowed.


```
[srcIP=1 dstIP=6 prot=TCP srcPort=ANY dstPort=ANY] →
 SCR     Encryption Strength=STRONG
         From=2 To= 4
         Trusted={3}
[srcIP=1 dstIP=6 prot=TCP srcPort=ANY dstPort=ANY] →
 CAR     {Encryption, Authentication}
          AccessNode=4
[srcIP=1 dstIP=6 prot=TCP srcPort=ANY dstPort=ANY] →
 SAR     NotEncryption
          From=2 To=5
```

```
SCR#1: ENC  2-4 trusted 3
SCR#2: AUTH 1-4 trusted 3
SCR#3: ENC  3-6 trusted 4
CAR#1: (ENC, AUTH)   by 4
SAR#1: not-ENC        2-5
SAR#2: not-ENC        1-5
SAR#3: not-AUTH       1-4
```

Figure 4. An end-to-end flow with 7 security requirements

Based on the definitions, the following policy solution shown in Figure 5 (we use a linear picture for an intuitive view) satisfies all the requirements. In order to provide flow protection to satisfy ENC SCRs, two encryption tunnels are built between 1 and 4, and between 4 and 5. Similarly, to satisfy AUTH SCR, two authentication tunnels are built between 1 and 3, and between 3 and 4. Obviously, all the tunnels are built to guarantee the certain protection, without violating any of the CARs and SARs. Due to the size limit, we only show the formal definition of one of the policies as follows:

```
[srcIP=1 dstIP=6 prot=TCP srcPort=ANY dstPort=ANY] →
 IPSec   Prot=ESP Mode=Tunnel
         Strength=STRONG
         from=1 to=4
```

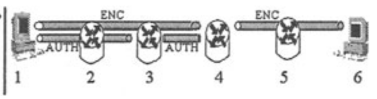


Figure 5. The policy solution

However, the following scenario as shown in Figure 6 may happen sometimes. Each policy satisfies its corresponding requirement, while putting all the policies together may cause conflicts. In this example, the flow is tunneled to 3 with authentication. On the other hand, it is tunneled from 2 to 4 with encryption before the authentication tunnel exits. It's easy to see that the authentication function applies at 1 and will not be de-applied at the tunnel 2 to 4. Thus, the traffic will be encapsulated by 1 for authentication and be encapsulated again by 2 for encryption. When 4 decapsulates and finds out that the destination is 3, the flow will be sent back to 3. Eventually, 3 will decapsulate the flow and send it to its destination. As a result, the flow is sent in plaintext from 3 to 4 because of the tunnel interaction, which violates the original security intentions (requirements). This is one of the reasons to avoid overlapping tunnels in BANDS architecture.

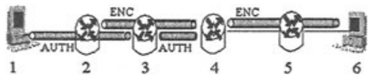


Figure 6. Overlapping Tunnels that cause conflicts

2.2 Inter-Domain Security Requirement Engineering

Under the BANDS framework, a system administrator and a user will use the SRSL (Security Requirement Specification Language) to specify one or more requirements related to either a particular domain (such as AS) or an information flow/bundle. Then, based on all the requirements we have in the Internet, in [7], we show that we can automatically and efficiently generate a set of IPSec security policy rules such that all requirements will be satisfied. Furthermore, if there are any conflicts among the requirements such that it is impossible to find a policy set to satisfy all the requirements, our program can detect such a case as well.

However, in an inter-domain environment such as Internet, it is practically and politically impossible, very inefficient, and un-scaleable to "collect all the requirements in the whole system" and then perform the task of requirement analysis and policy generation. One trick we can do under this situation is to determine the exact "route" path from the source of the information flow to the destination. Then, we can collect the requirements along the route path and then determine how to set up the security policy rules to satisfy all the security requirements along the route path. Furthermore, if BANDS detects that it is impossible to satisfy all the requirements along one particular route path, it might be able to try another route path (for example, in the case of multi-homing). Finally, we need to worry about "routing dynamics" in the Internet. Whenever the route is changed, BANDS needs to decide whether our current policy set has been affected or not. If necessary, we need to re-collect the requirements and re-compute the policies such that the security will not be affected by the routing dynamics.

3. RELATED WORK

Currently, there are mainly two works related to our research, MSME project at BBN Technologies and one of our research work --- Celestial system at NCSU.

– BBN's MSME architecture

MSME (Multidimensional Security Management and Enforcement) [2] is a research project being conducted at BBN technologies. It presents multidimensional architecture to allow each member in the distributed system to maintain its own policy management system while enabling him to exchange and resolve policies with other members of the coalition. MSME uses one level policy model to achieve the correctness of policy management, because there were vague definitions of security requirements and policies. The high-level security requirement is defined in an abstract format and somehow is mapped into a binding of the implementation, i.e. security policy. When the policy agreements are exchanged, the policies are complied to determine any conflict and to resolve it (if any). It is easy to see that, in MSME architecture, security requirements are exposed when the agreements are exchanged. Yet, in the network nowadays, people anticipate to keep the requirement information sharing as minimal as possible.

– Celestial system

The other research work dealing with security policy management is called Celestial system [3] at NCSU, which was designed to automatically discover security policies along the network path and dynamically configure security mechanisms across the network. Similarly, the Celestial system does not have explicit definition between security requirements and security policies. It just addresses every node's security capabilities and policies, and the receiver computes the corresponding policy strategy. Furthermore, the Celestial architecture is an unscalable and pure centralized security management system, indeed.

4. BANDS: A SECURITY POLICY MANAGEMENT SYSTEM ARCHITECTURE

4.1 Architecture overview

As we addressed in Section 2, there is a need to separate high-level requirements from low-level policies. Therefore, the ultimate aim is to be able to define high-level requirements beforehand and to automatically generate the low-level policies. On the other hand, the information shared must be kept as minimal as possible because we require only relevant information to be exposed. In order to achieve maximum autonomy, the principle of providing policy implementers with everything they need to know to satisfy the relevant requirements, but nothing more, should be respected. Furthermore, today's Internet acts like a huge distributed system. Therefore a pure centralized network management model is not ideal enough to provide reliable and scalable service, which could also guarantee the minimal information sharing. Therefore, the architecture that we developed adopts a hybrid structure of centralized and distributed systems. One of the important roles in the architecture is that every domain (i.e. AS) contains a Requirement Server (RS), which is responsible for cooperation and policy negotiation with other RSs in a distributed environment, as shown in Figure 7. Based on the architecture of RS, a two-phase policy negotiation process is preceded by each RS to generate correct policies automatically.

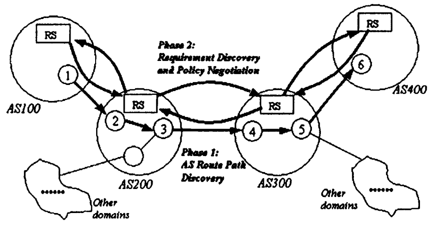

RS
RS
RS
RS
Phase 2: Requirement Discovery and Policy Negotiation
Phase 1: AS Route Path Discovery
1
2
3
4
5
6
AS100
AS200
AS300
AS400
Other domains
Other domains
••••••
••••••

Figure 7. An architecture example in a multiple-domain environment

The first step of the overall protocol is to discover the AS route path, in order to get the RS of each AS involved along the path ready for the incoming negotiation process. This phase is called "route path discovery", explained in Section 4.4. Based on the discovered AS route path, each RS should be able to identify the IP addresses (e.g. with DNS servers or LDAP) of other RSs along the path, such that the RS in the original AS could find out the corresponding security requirements along the path and exchange those which are relevant to each other. Each RS needs to make queries to its "neighbor" RS along the path for requirement discovery request. This requirement discovery phase is followed by policy negotiation. When the RS receives the negotiation message from the remote RS, it computes the corresponding policies using the automatical policy generation algorithm, direct approach [1], as explained in Section 4.5. Eventually, each RS notifies its local routers the security policies for a certain flow. Hence, under BANDS, we only introduce and add a requirement server in each domain, which stores routing information, maintains requirement information and performs policy negotiation, while routers remain unchanged. From the routers' point of view, the operations of BANDS operation are transparent, because the routers still carry out the regular router operations. A router performs its corresponding action only when it receives the policies from its RS.

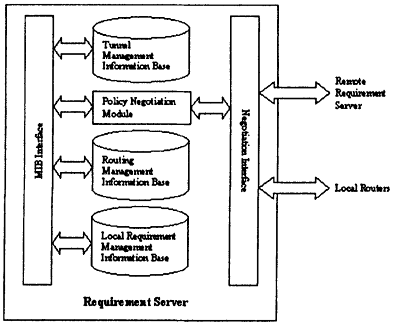

Tunnel Management Information Base
Policy Negotiation Module
Routing Management Information Base
Local Requirement Management Information Base
MIB Interface
Negotiation Interface
Remote Requirement Server
Local Routers
Requirement Server

Figure 8. The architecture of a Requirement Server

Figure 8 illustrates each AS's RS's architecture, which has several sub-components to interoperate with each other to precede the requirement discovery and policy negotiation with remote RSs and has interfaces to access its routing information, requirement information and tunnel information databases.

4.2 Components

The functionality of the components shown in Figure 8 is described in the following sections.

a) Routing Management Information Base (Routing MIB) and Local Requirement Management Information Base (Local Requirement MIB)

All the routing information of local routers is stored in each RS's Routing MIB such that the RS will be able to calculate the route path within the AS and the AS path across the AS. The RS needs to maintain periodical routing update from each of the routers in the AS through its Information Base Interface. Similarly, all the security requirements of local routers are stored in each RS's Local Requirement MIB. Together with the information in the Routing MIB, the Local Requirement MIB provides the RS the capability of computing the security policy for each local router for different flows. The RS also needs to maintain periodical requirement update from each of the routers in the AS through its MIB Interface. And the consistent maintenance is implemented using SNMP.

Each router runs an SNMP agent to maintain a local database of its security/routing requirements. Each RS, as SNMP management station, must have some SNMP management software, which is running one or more processes to communicate with the SNMP agents within the local domain (i.e. in the same AS) using SNMP protocol, in order to query the state of an agent's local requirement/routing information (under PKI infrastructure if necessary).

b) Tunnel Management Information Base (Tunnel MIB)

Since each RS knows what local routers establish what kinds of tunnels with what remote routers, every RS should be able to make query to each of its neighbors to find out what are the existing tunnels that are related to certain flows. In another word, with the information in Tunnel MIB, the RS is capable of building a tunnel map to certain flow, with which it could easily find out all the relevant security requirements and all the relevant existing tunnels to run the direct approach algorithm. In addition, the RS needs to maintain periodical tunnel update from each of the remote RSs because it must delete relevant information from its Tunnel MIB when some tunnel has been withdrawn.

c) Policy Negotiation Module (PolNegM)

With PolNegM, the RS negotiates with the remote RS for a set of security policies. Through Negotiation Interface, the PolNegM not only receives the policy negotiation requests from the remote RS and responds with its relevant requirement information to the remote RS, but also informs the appropriate security policy to each of the routers, which will participate along the route path. It will need to access the Routing MIB and the Local Requirement MIB to obtain the routing information for the flow and to gather the requirement details for each of the local routers on the route path of the flow. After collecting all the information, including the routing information and existing tunnel information, the PolNegM runs algorithm for each of the security coverage requirements to obtain the policy solution and then notifies both the local router and the remote router for the SA establishment.

4.3 Interface between Modules

a) MIB interface

The Information Base Interface provides the access interface between the database and the Policy Negotiation Module in the requirement server module. When the PolNegM needs to access the Routing MIB, it sends the query to the MIB Interface. The interface forwards the message to Routing MIB and sends the response back to PolNegM. The same procedure applies to the communication between PolNegM and Local Requirement MIB, and between PolNegM and Routing MIB.

b) Negotiation Interface

The Negotiation Interface provides the communication interface between the requirement server and the remote requirement server to negotiate security policy. When the RS contacts the remote RS to exchange requirements and to negotiate the policies, the Negotiation Interface forwards the negotiation message to the remote RS's PolNegM component through its Negotiation Interface

4.4 AS Route Path Discovery

The overall protocol consists of two phases, "AS route path discovery" phase and "collaborative policy negotiation protocol" phase. Before an end-to-end connection can be established securely, the RS in the original AS needs to initiate the "AS router path discovery" phase to explore the AS route path to discover all the RSs that will be involved in the subsequent policy negotiation phase.

Depending on what routing mechanism is used, the route path discovery strategies could vary. For instance, Border Gateway Reservation Protocol (BGRP) [11] is an inter-domain aggregated resource reservation protocol for unicast traffic, in which a sink tree is built for each of the stub domains to perform a destination-based reservation aggregation as shown in the example in Figure 9. If we use it under BANDS architecture, the overall protocol starts a route path discovery phase by sending a BGRP PROBE message to the destination. After the initiator gets a GRAFT message back, the exact AS route path has been probed and reserved. Each RS that will be involved in the following protocol then needs to sustain the flow information and launch the PolNegM to prepare for the negotiation.

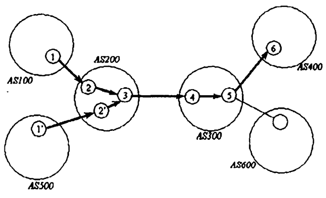


Figure 9. An example of a BGRP reservation sink tree rooted at router 6

4.5 Algorithm: direct approach

Before we dive into the next section for the detailed collaborative negotiation protocol, we will briefly introduce our automatic policy generation algorithm in each RS under BANDS, i.e. direct approach presented in [1], an efficient and scalable way for calculating policies. The algorithm considers one SCR at one time and takes in other relevant requirements (e.g. CARs and SARs) as parameters. By knowing the exact route path and the existing tunnels along the path, it computes the solutions to satisfy one SCR and related requirements without violating any existing tunnels.

To ensure that the generated policy satisfies one SCR without violating corresponding CARs and SARs, the algorithm starts with the initial (node) graph with full connection. To remove CAR conflicts, it eliminates all the links crossing the nodes that have CARs. Then it deletes all the links that starts or ends at the distrusted nodes. Finally among the rest of the edges, it uses Dijkstra shortest path algorithm to get the final tunnel solution. To better understand the algorithm, we use the previous scenario (SCR #2) in Section 2.1 as an example. Figure 10 (a) presents the initial primary graph and Figure 10 (b) shows the graph after CAR conflict check (i.e. after removing all the edges crossing node 4). Then Figure 10 (c) takes away all of the distrusted edges (i.e. edges that starts or ends at node 2, the distrusted node) and Figure 10 (d) comes with the final solution.

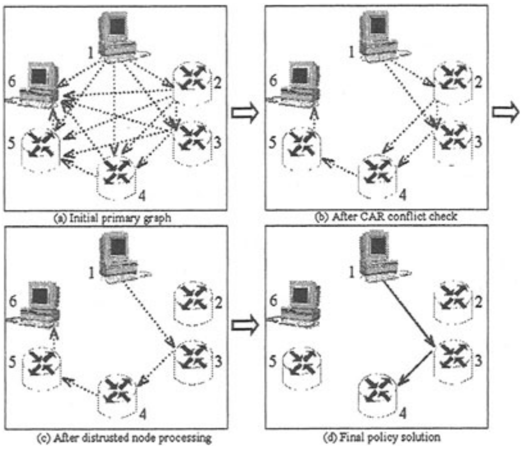

1
2
3
4
5
6
(a) Initial primary graph
(b) After CAR conflict check
(c) After distrusted node processing
(d) Final policy solution

Figure 10. An example of direct approach

4.6 The Collaborative Negotiation Protocol

The architecture is designed to provide reliable and secure end-to-end connections in a distributed environment. On one hand, the exact AS route path for a certain flow needs to be discovered. On the other hand, after the route path is explored, the sub-components of each RS along the path must collaborate and negotiate to figure out the appropriate tunnel (policy) solutions to guarantee that all the security requirements along the route path are satisfied. Our collaborative negotiation protocol consists of two steps.

1. Requirement Discovery Phase

```
Procedure RequirementDiscovery(flow)
1.  /* Get the route path of the flow through MIB Interface */
2.  routepath =MIBI_Get_Route_Path(flow)
3.  /* Check for all CARs on the path through MIB Interface */
4.  CARList = MIBI_Get_CAR_List(flow, routepath)
5.  /* If any CARs, send to previous Requirement Server through Negotiation Interface */
6.  For every CAR in CARList
7.      NI_Send_Req(CAR, PreRS)
8.  /* Check for all SCRs on the path through MIBI */
9.  SCRList = MIBI_Get_SCR_List(flow, routepath)
10. /* Requirement Aggregation */
11. For every SCR in SCRList
12.     if SCR. strength < flow. strength
13.     then SCR. strength = flow. strength
14.     if flow.ReqAggregate = 1
15.    then SCR.from = PreRS
16.          NI_Send_Req(SCR, PreRS)
17.          MIBI_Remove_Req(SCR)
18.          flow.ReqAggregate = 0
19. If no SCR between myself and next RS
20. then flow.ReqAggregate = 1
End of Procedure
```

```
Procedure GetRequirements(Req, flow )
1.  /* For CAR, insert into Local Requirement MIB */
2.  /* For SCR, if it is for myself, change SCR.from and insert into Local Requir
3.  /* otherwise send to previous RS */
4.   if (Req is SCR and ReqAggregate = 0)
5.   then Req.from = MIBI_Get_Reg_From(Req, flow)
6.  MIBI_Insert_Req(Req)
End of Procedure
```

Figure 11. The pseudo code of Requirement Discovery

After the AS route path is discovered, the RS in the original AS should start phase II to find out the requirements. The RS in the first AS sends out a "Requirement Discovery" message to the RS in the next AS, including the target security requirements for this end-to-end flow. It is easy for the RS to find out what are the local requirements for this certain flow in its Local Requirement MIB. The RS launches its PolNegM to prepare for the incoming negotiation request from remote RS. When the "Requirement Discovery" message reaches another RS in another AS, the RS needs to be prepared to participate in the negotiation and find out what are the related requirements for the flow. Furthermore, if the flow requirements could be aggregated with certain local requirements, the RS should be able to make the required aggregation and notify the RS, that may be affected accordingly, to update the requirements. If a router on the route path is the one which has CARs, its RS needs to advertise its CARs to its previous RS through its Negotiation Interface and the latter propagates them to every RS along the path. Therefore, for CARs, every RS on the path knows who are the ones that need to access the content for certain flow. For SARs, because SAR is the security requirement that two network entities cannot establish Security Associate between each other, both RSs should maintain the SARs. When the destination RS gets the "Requirement Discovery" message, it will respond a confirmation message to the sender and this step finishes. The pseudo code of the phase is as follows in Figure 11. It should be noted that solid synchronization mechanism should ensure the inter-domain requirement consistency.

2. Policy Negotiation Phase

Next, the initiator RS sends out a "Policy Negotiation" message to the destination RS. Accordingly, the PolNegM checks its Routing MIB to get the route path and queries Local Requirement MIB to obtain the relevant requirements. With the interact tunnel information in the Tunnel MIB, the PolNegM runs the direct approach algorithm to determine what policy set it will be using. After the solution is obtained, the RS must inform its neighbor and the next RS on the route path, the intended policy solution, which will be considered as the existing tunnel as the input parameters for the direct approach algorithm. If there is no appropriate solution available, the RS must send an error message back to the original RS to either adjust the requirements or withdraw the negotiation process. The pseudo code of this phase is as above in Figure 12.

```
Procedure PolicyNegotiation(flow)
1.  /* Generate policies for each SCR */
2.  /* Get the route path of the flow through MIB Interface */
3.  routepath = MIBI_Get_Route_Path(flow)
4.  /* Check for all SCRs on the path through MIB Interface */
5.  SCRList = MIBI_Get_SCR_List(flow, routepath)
6.  /* Check for all the existing interacted tunnels through MIB Interface */
7.  ExistTunnels = MIBI_Get_Exist_Tunnels(flow, routepath)
8.  /* Run the direct approach to generate the tunnel */
9.  /* If no solution found, send a fail message to previous RS */
10. For every SCR in SCRList
11.     policy = NonoverlappingDirectApproach(flow, SCR, ExistTunnels)
12.     if (policy = NULL) NI_Send_Fail(PreRS)
13.     Insert_Policy(policy, PolicyList)
14. /* If a confirmation is received, notify local routers of policies */
15. /* otherwise send a fail message back */
16. If a confirmation is received
17. then NI_Notify_Router(policy)
18. else NI_Send_Fail(PreRS)
End of Procedure
```

Figure 12. The pseudo code of Policy Negotiation

When the "Policy Negotiation" message reaches the receiver and no error occurs, the receiver responds with a confirmation message to the sender. As a result, every RS along the path knows the solution plan is feasible and it will inform the corresponding local router so that the router could initiate Internet Key Exchange (IKE) to exchange the session key between itself and the certain remote router and then set up IPSec Security Associate with the remote router. The specifications of IKE SA and IPSec SA establishment are in [9] [10].

The overall protocol finishes when the corresponding routers start to establish Security Associations with remote routers. Apparently, phase I guarantees the determination of the exact route path and make resource reservation on it if necessary. Next, all the RSs participate in the policy negotiation, including discovering all of the relevant requirements and calculating the policies. Thus certain security mechanism and functions could be applied to corresponding area/links to protect the flow. The security requirement information revealed in the protocol are just those CARs on the route path that every RS must know to avoid CAR violation during the policy computation and the existing tunnel information that is only related to the flow.

5. EXAMPLE SCENARIO

The following example demonstrates the whole operation of the various modules in the architecture and explains how the collaborative negotiation protocol works among the network peers. Suppose that there are 6 autonomous systems in the network using BGRP, each of which has an RS and some routers. Also, we have some security requirements[1] at relevant routers as three SCRs (ENC, ordinary, 2, 4, {3}), (ENC, strong, 3, 6, {4}) and (AUTH, middle, 1, 4, {3}), one CAR (ENC, AUTH, 4) and three SARs (2, 5, ENC), (1, 4, AUTH) and (1, 5, ENC).

1. Discover the route path

Router 1 needs to communicate with router 6 through a strong encryption mechanism, in our example. It initiates a route path discovery phase by sending a BGRP PROBE message to router 6. After router 1 gets a GRAFT message back from router 6, the exact path has been probed and reserved as shown in Figure 13.

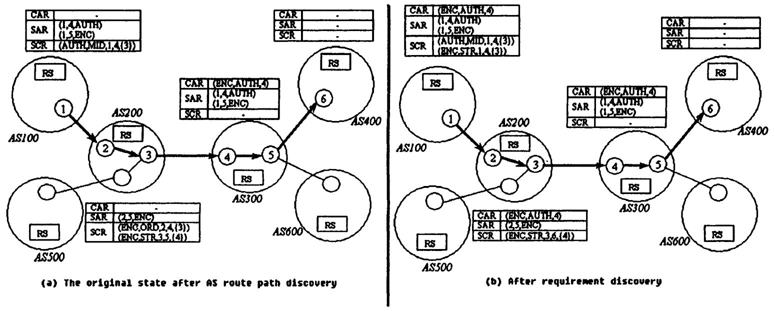


Figure 13. Example scenario: The original state after AS route path discovery (a) and after requirement discovery (b)

2. Discover the requirements

Now the RS in AS100 should begin the second phase with the requirement discovery. It sends the "Requirement Discovery" message to next RS in AS200 to see if there is any related requirement the latter has to share with it. The RS in AS200 figures out that one of its SCR (ENC, ORD, 2, 4, {3}) could be aggregated with the original flow requirement to be (ENC, STR, 1, 4, {3}). As the SCR.from is router 1, the RS in AS200 must inform the RS in AS100 of the newly aggregated requirement so that the latter must consider this new requirement when calculating the policy while the RS in AS200 must not. When the RS in AS300 receives the message and notices that there is a CAR related to the flow (ENC, AUTH, 4), therefore, it passes the CAR to its previous RS (in AS200) and the RS propagates it to the RS (in AS100) through its Negotiation Interface. Until the message reaches the other end (the RS in AS400), the requirement discovery part finishes with the last RS sending back a confirmation message.

For simplicity, we use the abbreviation MID, ORD, STR for MIDDLE, ORDINARY, STRONG respectively as one of the parameters of the requirements in the figure.

3. Negotiate the policies

The RS in AS100 gets the confirmation message and thus it could run the PolNegM to calculate the policy solutions shown in Figure 14 (a) (we use a linear picture for a intuitive view). Similarly, the RS in AS200 runs the algorithm in the PolNegM to figure out the policy solution shown in Figure 14 (b). Therefore, the overall security policy solution is as follows in Figure 14 (c).

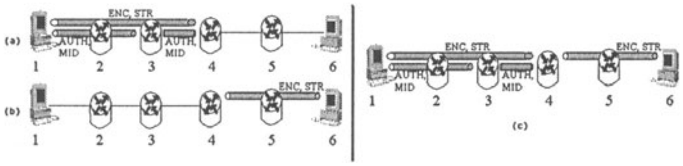


Figure 14. Example scenario: The policy solution computed by the RS in AS100 (a) and AS 200 (b) and the final policy solution (c)

6. EVALUATION

We have implemented a simulation program for the BANDS architecture. Under a simulated network topology, our program takes security requirements file as input and outputs IPSec/VPN policy rules. According to our preliminary performance results shown in Figure 15, while the number of security requirements increased dramatically as well as the number of messages transmitted during the whole process, the number of automatically produced security policies (the actual number of IPSec tunnels that will be built) increased linearly with the number of requirements. Because in the direct approach, one SCR requirement is considered at a time to compute the corresponding policy solution, our algorithm could be considered as a requirement-based algorithm. Certain requirement-based aggregation will be performed during "requirement discovery phase", as we show in the pseudo code. In summary, the BANDS framework from our simulation-based evaluation is indeed very scaleable and practical. Currently, our simulation program does not simulate the dynamic routing aspect of BANDS, and therefore, the route discovery part is omitted. More real network experiments involving routing are with development, in order to test the correctness and scalability of our architecture.

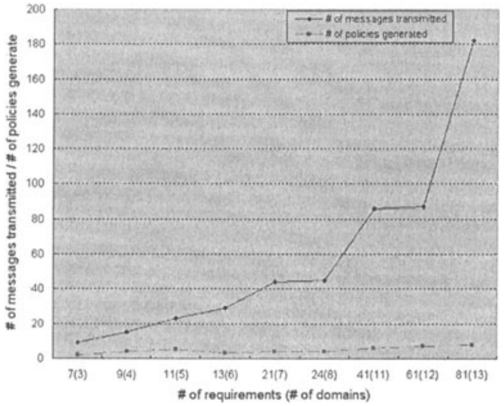


Figure 15. Experimental results

7. CONCLUSIONS

We presented a distributed architecture and a collaborative negotiation protocol for inter-domain Internet security policy management. We introduced the separation principle between security policies and requirements such that we can express the security intentions unambiguously. Furthermore, under the BANDS framework, the low-level policies are automatically generated with high-level requirements defined in advance. As a result, with the requirement server in each autonomous system in an inter-domain environment, we could manage the local requirements and calculate the corresponding policies for a certain flow. The overall protocol includes "AS route path discovery" phase and "collaborative negotiation protocol". After the AS route path is discovered, each RS along the AS path participates in the requirement discovery phase, followed by policy negotiation phase, which are both called the overall collaborative policy negotiation protocol. During the negotiation phase, the requirement information shared is kept as minimal as possible, so that only requirements that are pertinent to the flow are revealed to others.

While the performance simulation we have so far is only for static routing on some given end-to-end connections, our preliminary results demonstrate that our framework is very scaleable with respect to the number of automatically generated policy rules in an inter-domain networking environment. In the future, we will extend our evaluation to a prototype implementation on a routing network test-bed.

REFERENCES

[1] Z. Fu and S. F. Wu, "Automatic Generation of IPSEC/VPN Policies in an Intra-Domain Environment", 12th International Workshop on Distributed Systems: Operations & Management (DSOM 2001), October 15-17, 2001, Nancy, France.

[2] M. Condell, G. Patz, R. Krishnan, C. Lynn, "MSME Architecture", December 17, 2001, BBN Technologies.

[3] C. Xu, F. Gong, I. Baldine, C. Sargor, F. Jou, S. F. Wu, Z. Fu, H. Huang, "Celestial Security Management System", DARPA Information Survivability Conference and Exposition(DISCEX2000),IEEE Computer Society Press, Proceedings, pp.162-172, vol. 1.

[4] Moffett, J. D., "Requirements and Policies", Position paper for Workshop on Policies in Distributed Systems, 15-17 November 1999, HP- Laboratories, Bristol, UK.

[5] Moffett, J. D., Sloman, M. S., "Policy Hierarchies for Distributed Systems Management", IEEE Journal on Selected Areas in Communication, 11(9), 1404-1414.

[6] Z. Fu, "Automatic Generation of Security Policies", Technical Report, http://shang.csc.ncsu.edu/secpolicy.pdf.

[7] Z. Fu, S. F. Wu, H. Huang, K. Loh, F. Gong, "IPSec/VPN Security Policy: Correctness, Conflict Detection and Resolution", IEEE Policy 2001 Workshop, Jan. 2001.

[8] S. Blake et al., "An architecture for differentiated service," RFC 2475, Internet Engineering Task Force, Dec.1998.

[9] Dan Harkins et al., "Proposal for the IKEv2 Protocol", Internet Draft, <draft-ietf-ipsec-ikev2-02.txt>, April 2002.

[10] S. Kent et al., "Security Architecture for the Internet Protocol", RFC 2401, Internet Engineering Task Force, November 1998.

[11] P. Pan, E, Hahne, and H. Schulzrinne, "BGRP: A Tree-Based Aggregation Protocol for Inter-domain Reservations", Journal of Communications and Networks, Vol. 2, No. 2, June 2000, pp. 157-167.

SESSION 5

Monitoring and Performance

Chair: Rolf Stadler
KTH Stockholm, Sweden

PERFORMANCE MANAGEMENT FOR CLUSTER BASED WEB SERVICES

R. Levy, J. Nagarajarao, G. Pacifici, M. Spreitzer, A. Tantawi, and A. Youssef
IBM T.J. Watson Research Center, Yorktown Heights, NY 10598

Abstract: We present an architecture and prototype implementation of a performance management system for cluster-based *web services*. The system supports multiple classes of web services traffic and allocates server resources dynamically so to maximize the expected value of a given cluster utility function in the face of fluctuating loads. The cluster utility is a function of the performance delivered to the various classes, and this leads to differentiated service. In this paper we will use the average response time as the performance metric. The management system is *transparent*: it requires no changes in the client code, the server code, or the network interface between them. The system performs three performance management tasks: resource allocation, load balancing, and server overload protection. We use two nested levels of management mechanism. The inner level centers on queuing and scheduling of request messages. The outer level is a feedback control loop that periodically adjusts the scheduling weights and server allocations of the inner level. The feedback controller is based on an approximate first-principles model of the system, with parameters derived from continuous monitoring. We focus on SOAP-based web services. We report experimental results that show the dynamic behavior of the system.

1. INTRODUCTION

Today we are seeing the emergence of a powerful distributed computing paradigm, broadly called web services [17]. Web services feature ubiquitous protocols, language-independence, and standardized messaging. Due to these technical advances and growing industrial support, many believe that web services will play a key role in dynamic e-business [2]. In such an environment, a web service provider may provide multiple web services, each in multiple grades, and each of those to multiple customers. The provider will thus have multiple classes of web service traffic, each with its own characteristics and requirements. Performance management becomes a key problem, particularly when service level agreements (SLA) are in place. Such service level agreements are included in service contracts between providers and customers and they specify both performance targets, known as performance objectives, and financial consequences for meeting or failing to meet those targets. A service level agreement may also depend on the level of load presented by the customer.

In this paper we present an architecture, and describe a prototype implementation, of a performance management system for web services that supports service level agreements. We have designed and implemented reactive control mechanisms to handle dynamic fluctuations in service demand while keeping service level agreements in mind. Our mechanisms dynamically allocate resources among the classes of traffic, balance the load across the servers, and protect the servers against overload — all in a way that maximizes a given cluster utility function. This produces differentiated service.

We introduce a *cluster utility function* that is a composition of two kinds of functions, both given by the service provider. First, for each traffic class, there is a class-specific *utility function* of performance. Second, there is a *combining function* that combines the class utility values into one cluster utility value. This parameterization by two kinds of utility function gives the service provider flexible control over the trade-offs made in the course of service differentiation. In general, a service provider is interested in profit (which includes cost as well as revenue) as well as other considerations (e.g., reputation, customer satisfaction).

We have organized our architecture in two levels: (i) a collection of in-line mechanisms that act on each connection and each request, and (ii) a feedback controller that tunes the parameters of the in-line mechanisms. The in-line mechanisms consist of connection load balancing, request queuing, request scheduling, and request load balancing. The feedback controller periodically sets the operating parameters of the in-line mechanisms so as to maximize the cluster utility function. The feedback controller uses a performance model of the cluster to solve an optimization problem. The feedback controller continuously adjusts the model parameters using measurements of actual operations. In this paper we report the results obtained using an approximate, first-principles model.

We focus on SOAP-based web services and use statistical abstracts of SOAP response times as the characterization of performance. We allow ourselves no functional impact on the service customers or service implementation: we have a transparent management technique that does not require changes in the client code, the server code, or the network protocol between them.

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 presents the system architecture and prototype implementation. Performance modeling and optimization analysis are described in Section 4. Section 5 illustrates some experimental results, showing both transient responses and service differentiation. Section 6 presents conclusions and discusses future work.

2. RELATED WORK

Several research groups have addressed the issue of QoS support for distributed systems [15]. In this section we summarize the current state of the art.

The first class of research studies deals with session-based admission control for overload protection of web servers. Chen et al. [9] proposed a dynamic weighted fair sharing scheduler to control overloads in web servers. The weights are dynamically adjusted, partially based on session transition probabilities from one stage to another, in order to avoid processing requests that belong to sessions likely to be aborted in the future. Similarly, Carlström et al. [7] proposed using generalized processor sharing for scheduling requests, which are classified into multiple session stages with transition probabilities, as opposed to regarding entire sessions as belonging to different classes of service, governed by their respective SLAs.

Another area of research deals with performance control of web servers using classical feedback control theory. Abdelzaher et al. [1] used classical feedback control to limit utilization of a bottleneck resource in the presence of load unpredictability. They relied on scheduling in the service implementation to leverage the utilization limitation to meet differentiated response-time goals. They used simple priority-based schemes to control how service is degraded in overload and improved in under-load. In this paper we use a new technique that gives the service provider a finer grain control on how the control subsystem should tradeoff resources among different web services requests. Diao et al. [10] used feedback control based on a black-box model to maintain desired levels of memory and CPU utilization. In this paper we use a first-principles model and maximize a cluster objective function.

Web server overload control and service differentiation using OS kernel-level mechanisms, such as TCP SYN policing, has been studied in [18]. A common tendency across these approaches is tackling the problem at lower protocol layers, such as HTTP or TCP, and the need to modify the web server or the OS kernel in order to incorporate the control mechanisms. Our solution on the other hand operates at the SOAP protocol layer, which does not require changes to the server, and allows for finer granularity of content-based request classification.

Service differentiation in cluster-based network servers has also been studied in [4] and [20]. The approach taken here is to physically partition the server farm into clusters, each serving one of the traffic classes. This approach is limited in its ability to accommodate a large number of service classes, relative to the number of servers. Lack of responsiveness due to the nature of the server transfer operation from one cluster to another is typical in such systems. On the other hand, our approach uses statistical multiplexing, which makes fine-grained resource partitioning possible, and unused resource capacities can be instantaneously shared with other traffic classes.

Chase et al. [8] refine the above approach. They note that there are techniques (e.g., cluster reserves [5], and resource containers [6]) that can effectively partition server resources and quickly adjust the proportions. Like our work, Chase et al. also solve a cluster-wide optimization problem. They add terms for the cost (due, e.g., to power consumption) of utilizing a server, and use a more fragile solution technique. Also, they use a black-box model rather than first-principles one.

Zhao and Karamcheti [19] propose a distributed set of queuing intermediaries with non-classical feedback control that maximizes a global objective. Their technique does not decouple the global optimization cycle from the scheduling cycle.

In this paper we use the concept of *utility function* to encapsulate the business importance of meeting or failing to meet performance targets for each class of service. The notion of using a utility function and maximizing a sum [13] or a minimum [14] of utility functions for various classes of service has been used to support service level agreements in communication services. In such analyses, the utility function is defined in terms of bandwidth allocated (i.e. resources). In our work, we define the class utility function in terms of the experienced performance relative to the guaranteed service objective. Thus, it is possible to express the business value of meeting the service level objective as well as deviating from it. Further, the effect of the amount of allocated resources on performance level is separated from the business value objectives.

3. PERFORMANCE MANAGEMENT SYSTEM ARCHITECTURE AND IMPLEMENTATION

In this section we present the system architecture and prototype implementation of a management system for web services. This system allows service providers to offer and manage service level agreements for web services. The service provider may offer each web service in different *grades* of service level, with each *grade* defining a specific set of performance objective parameters. For example, the `StockUtility` service could be offered in either *gold*, *silver*, or *bronze* grade, with each grade differentiated by performance objective and base price. A prototypical grade will say that the service customers will pay $10 for each month in which they request less than 100,000 transactions and the 95th percentile of the response times is smaller than 5 seconds, and $5 for each month of slower service.

Using a configuration tool, the service provider will define the number and parameters of each grade. Using a subscription interface users can register with the system and subscribe to services. At subscription time each user will select a specific offering and associated grade.

The service provider uses the configuration tool to also create a set of traffic classes and map a `<customer, service, operation, grade>` tuple into a specific *traffic class* (or simply *class*). The service provider assigns a specific response time target to each traffic class. Our management system allocates resources to traffic classes and assumes that each traffic class has a homogenous service execution time.

We introduce the concept of class to separate operations with widely differing execution time characteristics. For example the `StockUtility` service may support the operations `getQuote()` and `buyShares()`. The fastest execution time for `getQuote()` could be 10 ms while the `buyShares()` cannot execute faster that 1sec. In such a case the service provider would map these operations into different classes with different set of response time goals. We also use the concept of class to isolate specific contracts to handle the requests from those customers in a specific way.

Figure 1 shows the system architecture. The main components are: a set of gateways, a global resource manager, a management console, and a set of server nodes on which the target web services are deployed. We use gateways to execute the logic that controls the request flow and we use the server nodes to execute the web services logic. Gateway and server nodes are software components. We usually have only one gateway per physical machine and in general we have server nodes and gateways on separate machines. The simplest configuration is one gateway and one server node running on the same physical machine.

In this paper we assume that all server nodes are homogeneous and that every web service is deployed on each server. We can deal with heterogeneous servers by partitioning them into disjoint pools, where all the servers in a given pool have the same subset of web services deployed. Refer to [12] for details on how to use server pools.

The servers, gateways, global resource manager, and console share monitoring and control information via a publish/subscribe network. In coping with higher loads, the system scales by having multiple gateways. An L4 switch distributes the incoming load across the gateways.

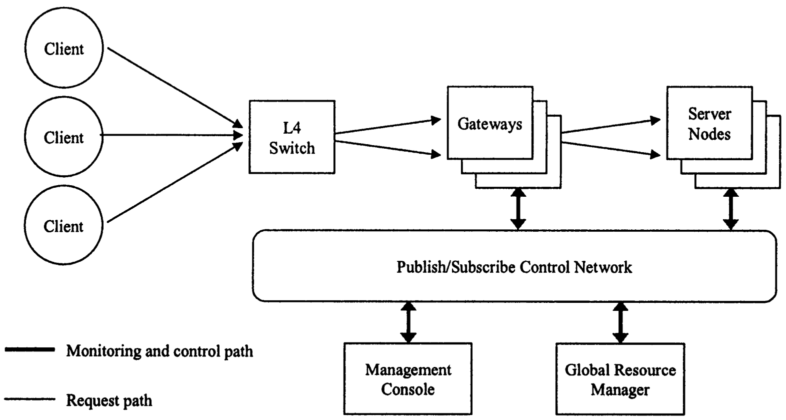


Figure 1. System Overview

3.1 Gateway

We use gateways to control the amount of server resources allocated to each traffic class. By dynamically changing the amount of resources we can control the response time experienced by each traffic class.

Gateways dispatch requests to servers. We denote with N_s the capacity of server s. N_s represents the maximum number of web services requests that server s can execute *concurrently*. We select N_s to be large enough to efficiently utilize the server's physical resources, but small enough to prevent overload and performance degradation. In the remainder of this paper we assume that N_s is given.

We partition N_s among all gateways and we denote with $N_{g,s}$ the maximum number of concurrent requests that server s executes on behalf of gateway g. We also use $w_{g,c}$ to describe the minimum number of class c requests that all servers will execute on behalf of gateway g. Each request executes in a separate initial thread. Thus, we refer to $w_{g,c}$ as server *threads*. In Section 4 we will describe how we compute $w_{g,c}$ and $N_{g,s}$, while, in this section we describe how gateway g enforces the $w_{g,c}$ and $N_{g,s}$ constraints. For each gateway g, we use w_g and N_g to denote the following:

$$w_g = \sum_{c \in C} w_{g,c} \ , \quad N_g = \sum_{s \in S} N_{g,s} \ , \tag{1}$$

where C and S denote the set of all classes and servers, respectively. Figure 2 illustrates the gateway components. We have used Axis [3] to implement all our gateway components and we have implemented some of the mechanisms using Axis *handlers*, which are generic interceptors in the stream of message processing. Axis handlers can modify the message, and can communicate out-of-band with each other via an Axis message context associated with each SOAP invocation (request and response) [3].

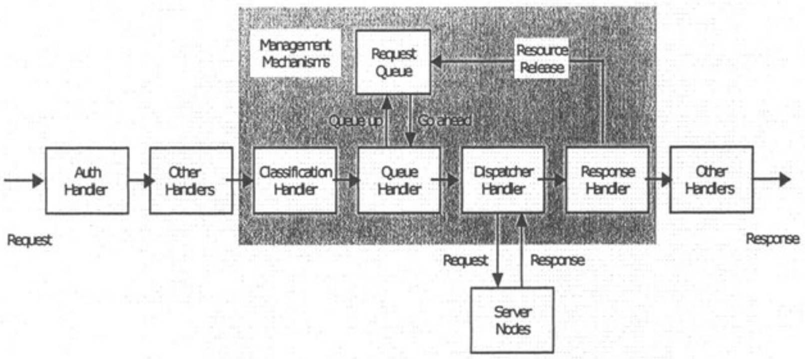


Figure 2. Gateway components

When a new request arrives a *classification handler* determines the traffic class of the request. The mapping functions use the request meta-data (user id, subscriber id, service name, etc.). In our implementation the classification handler uses the user and SOAP action fields in the HTTP headers as inputs, and reads the mappings from configuration files. We avoid parsing the incoming SOAP request to minimize the overhead.

After we classify the requests, we invoke the *queue handler*, which in turn contacts a *queue manager*. The queue manager implements a set of logical FIFO queues one for each class. When the queue handler invokes the queue manager the queue manager suspends the request and adds the request to the logical queue corresponding to the request's class.

The queue manager includes a *scheduler* that runs when a specific set of events occurs and selects the next request to execute. The queue manager on gateway g tracks the number of outstanding requests dispatched to each server and makes sure that there are at most N_g requests concurrently executing on all the servers. When the number of concurrently outstanding requests from gateway g is smaller than N_g the scheduler selects a new requests for execution.

The scheduler uses a weighted round robin scheme. The total length of the round robin cycle is w_g and the length of class c interval is $w_{g,c}$. We use a dynamic boundary and work conserving discipline that always selects a non-empty queue if there is at least one. The above discipline guarantees that during periods of resource contention the server nodes will concurrently execute at least $w_{g,c}$ requests of class c on behalf of gateway g.

After the scheduler selects a request, the queue manager resumes the execution of the request's corresponding queue handler. The queue manager collects statistics on arrival rates, execution rates, and queueing time and periodically broadcasts these data on the control network.

The *dispatch handler* selects a server and sends the request to the server, using a protocol defined by configuration parameters. Our implementation supports SOAP over HTTP and SOAP over JMS [16]. The dispatch handler distributes the requests among the available servers using a simple load balancing discipline while enforcing the constrain that at most $N_{g,s}$ requests execute on server s concurrently on behalf of gateway g.

When a request completes its execution the *response handler* reports to the *queue manager* the completion of the request's processing. The queue manager uses this information to both keep an accurate count of the number of requests currently executing and to measure performance data such as service time.

The gateway functions may be run on dedicated machines, or on each server machine. The second approach has the advantage that it does not require a sizing function to determine how many gateways are needed, and the disadvantage that the server machines are subjected to load beyond that explicitly managed by the gateways.

3.2 Global Resource Manager

The *global resource manager* runs periodically and computes $N_{g,s}$ and $w_{g,c}$ using the request load statistics and performance measurements from each gateway. Figure 3 shows the global resource manager inputs and outputs. In addition to real-time dynamic measurements, the global resource manager uses resource configuration information and the *cluster utility function*. The cluster utility function consists of a set of class *utility functions* and a *combining function*. Each class *utility function* maps the performance of a particular traffic class into a scalar value that encapsulates the business importance of meeting, failing to meet, or exceeding the class service level objective. A *combining function* combines the class *utility* function into one *cluster utility* function. In this paper we have implemented the combining function as a sum of the utility functions, however, our work could be extended to study the impact of other combining functions on the structure of the solution.

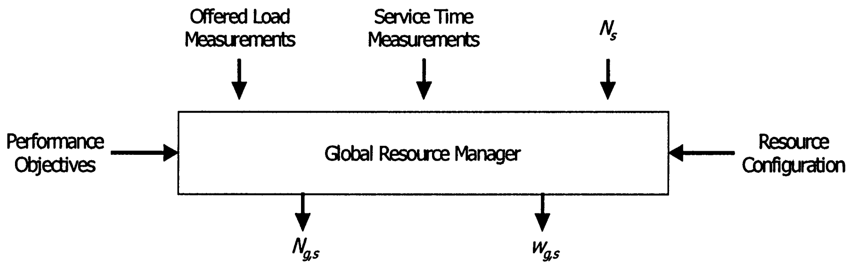
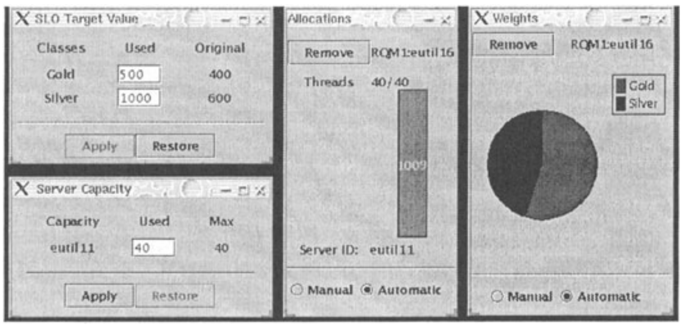


Figure 3. Global resource manager inputs and outputs

The global resource manager uses a queuing model of the system to predict the performance that each class would experience for a given allocation $w_{g,c}$ and the corresponding $N_{g,s}$. The global resource manager implements a dynamic programming algorithm to find the $w_{g,c}$ and $N_{g,s}$ that maximize the cluster utility function. After the global resource manager computes a new set of $w_{g,c}$ and $N_{g,s}$ values, it broadcasts them on the control network. Upon receiving the new resource allocation parameters each gateway switches to the new values of $w_{g,c}$ and $N_{g,s}$. We discuss the algorithm used to predict the class performance and maximize the cluster utility function in Section 4.

3.3 Management Console

The *management console* offers a graphical user interface to the management system. Through this interface the service provider can view and override all the configuration parameters. We also use the console to display the measurements and internal statistics published on the control network. Finally we can use the console to manually override the control values computed by the global resource manager. Figure 4 shows a subset of the views available from our management console.


SLO Target Value

Classes	Used	Original
Gold	500	400
Silver	1000	600

Server Capacity

Capacity	Used	Max
eutil11	40	40

Allocations
RQM1:eutil16
Threads 40/40
100%
Server ID: eutil11
○ Manual [✓ Automatic]

Weights
RQM1:eutil16
Gold
Silver
○ Manual [✓ Automatic]

Figure 4. Management console: configuration and control values

4. MODELING AND OPTIMIZATION

In this section we describe how the global resource manager computes the resource allocation. First we give an abstract definition of the problem solved. Then we discuss the simplified queuing model used to predict the performance of each class for a given resource allocation. Finally, we examine the class utility functions detail.

4.1 The Resource Allocation Problem

The global resource manager computes the $N_{g,s}$ and $w_{g,c}$ values to maximize the cluster utility function over the next control period. We decouple the $N_{g,s}$ and $w_{g,c}$ problems by solving for the $w_{g,c}$ first, and then deriving the $N_{g,s}$ from them.

To determine the $w_{g,c}$, we use dynamic programming to find the $w_{g,c}$ that maximizes the cluster utility function Ω which we define as the sum of each class utility function U_c. In particular Ω is given by:

$$\Omega = \sum_{c \in C} \sum_{g \in G} U_c(w_{g,c}) \qquad (2)$$

subject to:

$$1 \le w_{g,c} \le N, \qquad \sum_{c \in C} \sum_{g \in G} w_{g,c} = N, \qquad (3)$$

where $N \equiv \sum_{s \in S} N_s$, (4)

and where C, G and S denote the set of classes, gateways and servers, respectively. The utility function $U_c(w_{g,c})$ defines the utility associated with allowing $w_{g,c}$ requests of class c traveling through gateway g to concurrently execute on any of the servers. In the following section we discuss the structure of the utility function and in Section 4.3 we show how we compute U_c as a function of $w_{g,c}$.

As we mentioned in the previous section, we enforce for each server s, a limit N_s on the maximum number of requests that may be concurrently active on that server [12]. Once we have computed $w_{g,c}$. the value w_g derived from equation 1 represents the portion of server resources that have been allocated to gateway g. To compute $N_{g,s}$ for each gateway g we divide each server s available concurrency N_S among the gateways in proportion to w_g. In particular for each server s we select the point

$$[N_{1,s}, \text{K } N_{n_G,s}]$$

where n_G is the number of gateways) with integer-valued coordinates constrained by

$$\sum_{g \in G} N_{g,s} = N_s, \qquad (5)$$

and near the point $[\hat{N}_{1,s}, \mathrm{K}\ \hat{N}_{n_G,s}]$ defined by

$$\hat{N}_{g,s} = \frac{w_g}{N} N_s \tag{6}$$

where N is the total number of resources across all servers as defined in equation 4.

4.2 The Structure of Class Utility Functions

We use U_c to encapsulate the business importance of meeting or failing to meet class c performance. In this paper, we express each class performance objective as an upper bound on the average response time and therefore U_c will depend on the negotiated upper bound as well as the predicted response time given an allocation of $w_{g,c}$ resources. In the studies reported in this paper, we use a prototypical function to express the utility of class c when its requests experience a performance t_c under a contracted performance objective τ_c. An example for such a function is given below.

$$U_c(\tau_c, t_c) = \begin{cases} \alpha_c & \mathit{if}\ 0 \le t_c < 1/\mu_c \\ a_c\left(\dfrac{\tau_c - t_c}{\tau_c - 1/\mu_c}\right) & 1/\mu_c \le t_c < \tau_c \\ \dfrac{-\alpha_c (t_c - \tau_c)^{\beta_c}}{\beta_c(\tau_c - 1/\mu_c)} & t_c \ge \tau_c \end{cases} \tag{7}$$

The function in equation 7 and shown in Figure 5 compares average response time t_c to target response time τ_c for class c as follows. The best possible long-term average is $1/\mu_c$ where μ_c is the mean service rate for class c. When $t_c = 1/\mu_c$ $U_c(\tau_c, t_c)$ is constant. Between that point and $t_c = \tau_c$, we simply follow a straight line. For $t_c > \tau_c$ we use a negative polynomial function to map response times bigger than the objective into a negative value of $U_c(\tau_c, t_c)$. For the plot in Figure 5 we have used $\mu_c = 1$, $\tau_c = 6$, $\alpha_c = [1,2,3]$ and $\beta_c = [1,3,5]$. By increasing α_c we control the business importance of exceeding the target for class c, while by increasing β_c we can control how fast the business utility degrades when class c experiences a delay bigger than the objective.

By changing the size and shape of the utility function we can influence how resource are allocated to each class of traffic and in turn the class performance. A more detailed description of the concept of the utility function and its impact on the overall system is given in [12]. In the next section we describe how we estimate the expected response time t_c for class c given a scheduling weight of $w_{g,c}$.

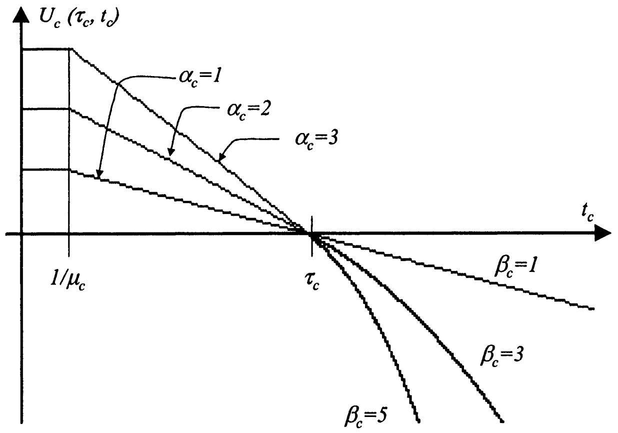


Figure 5. Utility function

4.3 System Modeling

To predict the average response time $t_{g,c}$ given a proposed allocation $w_{g,c}$ we use the observed arrival rate, response time, and the previous allocation values, denoted by $\tilde{\lambda}_{g,c}$, $\tilde{t}_{g,c}$, and $\tilde{w}_{g,c}$, respectively.

We use an M/M/1 queue [11] to model the response time behavior of requests of class c traveling through gateway g, i.e., we assume that $\tilde{\lambda}_{g,c}$ is evenly divided among the $\tilde{w}_{g,c}$ server threads that have been concurrently executing all requests of class c traveling through gateway g during the previous control cycle. Using this assumption we compute the *equivalent service rate* of the M/M/1 queue that has been handling the fraction of requests served by one of the $w_{g,c}$ threads. The equivalent service rate is given by:

$$\tilde{\mu}_{g,c} = 1/\tilde{t}_{g,c} + \tilde{\lambda}_{g,c}/\tilde{w}_{g,c} \tag{8}$$

Figure 6 exemplifies the above assumption. We now use $\tilde{\mu}_{g,c}$ to predict the response time of all class c requests traveling through gateway g in the next control cycle under an allocation of $w_{g,c}$ threads, as follows

$$t_{g,c}(w_{g,c}) = \frac{1}{1/\tilde{t}_{g,c} + \tilde{\lambda}_{g,c}\left(1/\tilde{w}_{g,c} - 1/w_{g,c}\right)} \tag{9}$$

In the previous calculation we have assumed that the request load in the new cycle is equal to the previous one.

Using equation 9 and 7 we can express the utility U_c (τ_c, t_c) as a function of the expected allocation $w_{g,c}$. Using dynamic programming we can then compute the set of $w_{g,c}$ that will maximize the cluster utility function Ω in equation 2 under the constraints in equation 3.

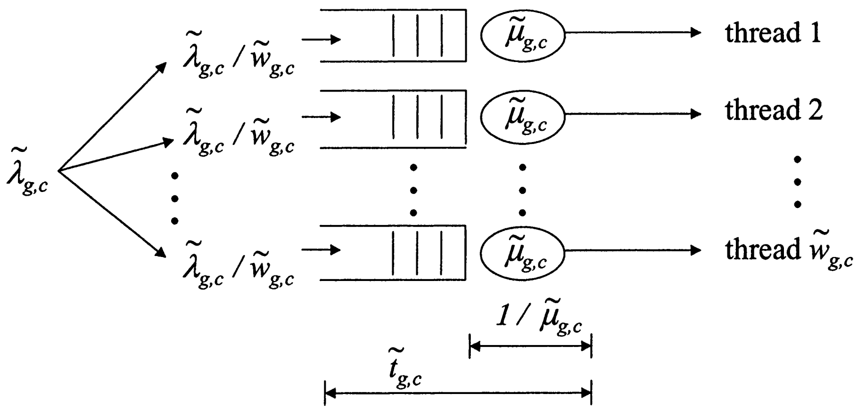

$\tilde{\lambda}_{g,c}$
$\tilde{\lambda}_{g,c} / \tilde{w}_{g,c}$
$\tilde{\lambda}_{g,c} / \tilde{w}_{g,c}$
$\tilde{\lambda}_{g,c} / \tilde{w}_{g,c}$
$\tilde{\mu}_{g,c}$
thread 1
thread 2
thread $\tilde{w}_{g,c}$
$1 / \tilde{\mu}_{g,c}$
$\tilde{t}_{g,c}$

Figure 6. Modeling the response time behavior for class *c* requests handled by gateway *g*

The resource allocation methodology described in this section will achieve an optimal resource allocation only under the assumptions mentioned above. For all other cases our methodology achieves a sub-optimal solution. Given the nature of our system an optimal allocation can be determined only by simulation and extensive search. More work is required to determine the difference between our approach and an optimal allocation of resources. In [12] we report the results of several experiments indented to study the effectiveness of this approach. In the next section we report a subset of these experiments.

5. EXPERIMENTAL RESULTS

In order to illustrate the fundamental behavior of the system, the following experiments were conducted using a combined gateway and server machine, while another machine was used to generate the traffic load.

During the experiment, clients connect to the gateway and send requests to a synthetic service, with exponentially distributed service time. The service alternates between CPU-bound processing and sleeping. The sleeping intervals are intended to emulate periods in which a process awaits response from a back-end server or database.

In order to determine the desired N_S for the one server, we examined the system throughput for various settings of N_S. In these experiments, the load consisted of only one traffic class, and we ensured that the request queue was always non-empty. As shown in Figure 7, a maximum throughput of 23.5 requests/sec is achieved at an N_S of 10. For larger values of N_S the CPU reaches saturation and the overhead begins to degrade the server throughput. In the experiments reported below N_S was always set to 10.

In the following experiment, clients are classified into two types: *gold* and *silver*. The gold clients' service level agreements specify a performance objective of *1 sec* average response time, while the silver clients' service level agreements specify a *2 sec* average response time target.

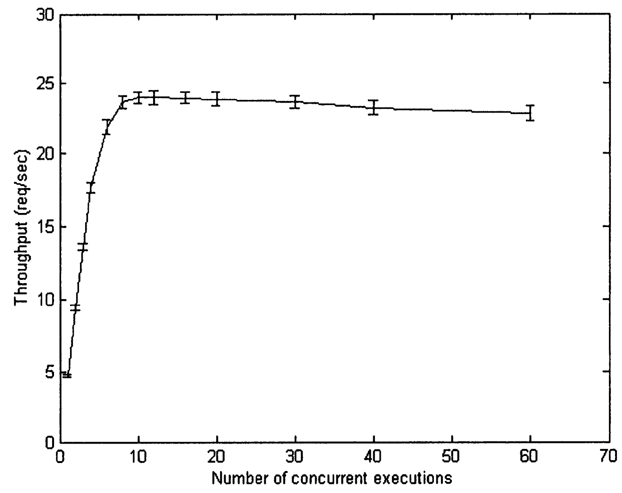


Figure 7. Throughput vs requested maximum number of concurrent executions

This experiment emulated an infinite client population, where initially the server was subjected to *1req/sec* from gold clients and *11.5req/sec* from silver clients. After *100sec*, the gold traffic rate was increased to *11.5req/sec*, which brought the total load close to the system capacity. We show the effect of this change in traffic on response time in Figures 8. Since the invoked service spends a good portion of its time performing CPU-bound processing, the service time increases as the degree of concurrency of executing requests increases. This experiment demonstrates that the control mechanism immediately started to react to the load changes in order to maximize the cluster utility. In this experiment, the control cycle for the global resource manager was set to *10sec*. We smoothed the load and response time statistics used by the global resource manager over a *30sec* intervals using a sliding window. The plot in Figures 8 shows values smoothed by that *30sec*-sliding window. We report more details on these experiments as well as additional experiments, with specific settings, in [12].

6. CONCLUSIONS AND FUTURE WORK

We have presented an architecture and a prototype implementation of a performance management system for cluster-based web services. The management system is transparent and allocates server resources dynamically so to maximize the expected value of a given cluster utility function. We use a cluster utility to encapsulate business value, in the face of service level agreements and fluctuating offered load. The architecture features gateways that implement local resource allocation mechanisms. A global resource manager solves an optimization problem and tunes the parameters of the gateway's mechanisms. In this study we have used a simple queuing model to predict the response time of request for different resource allocation values. Feedback controllers based on first-principles model of the system converge quickly and with fewer oscillations than controllers based on a black-box model.

Our work can be extended in several directions. Our platform could be enhanced with additional management functionality such as policing, admission control and fault management. We will need to develop more sophisticated models of web services and web services traffic loads to study and predict platform performance under different service and traffic conditions. The effect of control parameters, such as control cycle, on the performance of the feedback controller needs further study. We could refine our global resource manager by adding black box and hybrid control techniques. Finally, we will need to study the impact of using other scheduling algorithms on the end-to-end resource management problem, especially in the presence of multiple gateways.

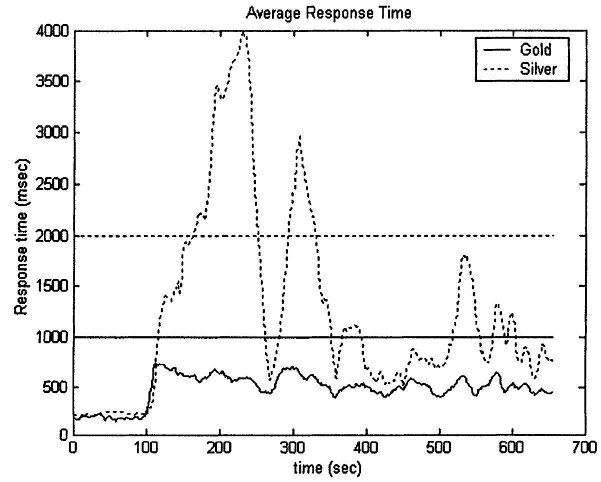


Figure 8. Response time for infinite client population experiment

REFERENCES

[1] T. Abdelzaher, K. Shin, and N. Bhatti, "Performance Guarantees for Web Server End-Systems: A Control-Theoretical Approach", *IEEE Transactions on Parallel and Distributed Systems* , Vol. 13, No. 1, Jan 2002.

[2] S. Aissi, P. Malu, and K. Srinivasan, "E-Business Process Modeling: The Next Big Step", *IEEE Computer* 35(5), pp 55-62, May 2002.

[3] Apache XML Project, http://xml.apache.org/axis/

[4] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalantar, S. Krishnakumar, DP. Pazel, J. Pershing, and B. Rochwerger, "Oceano SLA based management of a computing utility", *Proceedings of 2001 International Symposium on Integrated Network Management,* Page 14-18. May 2001.

[5] M. Aron, P. Druschel, and W. Zwaenepoel, "Cluster Reserves: A Mechanism for Resource Management in Cluster-based Network Servers", *ACM Sigmetrics 2000*, Santa Clara, CA, Jun 2000.

[6] G. Banga, J. Mogul, and P. Druschel, "Resource containers: A new facility for resource management in server systems", *Proceedings of the Third Symposium on Operating Systems Design and Implementation* (OSDI'99), New Orleans, LA, Feb 1999.

[7] J. Carlström, and R. Rom, "Application-aware Admission Control and Scheduling in Web Servers", *IEEE INFOCOM 2002, New York, NY*, Jun 2002.

[8] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle, "Managing Energy and Server Resources in Hosting Centers", *Proceedings of 18th ACM Symposium on Operating System Principles*, pages 103-116, Oct 2001.

[9] H. Chen and P. Mohapatra, "Session-Based Overload Control in QoS-Aware Web Servers", *IEEE INFOCOM 2002, New York, NY*, Jun 2002.

[10] Y. Diao, N. Gandhi, J. L. Hellerstein, S. Parekh, and D. M. Tilbury, "Using MIMO Feedback Control to Enforce Policies for Interrelated Metrics With Application to the ApacheWeb Server", Proc. NOMS 2002, 219-234, Apr 15-19, 2002, Florence, Italy.

[11] L. Kleinrock, Queueing Systems – *Volume 1: Theory*, John Wiley, 1975.

[12] R. Levy, J. Nagarajarao, G. Pacifici, M. Spreitzer, A. Tantawi, and A. Youssef, "Performance Management For Cluster Based Web Services", *IBM Research Technical Report*, RC22676, Dec 2002.

[13] S. H. Low and D. E. Lapsley, "Optimization Flow Control I: basic Algorithm and Convergence", *IEEE/ACM Transactions on Networking*, Vol. 7, No. 6, Dec 1999.

[14] P. Marbach, "Priority Service and Max-Min Fairness", *IEEE INFOCOM 2002, New York, NY*, Jun 2002.

[15] D. Schmidt, "Middleware for Real-Time and Embedded Systems", *Communications of the ACM*, Vol. 45, No. 6, Jun 2002.

[16] Sun Microsystems, *Java Messaging Service API*, http://java.sun.com/products/jms/

[17] S.J. Vaughan-Nichols, "Web Services: Beyond the Hype", *IEEE Computer*, Feb 2002.

[18] T. Voigt, R. Tewari, D. Freimuth, and A. Mehra, "Kernel Mechanisms for Service Differentiation in Overloaded Web Servers", *In Proceedings of the 2001 USENIX Annual Technical Conference, Boston, MA*, Jun 2001.

[19] T. Zhao and V. Karamcheti, "Enforcing Resource Sharing Agreements among Distributed Server Clusters", *Proceedings International Parallel and Distributed Processing Symposium, IPDPS 2002*, Ft. Lauderdale, FL, Apr 2002, pp. 501-510.

[20] H. Zhu, H. Tang, and T. Yang, "Demand-driven Service Differentiation in Cluster-based Network Servers", *IEEE INFOCOM 2001, Anchorage, Alaska*, Apr 2001.

FACILITATING EFFICIENT AND RELIABLE MONITORING THROUGH HAMSA

David Breitgand,[1], Danny Dolev[1], Danny Raz[2], and Gleb Shaviner[1]
[1]*School of Engineering and Computer Science*
The Hebrew University
*Jerusalem 91904, Israel**
{davb, dolev, gleb}@cs.huji.ac.il
[2]*Department of Computer Science*
The Technion, Haifa 32000, Israel†
danny@cs.technion.ac.il

Abstract: Monitoring is a fundamental building block of any network management system. It is needed to ensure that the network operates within the required parameters, and to account for user activities and resource consumption.

In the SNMP paradigm, network management systems have been structured using a two-tier architecture with managers being *thick* clients, and the target agents being *thin* servers. This architecture may be unreliable in times since it depends on the management station having an access to the targets. Network distance between the manager and the network elements also imposes high overhead traffic, large processing overheads, and long control loops. To overcome these drawbacks, distributed network management architectures based on a middleware layer were proposed. However, such approaches suffer both from the need to modify network elements, and the high (and sometime hard to predict) overhead and complexity.

In this paper we study a solution based on a lightweight middleware architecture that aims primarily at improving availability and efficiency of monitoring applications. We describe the *Highly Available Monitoring Services Architecture (HAMSA)*, present its implementation details, and evaluate its performance. Specifically, we demonstrate how the system can be easily deployed and used for several monitoring applications. HAMSA allows a high level of availability and abstraction, with relatively low overhead.

Keywords: monitoring, network management, high availability, middleware, group communication.

1. Introduction and Motivation

The traditional two-tier structure of network management applications based on the rigid client/server paradigm with thick managers and thin target agents suffers from scalability and availability limitations [16].

Among the more prominent problems with this approach are the following.

*This research was supported in part by Intel COMM Grant - Internet Network/Transport Layer & QoS Environment (IXA), and by Check Point PhD Fellowship program, Israel.
†Partial funding provided by the Fund for the Promotion of the Research at the Technion, and the Technion V.P.R. Fund - New York Metropolitan research Fund.

- Since the management agents have limited functionality, and only instrument the access to the management data, all the computations should be performed at the manager. Thus, large volumes of data should be transferred over the network, and the traffic overhead can be high.
- As shown in [7] reactive (*i.e.*, event-driven) monitoring schemes are far more efficient in terms of communication than polling-based ones. However, in standard management frameworks, such as SNMP, configuring application-specific threshold-driven traps is a non-trivial task, and not always possible.
- Manager station becomes a processing bottleneck as the size of the managed network increases.
- Manager station is a single point of failure, which hurts availability of the monitoring services. Although for some types of management data disconnected operation of monitoring can be achieved using RMON [15], the disconnected monitoring operation is not available in the general case. This type of operation, however, is essential for scaling monitoring services, reducing communication overhead, and increasing survivability of management services as explained below.
- The unavoidable network distance between the management station and (some of) the network elements makes it very hard to control the elements, due to the inherent instability imposed by long control loops.

Because of these problems, alternative approaches to monitoring architectures are being pursued [5, 6, 11, 12, 4, 13, 14]. The more important of these emerging approaches and their relationship to our proposal are discussed in Section 5.

One alternative is adopting the popular three-tier architecture. A typical three-tier monitoring application is described in Figure 1b. The target agents constitute the lowest tier and serve as the source of the management data. The monitoring components are dynamically dispatched at the middle tier. They monitor the target agents (and, possibly, communicate with other monitoring components) accumulating and pre-processing the information collected from them. The end-consumers of this information, the management front-ends, constitute the uppermost tier.

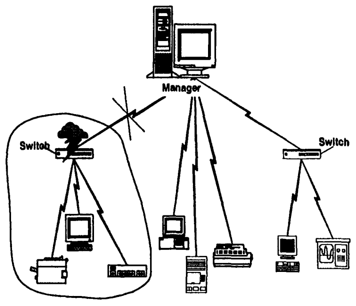


Figure 1a. Two-tier approach.

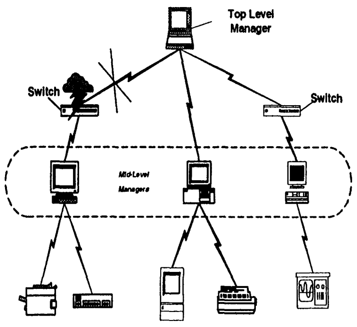


Figure 1b. Three-tier approach

Note that in the three-tier architecture, the components residing in the middle tier can partially or fully implement some of the processing functionality that was previously residing exclusively on the manager side. Thus, using this architecture reduces the traffic overhead, shortens the control loops, and extends the management functionality. In particular, the middle-tier components can implement efficient application-specific event-driven monitoring schemes. Survivability and availability of the network monitoring services are also improved. The mid-tier components can operate autonomously of the first-tier managers (see Figure 1b). When certain parts of the monitored network become unavailable, *e.g.*, due to a network split, the mid-tier monitoring components can continue monitoring in their respective partitions, and later merge the results. This is impossible in the centralized two-tier architecture.

On the down side, the three-tier client/server applications are much more difficult to control. Providing high availability of the mid-tier components in spite of host crashes, network splits and merges is especially challenging.

A standard way of creating a three-tier client/server application is using *application server* that provides the mid-tier run-time environment. However, to the best of our knowledge, no existing application server provides a highly available run-time environment that copes with the kind of failures described above, and can transparently restarting failed *stateful* monitoring components from a consistent point.

Given the complexity of handling distributed three-tier applications in an unpredictable environment prone to various network failures, it is both important and challenging to provide a maximally transparent infrastructure that allows a manager to deploy the needed monitoring components of the second tier in a highly available manner. This improves the overall failure behavior of the management applications, allows more efficient applications (such as event-driven monitoring applications), and therefore contributes to better provisioning of network services in general.

In this paper we propose a novel middleware architecture, HAMSA, that facilitates reliable and efficient deployment of three-tier monitoring applications. We describe the main building blocks of this architecture, and demonstrate it power for efficient and reliable monitoring by describing and analyzing the performance of monitoring applications implemented using HAMSA.

2. Architecture Overview

HAMSA is an architecture that defines the interfaces, and functionality of a highly available run-time environment for HAMSA-Compatible Components. We use the term *components* (as done in many other middleware software systems) to describe objects that implement a set of the predefined interfaces allowing dynamically to mix and match this object with other objects that conform to the same set of interfaces. The following guarantees on the execution of these components in the second tier generate the primary added value of HAMSA.

- Failures of the components are masked from the outside entities. As long as HAMSA has sufficient resources for executing the components, components function *continuously* despite component host failures (*i.e.*, the machine running the component), arbitrary asynchronous network splits, merges, and host recoveries.

- In case of network splits, a single instance of each component is executed per network partition.

- The component whose host machine has failed is guaranteed to resume operation from the last consistent state, *i.e.*, the last state of this component known to the outside world.
- Component's interactions with the environment may potentially influence the state of other components, and/or external entities. In this case, interactions (messages, method invocations) are termed *non-idempotent*. Failure, and a subsequent recovery, of a component being in the middle of a non-idempotent interaction may violate the original interaction semantics. An advantage of HAMSA over other middleware architectures is that it preserves the original *at-most-once*, or *at-least-once* semantic of the component interactions in spite of asynchronous network failures.

HAMSA highly available execution model is not trivial to achieve. Due to the lack of space we do not elaborate on these issues. More details will be provided in the full version of this paper.

These advantages come at a certain price in bandwidth and processing overhead. In Section 4 we discuss the trade-offs between the extended functionality of HAMSA and this overhead. Also HAMSA restricts the inter-process communication model to asynchronous communication and messaging. This communication model, though, fits well into the NM domain.

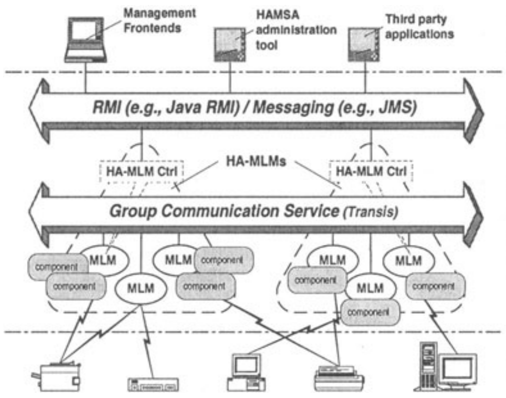

Management Frontends
HAMSA administration tool
Third party applications
RMI (e.g., Java RMI) / Messaging (e.g., JMS)
HA-MLM Ctrl
HA-MLMs
HA-MLM Ctrl
Group Communication Service (Transis)
MLM
component
MLM
MLM
component

Figure 2a. Detailed Logical Structure of HAMSA

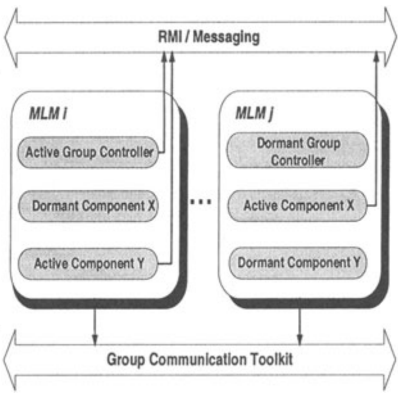


Figure 2b. Logical Structure of HA-MLM

2.1 Highly Available Mid-Level Managers (HA-MLMs)

The run-time environment with the above properties is provided by a set of virtual servers termed *Highly Available Mid-Level Managers (HA-MLMs)*. HA-MLMs are logical entities that are comprised of one or more physical servers called MLMs (see Figure 2a). To its users, every HA-MLM appears as a single object. Its interface is exported by the special component called *HA-MLM Controller*. Each MLM in a given HA-MLM is capable of running the HA-MLM Controller, but at any given moment only a single MLM is executing it. At other MLMs this component is being *dormant*, see Figure 2b.

HAMSA-compatible component is dynamically delegated to a specific HA-MLM through its controller interface using the HAMSA administration tool. The identity of the HA-MLM to which the component is being delegated, is part of the run-time identity of this component.

The executable code of the component is being reliably propagated to all MLMs in the HA-MLM through the group communication service (*e.g.*, Transis [2]), see Section 2.4. However, similarly to the HA-MLM controller component, this component will be executed only at one MLM at any given moment. Other MLMs within the same HA-MLM keep dormant replicas serving as warm backups for the components whose host MLM may fail, see Figure 2b. This is different from the more common practice of keeping components on a component server, and downloading them on demand. HAMSA performs component propagation as above to increase their availability.

The administrator has limited direct control over the physical location of the components within HA-MLM. This is motivated by the fact that components may need to be relocated automatically in case of failures. However, it is possible to influence the HA-MLM placement decisions by supplying some suggestive *policies* that are followed as long as no failures occur.

2.2 HAMSA-compatible Components

To render warm backups of the executing components, MLMs transparently replicate the state of the components delegated to their HA-MLM. To achieve high availability, the state is co-located with the component. This is another difference between HAMSA, and more traditional approaches in which a dedicated database is used to store the state of the components.

The state of a component consists of *Interaction State*, and *Component-Specific State*. Interaction State consists of all inbound and outbound unprocessed interactions between this component and external entities. Component-Specific State consists of arbitrary application-specific objects (*e.g.*, files) that implement a predefined interface. The state objects are managed by the components themselves. The predefined interface allows components to demand state replication without knowing any details of the replication mechanism. Similarly, MLMs handle a component's state with no knowledge of its semantics.

Components may interact with other components executing within the same HA-MLM, in different HA-MLM, and outside HAMSA, *e.g.*, with the front-ends residing in the first tier, and the network elements.

To provide its guarantees, HAMSA requires that all non-idempotent interactions between the HAMSA-compatible components, and any external entities are made through HAMSA Message Service. This allows to replicate the interaction part of a component state transparently.

However, using HAMSA Message Service is feasible only for interactions between the entities of the second and first tiers. It is not feasible to restrict in this way the interactions with the managed devices. Therefore, HAMSA is primarily targeted to monitoring, and not to other kinds of management activities that may change the state of the target devices.

2.3 HAMSA Messaging Service

Allowing a direct interactions of the components with their environment is not always safe. HAMSA defines that each component is assigned an *interaction approver* that policies its interactions. In particular, it may defer interactions with the outside entities depending on the specific state of the network, *e.g.*, when the network splits.

The interactions between the HAMSA-compatible components and the network elements are not restricted in any way. These interactions are supposed to be implicitly reflected in the component-specific state objects, which get replicated on demand from the components.

Each component is given a unique symbolic name consisting of its host HA-MLM name and a unique prefix within this HA-MLM. Each time a component is (re-)activated at an MLM, it updates the external Naming and Directory Service to renew the binding between the component's name and its communication handles. The HA-MLM assigns two types of communication handles for each component: *Mailbox*, and *Proxy*.

Mailbox is needed for directly sending a message to a component. There is one mailbox per HA-MLM that is shared by all components within this HA-MLM, and their clients. In order to support remote method invocations while using the HAMSA consistency and ordering mechanisms transparently, we use the standard proxy approach. Any remote invocation between a HAMSA component and any other party is intercepted, and processed by the per-component proxy. The proxy creates a message from the method call performed on it, and relays it to the HAMSA Messaging Service.

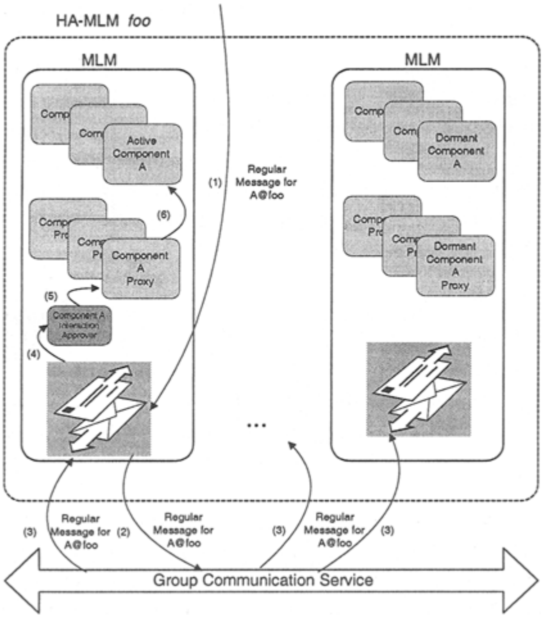

HA-MLM foo
MLM
MLM
Active Component A
Component A Proxy
Component A Interaction Approver
Dormant Component A
Dormant Component A Proxy
(1) Regular Message for A@foo
(2) Regular Message for A@foo
(3) Regular Message for A@foo
(4)
(5)
(6)
Group Communication Service

Figure 3. Interacting through HAMSA Messaging Service

Figure 3 depicts how a message is communicated to a component. The message is placed into the mailbox of this component by the MLM hosting it, (1). This message is not delivered to the target component immediately. Instead, (2), it is being propagated to all MLMs in the HA-MLM using the group communication service (see the next subsection). When the message is received at the group communication service level at all operational MLMs including the sender, (3), this message is assessed for delivery to the target by consulting the component's interaction approver, (4). If the interaction approver permits the interaction, the message is delivered to the active component proxy, (5). Finally, (6), the proxy delivers the message to the target component.

One restriction of this approach is that HAMSA-compatible component cannot support synchronous method invocations with non-void return values. HAMSA communication model requires that if a caller wants to receive information from a component it has to supply either a callback interface or to be registered as a recipient at the mailbox serving this component.

We use the Java programming language to implement HAMSA. We support Java RMI as an instance of the RMI technology. In our prototype, we implemented HAMSA Messaging Service as part of Java Messaging Service (JMS).

2.4 Group Communication Service (GCS)

The replication of the components state within a HA-MLM is facilitated by a group communication toolkit that is not visible outside HA-MLM (see Figure 2a). Such toolkit systems usually allow processes to form *groups* that can be addressed by a single logical name, so that messages can be sent to the group using this name as an address, and all operational members of the group receive them. HA-MLMs are realized in HAMSA as process groups.

- Reliable multicast FIFO delivery of messages.
- Per-group notification of membership changes either due to network failures, or members (*i.e.*, MLMs in the context of HAMSA) voluntarily joining/leaving the group.
- *Virtual Synchrony* model of message delivery, which, simply stated, means that members of the group that go together through the same set of membership changes receive the same set of messages.
- *Partitionable Membership Model* which means that although members of the same group can find themselves in different network partitions (due to asynchronous network splits), each connected component can continue its operation, and when a network merge occurs, the members can resume operation from a consistent point in the message stream so that the Virtual Synchrony model is preserved.

There are many group communication toolkits that supply this functionality [1].

3. Monitoring Applications

In this section we present two typical NM applications, demonstrate how one can deploy them using HAMSA, and explain the benefits NM applications gain from taking the HAMSA approach.

The first NM application is a highly available post-mortem failure analysis system. In this application, several MIB scalar variables from each network element are being kept in a centralized repository, and when a network failure occurs, the management system searches this repository for the relevant variables whose values may suggest the source for the failure (see for example [9]).

In a typical two-tier scenario such a system is deployed at a single station, and the MIB variables of all network elements are accessed from it. The collected data is kept in the local file system. When a failure of a monitored element (or of several elements from the same network region) is detected the collected data is searched and the behavior of the relevant MIB variables is examined in order to identify the cause of the problem.

In HAMSA, the centralized polling application and its repository are being handled transparently by the middleware. The administrator chooses a set of MLMs by either selecting an existing HA-MLM, or defining a new one, and delegates the polling component to this HA-MLM. Based on the component placement policy, the controller activates this polling component at one of the MLMs, while the replicas are kept for warm backup at other MLMs.

If the network splits, the monitoring continues automatically in each network partition where at least one MLM of the split HA-MLM is present. The state (*e.g.*, the collected MIB variables), is kept locally per replica of the monitoring component in each network partition. When the network re-merges these autonomously collected states become available to the administrator.

One question raised by this example concerns different configuration trade-offs available for the monitoring application that uses HAMSA. Consider the typical network configuration illustrated in Figure 4a. In this scenario, the information arrives at the monitoring station from k LANs. If the monitoring is done by centralized polling from the management station, and the connectivity to one of the LANs is lost, the monitoring of its elements cannot continue. In particular, if the failure is caused by a misconfigured access interface in the LAN's access router, the information about the cause of the problem will not be available. This is because the connectivity may be lost before the values of the router's MIB variables suggesting the cause of the problem are retrieved.

If however the administrator configures HA-MLM in such a way that there is at least one MLM per LAN, the MLM in the disconnected LAN will re-start a separate copy of the monitoring as soon as it detects (through the underlying group communication service) that there is a network partition, and all variables in the router's MIBs will be polled.

Once connectivity is re-established (say, through rolling back the configuration) the management station will be able to access this information, and the manager will be able to identify the source of the problem, (*i.e.*, a wrong configuration) and to fix it.

This example also demonstrates the importance of proper HA-MLM configuration. The administrator may be tempted to have at least one MLM in each LAN, as in our example. However, since the state of each monitoring component is distributed by HAMSA to all members of the HA-MLM, the communication costs induced by the replication may become too high.

In fact, one may choose to create k separate applications, each having a different HA-MLM containing only a pair of MLMs, as described in Figure 4b. In this case, the monitoring application for each LAN is running separately on the local MLM (according to the distance-based component placement policy), and thus being unaffected by a possible network partition. If, however, the local MLM itself fails, a copy of the

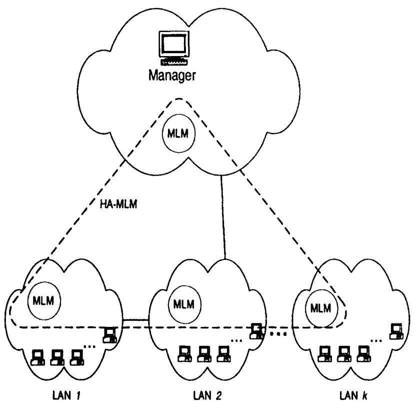


Figure 4a. All MLMs are put into a single HA-MLM

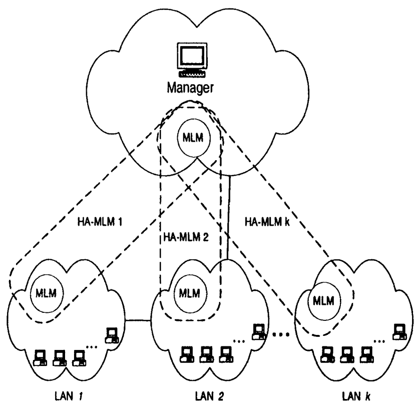


Figure 4b. Pair-wise Organization

monitoring process for that LAN will be initiated automatically by HAMSA on the MLM that is co-located with the management station. This configuration also reduces the overall monitoring traffic when there are no failures, since in this case the monitoring is done locally and the state of the monitoring components is synchronized among the two MLMs only upon the external interactions.

There exists a trade-off between the monitoring overhead traffic, and the overhead traffic induced by HAMSA due to replication it performs behind the scene. The actual amount of overhead depends on the total number of MLMs in a HA-MLM, the size of the application state in HAMSA, the frequency of external interactions, and the amount of data involved in these interactions.

For example, in the described post-mortem failure analysis application, one can choose to have a small state (*i.e.*, the serial number of the last poll), or a very large state (*i.e.*, the actual data of the last 10 minutes polling). Clearly, the latter choice allows a faster recovery after a failure of a monitoring component, but it generates much more overhead traffic. We study these trade-offs in Section 4, and show that the overhead required by HAMSA to provide the extra functionality is much smaller than the monitoring costs we saved.

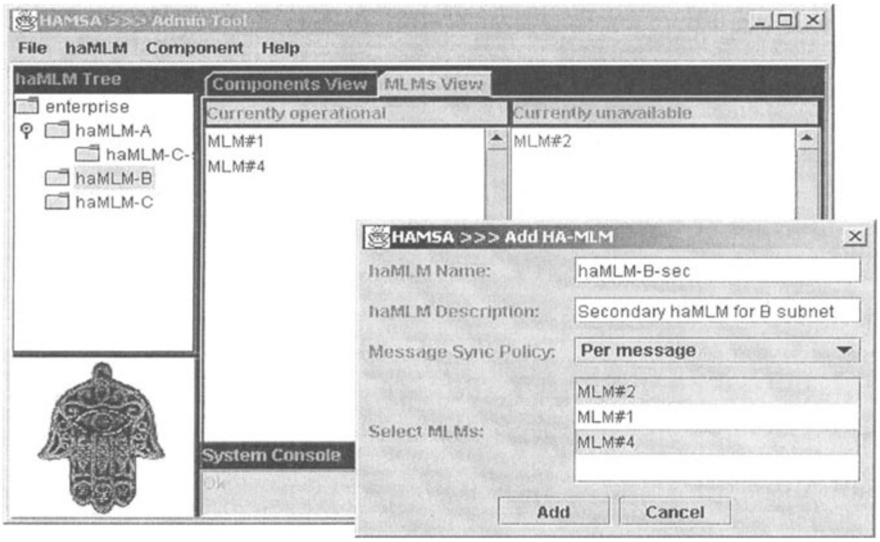
A more complex monitoring application demonstrating the inter-process communication capabilities of HAMSA is an event-driven reactive monitoring NM application. In such an application we are required to detect when a function (typically the sum) of a number of MIB variables, each belonging to a different network element, exceeds a predefined threshold (see [7]).

A centralized realization of this application involves a polling station that monitors all variables at all network elements, computes the function and sets up an alarm if the value has exceeded the threshold. This solution induces both significant traffic overhead and computation load at the monitoring station that grow linearly with the number of polled elements.

To address these issues, several algorithms that combine local computation, traps, and a centralized monitoring station were proposed in [7]. However, in order to deploy these algorithms the agent should be able to carry on simple computation tasks and issue traps, which are in many cases beyond the ability of the standard SNMP trap framework. This is a very good example where the extended functionality of HAMSA can be utilized. The global reactive monitoring application is executed in HAMSA in a distributed way. Namely, a number of copies of the same monitoring component are launched at several HA-MLMs. Each HA-MLM is responsible for its own local set of devices. According to the algorithm of [7], if a local threshold event has been detected (in this case the "local" means "with respect to the local set of variables"), then the other copies of the monitoring component are being notified using HAMSA messaging service. Then according to the algorithm, a global poll may be initiated, and, if needed, an alarm is declared. If one of the local monitoring processes fails, HA-MLM controller restarts it on another MLM, and the system continue functioning. This, of course, comes with a cost of increasing the monitoring traffic, but paying such a cost is definitely better than losing the ability to carry on with the critical monitoring task.

3.1 HA-MLM Administration

HA-MLMs are created through the HAMSA Administration Tool. A screenshot of HA-MLM creation operation is presented in Figure 5.



haMLM Name: haMLM-B-sec
haMLM Description: Secondary haMLM for B subnet
Message Sync Policy: Per message
Select MLMs: MLM#2, MLM#1, MLM#4

Figure 5. Creating new HA-MLM

The administrator picks the MLMs she wishes and groups them into HA-MLMs with a unique name. The same MLM may be a member of different HA-MLMs. A logical hierarchy of HA-MLMs may be formed. The MLMs chosen by the administrator form the *nominal view* of the HA-MLM, as opposed to the *current view*, which is always less or equal to the nominal view due to failures. MLMs that are bundled into a HA-MLM join the process group with the same name. The joining is triggered by the HA-MLM controller that multicasts a JOIN message into the ENTERPRIZE process group specifying the nominal view.

Each time a new member joins, it requests the *state* of the group from someone that is already in the group. The state versions are identified using the pair: $< current\ view >:< epoch\ number >$, where the epoch number is advanced each time the membership changes.

The communications cost involved in creating a new HA-MLM are dominated by the following. A single multicast through Transis is needed to propagate a *join* message. There are at most k membership change notifications delivered by Transis, where k is the size of the nominal view of the new HA-MLM. Finally, there are at most k state exchange messages needed to accommodate each newly joining MLM.

When a HA-MLM already exists, HAMSA compatible components can be delegated to it using the administration tool. The HAMSA components are realized as JAR packages.

The administrator specifies the target HA-MLM, component name, and the component specific parameters, such as the interaction approving policy, and placement policy. Two interaction approval policies are supported currently. One policy is always to approve all interactions. The second policy is to approve external interactions only if the majority of MLMs in HA-MLM are present in the network partition.

Placement policy defines either load-dependent, or distance-dependent placement of a component. These policies are best-effort ones.

The communication cost involved in the component delegation protocol is dominated by multicasting a component through Transis to all MLMs of the HA-MLM, and obviously, depends on the size of the component.

4. Performance Evaluation

In order to understand the trade-off between the communication overhead induced by HAMSA, and the possible reduction in monitoring overhead, consider again the scenario described in figures 4a, and 4b. We want to compare the amount of traffic overhead generated by the monitoring application without HAMSA with the overhead induced by HAMSA and the underlying group communication service.

The group communication service is responsible for failure detection that is based on periodic broadcasting of short *I-am-alive* messages. In general, this overhead grows as k^2, where k being the size of HA-MLM. Optimizations that reduce by factor l, the number of LANs, are possible [3]. However, this is inevitable overhead of failure detection that cannot be strictly attributed to HAMSA or group communication, because any application wishing to achieve the high availability guarantees of HAMSA on its own would pay these costs anyway. The experiments performed in [3] with the current implementation of Xpand and Transis show that group communication scales to 200 hosts dispersed over WAN without visible impact on the regular traffic.

As described in Section 3 the overhead of HAMSA itself strongly depends on the way we configure HA-MLMs and on the size of the application *state*. In order to investigate the trade-off, assume that the state size sent by HAMSA is 150 bytes. This is a reasonable size, when one chooses to use a small state (like a measurement sequence number).

Figure 6a depicts the tradeoff for the two choices of HA-MLM configuration and for 10, and 20 scalar MIBs variables in each LAN. We assumed here that due to the SNMP encoding (SMIv1/SMIv2), polling of one variable takes about 150 bytes, and thus polling 10 or 20 variables per LAN will consume 1500, and 3000 bytes respectively for each LAN. HAMSA's overhead depends on the HA-MLM configuration. If all k MLMs are members of the same HA-MLM, we need to update all k states each polling interval. This takes $150 * k^2$ bytes. On the other hand, if we use k different HA-MLMs, each of size 2, HAMSA's overhead is reduced to $150 * k$.

One can easily see that even for a very small number of monitored variables the overhead of HAMSA is significantly smaller than the monitoring overhead of a traditional application. This is a big advantage even without considering the HAMSA's main goals: extended functionality and reliability.

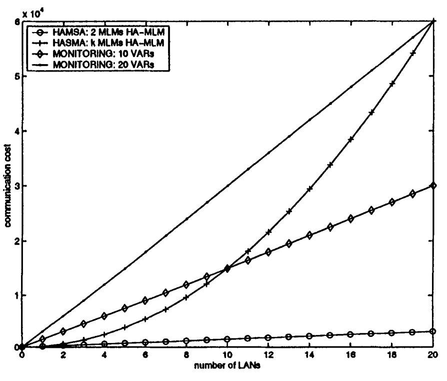


Figure 6a. HAMSA communication cost, and monitoring communication cost as a function of the number of LANs.

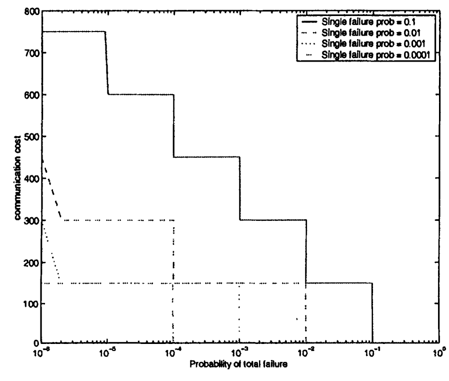


Figure 6b. HAMSA communication cost per state change as a function of the required system error failure probability.

The main mission of HAMSA is increasing the system reliability. However, the high reliability comes with the cost of introducing more MLMs. In particular, this implies a higher communication overhead. Thus, there is a trade-off between the level of availability and the traffic overhead. In order to evaluate this trade-off, we consider the same scenario as above.

The current host MLM of an active component replica propagates its state to the rest of its HA-MLM through multicast. Thus, communication cost of HA-MLM replicating the state is linear in the number of group members. However, when we increase the number of MLMs in the group, we reduce the probability of a total system failure, since HA-MLM restarts the failed process on a different MLM as long as they are available.

Thus, if we have n MLMs in a HA-MLM, and the independent probability of a single MLM failure is p, the probability of the application failure is $\tilde{p} = 1 - p^n$. Since we have only one active component per network partition then the communication is $s * (n - 1)$ per each state in the component state, where s being the component state size. To obtain specific numbers, let $s = 150$ bytes, as in our example. Then, in order to get an application failure probability of $\tilde{p}$ we pay $150(\lceil \frac{\log(1-\tilde{p})}{\log p} \rceil - 1)$ bytes per change. This cost is plotted in Figure 6b, for single MLM failure probability of 0.1, 0.01, 0.001, and 0.0001.

As one can see, in order to get the often desired "5 nines" reliability, starting from a very high error rate of 0.1 on a single machine, 6 MLMs are sufficient. The communication cost ($150(6-1) = 750$ bytes per state change) becomes much smaller when the reliability of a single machine increases.

5. Background and Related Work

The quest for more efficient and versatile management paradigms has been pursued by many researches over the last few years. One general line of approach suggests using mobile agents, active networks, or programmable networks for decentralizing and shortening the control loop [4, 13]. Usually, these proposals focus on the mechanics of the mobility and extended functionality rather than on the high availability and meta-management issues being in the focus of this paper.

Several approaches for integrating the management by delegation approach [17] into SNMP environment have been proposed recently [10]. With the advent of Java, the delegation is easily implemented by exploiting its mobility and security features making Java a preferred language for developing delegated programs.

Java Management Extension (JMX) [11] is an emerging Java standard for representing managed objects as Java Beans. JMX Bean is an object that serves as a Java wrapper facade for the actual managed object. JMX Beans may be co-located with the objects they represent at the agent side, or be deployed in a distributed fashion. In the latter case, JMX Beans need distributed object services of the second tier that are currently left unspecified by JMX. HAMSA components can be implemented as JMX Beans.

One of the more mature Java technologies for deploying three-tier Java applications is provided by Enterprise JavaBeans (EJB) [8]. EJB defines interfaces for Application Server, and Enterprise Java components (Beans) that execute in the environment of the application server that manages transactions, persistency, security, and naming services for the components.

The problems that HAMSA copes with are very similar to those of the stateful EJB clustering. Some of the existing EJB implementations provide fail-over models that allow replication of the beans' states, and support takeover of the failed beans by other servers in the cluster [8]. Most EJB servers perform stateful fail-over by using either in-memory replication, or persistent storage to a shared database. These solutions are inappropriate for the NM domain, since they rely on the fact that the network remains connected. To the best of our knowledge, there is no current implementation of EJB, or other application server technology that provide the high availability of the second-tier components execution to the level that allows their comparison with HAMSA.

6. Conclusion and Future Work

Efficient monitoring of large and dynamic distributed systems becomes challenging. Current standard technologies scale poorly due to their inherently centralized approach. We present a lightweight monitoring middleware called HAMSA that dynamically allows to enhance monitoring functionality, and decentralize it in a reliable and efficient manner. This work presents the architectural overview of the middleware, and the possible functional and performance trade-offs involved in its deployment. Our architecture uses a group communication middleware to increase availability, modularity, and scalability.

We are currently testing our implementation under different load conditions, and for different failure scenarios. The exetensive performance evaluation study will be presented in the full version of the paper.

7. Acknowledgments

We thank Elias Procopio Duarte, Jr, and Aldri L. dos Santos for their valuable comments.

References

[1] ACM. *Communications of the ACM, special issue on Group Communication Systems*, 39(4), April 1996.

[2] Y. Amir, D. Dolev, S. Kramer, and D. Malki. Transis: A communication sub-system for high availability. In *22nd Annual International Symposium on Fault-Tolerant Computing*, july 1992.

[3] T. Anker, G. Chockler, D. Dolev, and I. Shnaiderman. The design of xpand: A group communication system for wide area. Technical Report HUJI-CSE-LTR-2000-31, The Hebrew University, July 2000.

[4] Andrzej Bieszczad, Bernard Pagurek, and Tony White. Mobile agents for network management. *IEEE Communications Surveys*, 1(1):2–9, Forth Quarter 1998.

[5] D. Breitgand, G. Shaviner, and D. Dolev. Towards highly available three-tier monitoring applications (extended abstract). `http://cs.huji.ac.il/~davb/abstracts/hamsa.ps`, 2000. 11th IFIP/IEE Internationaal Workshop on Distributed Systems: Operations and Management, Austin TX, USA.

[6] M. Daniele, B. Wijnen, M. Ellison, and D. Francisco. Agent extensibility (AgentX) protocol, January 2000. RFC 2741.

[7] Mark Dilman and Danny Raz. Efficient reactive monitoring. *IEEE Journal on Selected Areas in Communications (JSAC), special issue on recent advances in network management*, 20(4):668–677, May 2002.

[8] Roman E., Ambler S., and Jewell T. *Mastering Enterprise JavaBeans(tm)*. Wiley, 2nd edition, 2002.

[9] E. P. Duarte Jr. and Aldri L. dos Santos. Semi-active replication of snmp objects in agent groups applied for fault management. In *7th IEEE/IFIP International Symposium on Integrated Network Management IM*, Seattle, WA, May 2001.

[10] D. Levi and J. Shonwalder. Definitions of Managed Objects for the Delegation of Management Scripts, May 1999. RFC 2592.

[11] Sun Microsystems. Java management extensions(JMX) instrumentation and agent specification, v1.1. `http://java.sun.com/products/JavaManagement/doc.html`, mar 2002.

[12] B. Pagurek, Y. Wang, and T. White. Integration of mobile agents with SNMP: Why and how. In *2000 IEEE/IFIP Network Operations and Management Symposium*, pages 609 – 622, Honolulu, Hawaii, USA, April 2000.

[13] Danny Raz and Yuval Shavitt. Active networks for efficient distributed network management. *IEEE Communications Magazine*, 38(3), March 2000.

[14] Marcelo Gonçalves Rubinstein and Otto Carlos Muniz Bandeira Duarte. Evaluating tradeoffs of mobile agents in network management. *Networking and Information Systems Journal*, 2(2):237–252, 1999. HERMES Science Publications.

[15] William Stallings. *SNMP, SNMPV2, SNMPV3, and RMON 1 and 2*. Addison-Wesley, January 1999.

[16] Y. Yemini. The OSI Network Managemnt Model. *IEEE Communications Magazine*, pages 20–29, may 1993.

[17] Yechiam Yemini, German Goldszmidt, and Shaula Yemini. Network Management by Delegation. In *The Second International Symposium on Integrated Network Management*, pages 95–107, Washington, DC, USA, April 1991.

DYNAMIC LOAD BALANCING FOR DISTRIBUTED NETWORK MANAGEMENT

Kiyohito Yoshihara

Manabu Isomura

Hiroki Horiuchi
KDDI R&D Laboratories Inc.,
2-1-15 Ohara Kamifukuoka-shi
Saitama 356-8502, Japan
yosshy@kddilabs.jp, isomura@kddilabs.jp, hr-horiuchi@kddilabs.jp

Abstract: The scalability limitations of centralized management models have motivated distributed management models, in which management programs describing some of management tasks are distributed and executed on managed systems. In the models, management program distribution that considers dynamic network resource utilization is one of the most important challenges, in striking a load balance between management and managed systems for an entire managed network. Some methods for load balancing have been studied; however, they cannot adequately be achieved throughout an entire managed network. This arises from criteria for load balancing that lacks dynamic network resource utilization, or from a localized subnetwork in which the performance is limited, although it does include processing loads for dynamic network resource utilization. To solve this, a new dynamic load balancing method is proposed for distributed network management. Thus, systems that execute management programs are decided dynamically on the basis of CPU utilization for each system and the bandwidth required for executing all management programs. Two typical algorithms derived from the proposed method, each having different criteria in the form of mean deviation and range types with respect to CPU utilization, are introduced. They were evaluated analytically according to capability, i.e., how well they perform as close to load balancing as possible, as well as time complexity. The results show that the mean deviation type algorithm performs better at almost the same computational cost. A prototype system is also implemented based on the proposed method, and evaluated empirically by applying it to an operational LAN. The proposed method performs well in trials with a trivial overhead.

Keywords: Distributed network management, load balancing, management by delegation, mobile agent, SNMP

1. Introduction

Centralized management models, such as SNMP (Simple Network Management Protocol) and TMN (Telecommunications Management Network), address scalability limitations, since management tasks, including management information-gathering, information analysis and results-oriented system controls are focused on a centralized management system, as well as increasing numbers of managed systems resulting in management traffic overhead [13, 9, 7, 8, 11, 12, 2, 14, 4, 1]. These are the motivated distributed management models. A typical management model is called Management by Delegation (MbD) [13, 9, 7, 8, 11], in which management programs describing some of management tasks are executed on managed systems such as routers and switches. Another model is management by mobile agents (MA) [12, 2, 14, 4, 1], in which mobile agents with some management tasks migrate to/from management and managed systems. They provide a means to enhance scalability by distributing some management tasks to the managed systems, and by reducing the management traffic overhead with appropriate filters at the managed systems.

In the distributed network management, dynamic load balancing across an entire managed network is one of the most important challenges in which the management programs should be distributed and executed on management and managed systems, when considering dynamically changing network configurations and resource utilization. In a theoretical sense, the problem of finding an optimal subset of systems on which the management programs should be distributed, such that a given objective function associated with resource utilization is minimized, is known as a "p-center" or a "p-median" problem, and both are NP-hard [3]. Some approximation methods do not always provide optimal solutions but offer practical, acceptable solutions, which have recently been studied [6, 5]. Yet, they cannot adequately perform dynamic load balancing. Since the criteria for load balancing is the number of management programs and is not associated with system processing load, the method [6] cannot perform dynamic load balancing in terms of network resource utilization. Although another criterion is associated with the processing load, the method [5] can be performed only within a localized subnetwork and cannot strike a load balance across an entire managed network.

For a solution to this, this paper proposes a new dynamic load balancing method for distributed network management. It allows us to strike a load balance between management and managed systems across an entire managed network, in considering the following two criteria: 1) processing load of the management and managed systems, and 2) bandwidth required for executing all management programs. A typical example of an entire managed network is a set of subnetworks connected together with links like Ethernet and managed by a single organization. The proposed method is realized by attempting to limit the value of a given function in terms of CPU utilization of systems within a small specified value, as well as limiting bandwidth required to execute all management programs within another specified value. In terms of functions, two typical algorithms based on the proposed method are introduced, each having different functions: 1) mean deviation function, and 2) range function. Mean deviation function is defined as the absolute difference between the CPU utilization of each system and the average CPU utilization for all systems, while range function is defined as the difference between the maximum and minimum CPU utilization for all systems.

The algorithms are evaluated analytically according to: 1) setting appropriate limiting values for effective load balancing, 2) capability of two algorithms in performing load balancing and 3) time complexity of two algorithms. Next, a prototype system is implemented based on the proposed method with the stronger algorithm, and evaluated empirically by applying the system to an operational LAN according to: 1) deriving suitable limiting values used in the method, and 2) whether the proposed method can perform well using the derived value, with trivial overhead.

The rest of the paper is organized as follows: Section 2 presents an overview of the distributed network management. It describes two existing methods and addresses their disadvantages with respect to dynamic load balancing in Section 3. In Section 4, a new dynamic load balancing method is proposed, and a prototype system is implemented in Section 5. Two typical algorithms derived from the proposed method are evaluated analytically while evaluating the proposed method in an operational LAN empirically in Section 6.

2. Overview of Distributed Network Management

The distributed management model assumes a managed system such as routers and switches with sufficient computational resources to execute management programs or mobile agents. As a mobile agent may be viewed as a management program capable of traversing one or more managed systems in a specified order, this section describes an overview of how the management program is deployed below.

A management system distributes a management program describing some of management tasks and delegates the task to a managed system as shown in Figure 1(1). Typical management programs sometimes collect management information from MIB (Management Information Base) in the managed system, by polling limited to within the managed system (2). Others aggregate, analyze or filter collected information (3),

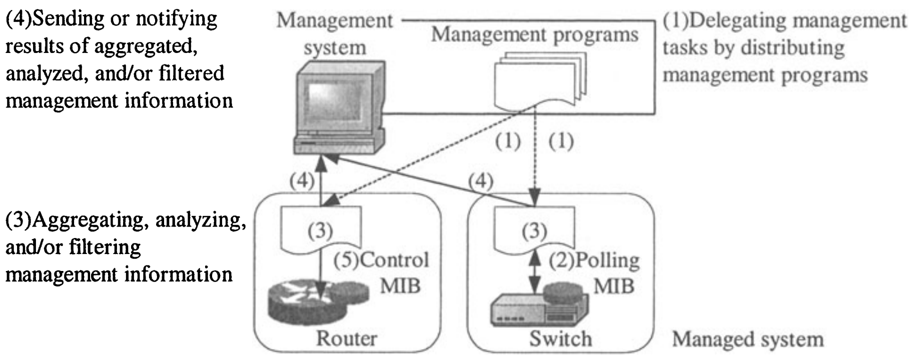

(4)Sending or notifying results of aggregated, analyzed, and/or filtered management information
Management system
Management programs
(1)Delegating management tasks by distributing management programs
(1)
(1)
(4)
(4)
(3)Aggregating, analyzing, and/or filtering management information
(3)
(5)Control MIB
(3)
(2)Polling MIB
Router
Switch
Managed system

Figure 1. Overview of distributed network management

send the information or notification to the management system (4) and control the managed systems according to the results (5). The distributed network management provides a promising means to improve scalability of centralized management models, typically through the distribution of tasks and by reducing management traffic overhead.

3. Existing Methods and Disadvantages

In the distributed network management, dynamic load balancing is a key challenge. For this purpose, some methods have been studied as described below.

3.1 Existing Methods

3.1.1 Method Based on Number of Management Programs. This method [6] performs load balancing such that the number of managed systems assigned to each management program may be almost the same when the number of management programs has been changed by executing a new management program and by terminating existing management programs, or on a regular basis.

When a new management program is executed, this 'leader' management program directs an entire load balancing process. First, the leader collects information required for the process from all management programs, including distances between each management program and the leader, and identifiers of managed systems that each management program currently manages. Next, the leader determines managed systems to be assigned to the leader in order of proximity, until the number of managed systems assigned to each management program may be almost the same. Then, the leader determines managed systems to be assigned to other management programs in the same manner. Finally, the leader makes notification on process termination with information covering all changes.

3.1.2 Method Based on Number of Managed Systems. In the method [5], an entire managed network is divided into some subnetworks such that the number of managed systems in each subnetwork is less than or equal to a specified threshold. A management program is deployed per subnetwork and load balancing is also performed per subnetwork.

When a new managed system is attached to a subnetwork and the number of managed systems in the subnetwork exceeds the threshold, this subnetwork is divided into two subnetworks. A new management program is executed on a managed system in one subnetwork, while the existing management program remains on a managed system in another subnetwork. The two management programs manage each subnetwork independently.

The management program can move to another managed system a within a subnetwork it is responsible for from their current managed system b, if $U_a < (1 - r)$

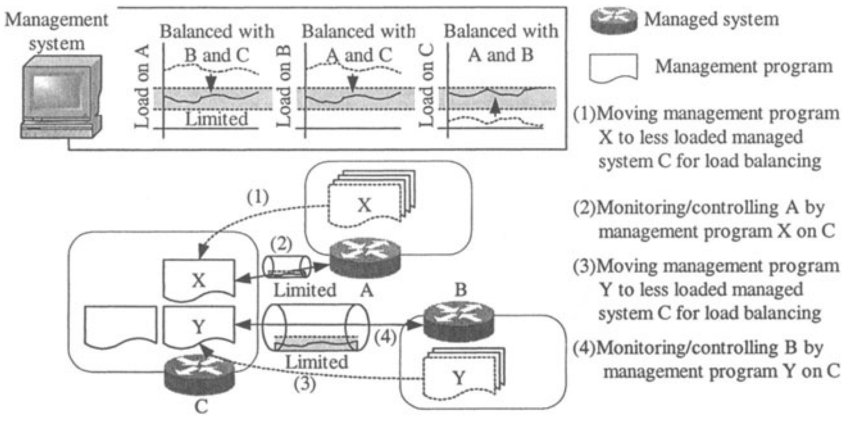

Management system
Balanced with B and C
Balanced with A and C
Balanced with A and B
Load on A
Load on B
Load on C
Limited
Managed system
Management program
(1)Moving management program X to less loaded managed system C for load balancing
(2)Monitoring/controlling A by management program X on C
(3)Moving management program Y to less loaded managed system C for load balancing
(4)Monitoring/controlling B by management program Y on C
X
Y
A
B
C
(1)
(2)
(3)
(4)
Limited

Figure 2. Principle of proposed method

U_b, where U_x denotes a linear function of CPU and memory utilization of managed system x, and $r \in (0,1)$ is constant. CPU and memory utilization are collected by using another probe management program in the subnetwork.

3.2 Disadvantages

The first method based on the number of managed programs certainly takes note of network configurations such as distance, yet it cannot perform dynamic load balancing sufficiently as load balancing criteria is not at all associated with dynamically changing network resource utilization such as processing load of management and managed systems.

The second method based on the number of managed systems performs dynamic load balancing to a certain extent, in considering both network configuration and resource utilization; however, the method performs only within a localized subnetwork and thus, cannot strike a load balance across an entire managed network. Even if the method could be applied to an entire managed network, the function would be too simple to perform closer load balancing where the absolute differences between CPU utilization of any two systems should be small, as will be analyzed in Section 6.

For a solution to this, a new dynamic load balancing method is proposed for the distributed network management below, which incorporates dynamically changing network configuration and resource utilization, and strikes a load balance between systems across an entire managed network.

4. Proposed Method

4.1 Design Principle

4.1.1 Coexistence of Distributed and Centralized Management Models. As described in Section 2, with respect to processing load associated with management tasks on management and managed systems, as well as management traffic overhead, the distributed management model complements the centralized one where management tasks including management information gathering via polling are concentrated to a management system, while processing load associated with management tasks on managed systems can be minimized. In the proposed method, the distributed and centralized management models coexist, and load balancing is achieved by making full use of both models.

As shown in Figure 2, management programs are executed not only on direct target managed systems to be monitored or controlled, but also on less loaded other systems, including management systems. Suppose the management programs X and Y are responsible for monitoring managed systems A and B, respectively. If the managed system C is loaded less than A, the method moves X from A to C for load balancing (Figure 2(1)). After that, X resumes monitoring their direct target managed system A from C by polling or with management operations via a network similar to management systems in the centralized management model (2). If managed system C is still loaded less than B, the method moves Y from B to C (3). Y resumes monitoring their direct target managed system B from C in the same manner as X (4).

4.1.2 Processing Load of Management and Managed Systems as First Criteria for Load Balancing. The CPU utilization associated with processing load of management and managed systems is used as the first criteria for load balancing. A new threshold of CPU utilization is introduced to detect a system overload. The proposed method achieves load balancing by limiting a value of a given function in terms of CPU utilization within a given threshold as shown in Figure 2. A variety of functions, from simple to complex ones, are possible. Two typical and basic functions will be defined in Section 4.2.1.

4.1.3 Bandwidth Required for Executing all Management Programs in Managed Network as Second Criteria for Load Balancing. Some management programs may be executed on a less loaded, indirect target managed system for monitoring their direct target managed system by polling or management operations through a network in the proposed method. This may cause additional management traffic. The bandwidth required for executing all management programs in a managed network is used as the second criteria to prevent the traffic. A new bandwidth threshold is introduced, and the proposed method is realized by limiting the bandwidth to within the given threshold as shown in Figure 2

4.1.4 Load Balancing by Management System. The management system is responsible for an entire load balancing process in the proposed method in order for load balancing to occur across an entire managed network. The system collects information such as system processing loads, updates information, determines management programs to be moved and destination systems when system overloads are detected, and performs load balancing in accordance with the determination. Leaders may be changed in every process as described in Section 3.1.1, though the process may become more complicated due to information and status synchronization between management programs.

4.2 Dynamic Load Balancing by Proposed Method

4.2.1 Defining Criteria for Load Balancing. The proposed method uses 1) processing load of management and managed systems, and 2) bandwidth required for executing all management programs as criteria for load balancing. The following two basic functions from each type are defined for the first criteria associated with processing load as the variability indices of system CPU utilization can be mainly classified into deviation and range types. For the sake of simplicity, the computational power of all nodes is assumed as the same in the following definitions. Note that it may be better to use stronger nodes over weaker ones. Definitions should be changed in this instance. A typical modification is weighted CPU utilization multiplied by some factor depending on the calculation power.

Mean Deviation Function. Let $c_{i,T}$ (%) be the average system CPU utilization i $(1 \leq i \leq N)$ over T seconds, where N denotes the number of management and managed systems. As provided by Equation (1), the mean deviation function $\epsilon_{i,T}$ is defined as the deviation of $c_{i,T}$ from the average (mean) value of all $c_{j,T}$'s $(1 \leq j$

$\leq N$) systems. Note that the average for all CPU utilization or maximum utilization should be the CPU utilization for a multi-CPU system.

$$\epsilon_{i,T} \overset{\text{def}}{=} \left|c_{i,T} - \left(\Sigma_{j=1}^{N} c_{j,T}\right)/N\right| \quad (\%). \tag{1}$$

Range Function. As given in Equation (2), the range function is defined as the difference between the maximum and minimum CPU utilization for all systems.

$$\delta_T \overset{\text{def}}{=} \max_i c_{i,T} - \min_i c_{i,T} \quad (\%). \tag{2}$$

If the proposed method cannot make determination with only two criteria, priority is assigned to each management program by a network operator.

4.2.2 Determining Management Program and Destination System. Two thresholds are introduced here. One is tolerable variation Δ (%) and another is tolerable bandwidth B (bps). The proposed method attempts to limit all $\epsilon_{i,T}$s' or δ_T within Δ, as well as bandwidth required for executing all management programs within B. They are specified by a network operator. How to set appropriate values of these thresholds for effective load balancing will be evaluated in Section 6.1.1. For the sake of simplicity, the rest of this section focuses attention on $\epsilon_{i,T}$, and the same holds for δ_T.

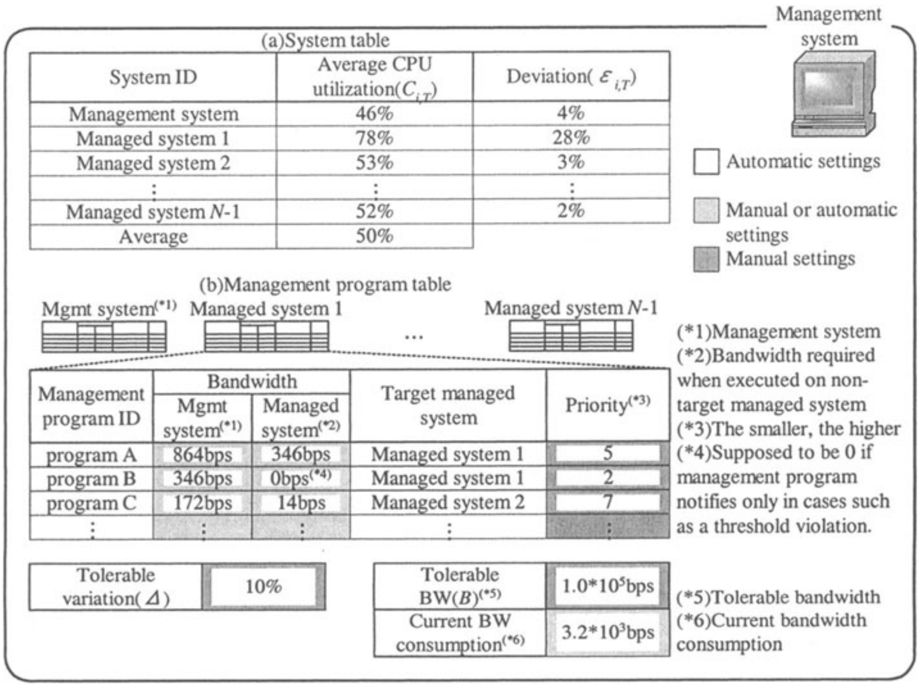
As candidates for load balancing, the proposed method first selects management programs on the most loaded system i where $\epsilon_{i,T} > \Delta$ and $c_{i,T}$ is the maximum for all the systems. Next, the proposed method determines the system j with a minimum $c_{j,T}$ as the destination system, as well as the management program to be moved from the candidates for load balancing, such that the decrease in bandwidth required for executing all management programs after moving towards the destination system is maximized. If there is no such management program, it determines one where the increase is minimized.

The management system notifies the network operator and terminates the execution of the management program with the lowest priority on the most loaded system if any management program movement results in excess bandwidth over the tolerable bandwidth B.

4.2.3 Performing Load Balancing. The management system moves and resumes the management program to the destination system in accordance with the determination as described in Section 4.2.2. The management system notifies a network operator of a failure if move or resume fails for some reason. The management system updates information required for a subsequent load balancing process such as System table, Management program tables and current bandwidth consumption as described in Section 4.2.4.

4.2.4 Managing Information for Load Balancing. Figure 3 shows all information required for load balancing managed by the management system, including (a) System table, (b) Management program tables, tolerable variation Δ, tolerable bandwidth B and current bandwidth consumption.

System table maintains average CPU utilization for each system and derives mean deviation $\epsilon_{i,T}$ from utilization. The management system distributes the CPU monitoring program to each system to monitor CPU utilization. The management system sets $c_{i,T}$ to System table obtained by CPU monitoring programs. The Management program table is provided for each system to maintain information on management programs executed on the system. It includes the identifier of a management program executed on the system, the bandwidth required when executed on the management



(a)System table

System ID	Average CPU utilization($C_{i,T}$)	Deviation($\varepsilon_{i,T}$)
Management system	46%	4%
Managed system 1	78%	28%
Managed system 2	53%	3%
:	:	:
Managed system N-1	52%	2%
Average	50%	

(b)Management program table

Mgmt system(*1) Managed system 1 ... Managed system N-1

Management program ID	Bandwidth		Target managed system	Priority(*3)
	Mgmt system(*1)	Managed system(*2)		
program A	864bps	346bps	Managed system 1	5
program B	346bps	0bps(*4)	Managed system 1	2
program C	172bps	14bps	Managed system 2	7
:	:	:	:	:

Tolerable variation(Δ)	10%

Tolerable BW(B)(*5)	$1.0*10^5$bps
Current BW consumption(*6)	$3.2*10^3$bps

Management system
□ Automatic settings
▨ Manual or automatic settings
■ Manual settings
(*1)Management system
(*2)Bandwidth required when executed on non-target managed system
(*3)The smaller, the higher
(*4)Supposed to be 0 if management program notifies only in cases such as a threshold violation.
(*5)Tolerable bandwidth
(*6)Current bandwidth consumption

Figure 3. Operational information and settings required for load balancing

system and on managed systems, the direct target managed system for a given management task and priority.

The current bandwidth consumption is summed up by the bandwidth consumed by all management programs executed, which is maintained by Management program tables. The bandwidth consumed by each management program is manually or automatically derived. For example, if a management program uses a specific management protocol such as SNMP, the bandwidth can be roughly derived from the polling interval and the expected length of PDU (Protocol Data Unit), including the number of pieces of management information, their syntax and expected returned values.

4.3 Example

We show how the proposed method performs load balance below, with operational information and settings in Figure 3 and the flowchart in Figure 4.

The tolerable variation Δ and tolerable bandwidth B are set 10% and $1.0 * 10^5$ bps, respectively (Figure 4 S1). The CPU monitoring program is executed on each system (S2). Then, the method configures System table, as well as Management program tables (S3), and distributes management programs on direct target systems and executes them (S4). The method attempts to perform load balancing, since $\epsilon_{1,T}$ (= 28%) > Δ (= 10%) (S5). The method selects management programs on the most loaded system 1 as the load balancing target system (S6). The method determines the management system with the minimum average CPU utilization as a destination system (S7) and management program C as the load balancing target, resulting in a maximum 14 bps (from 172 + 14 bps to 172 bps) decrease in bandwidth consumption (S8). Since the resulting bandwidth consumption does not yet exceed the tolerable bandwidth B (S9), the method performs load balancing by oving the management program C from sys-

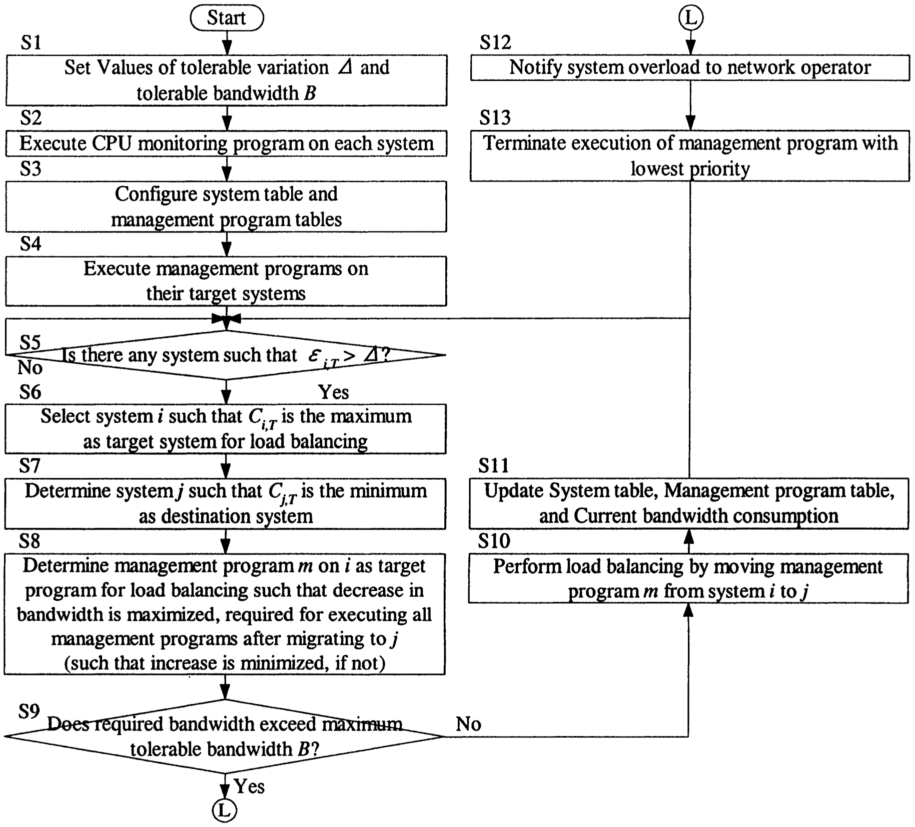

Start
S1
Set Values of tolerable variation Δ and tolerable bandwidth B
S2
Execute CPU monitoring program on each system
S3
Configure system table and management program tables
S4
Execute management programs on their target systems
S5
Is there any system such that ε_{i,T} > Δ?
No
S6
Yes
Select system i such that C_{i,T} is the maximum as target system for load balancing
S7
Determine system j such that C_{j,T} is the minimum as destination system
S8
Determine management program m on i as target program for load balancing such that decrease in bandwidth is maximized, required for executing all management programs after migrating to j (such that increase is minimized, if not)
S9
Does required bandwidth exceed maximum tolerable bandwidth B?
No
Yes
L
S12
Notify system overload to network operator
S13
Terminate execution of management program with lowest priority
S11
Update System table, Management program table, and Current bandwidth consumption
S10
Perform load balancing by moving management program m from system i to j

Figure 4. Flowchart of proposed method

tem 1 to the management system (S10). Finally, the method returns the process to S5 after updating the tables and current bandwidth consumption (S11).

If the resulting bandwidth consumption exceeds the tolerable bandwidth B (S9), the method does not move management program C while notifying the network operator of system overloads (S12). The method terminates execution of the management program with the lowest priority on the system (S13), and returns the process to S5.

5. Prototyping

A prototype system is implemented in Java. Currently, minimum functions sufficient for evaluating the proposed method are implemented. Figure 5 shows the system diagram. The system runs on a PC, which emulates a managed system with sufficient computational resources for a management program. We use legacy SNMP agent, and AdventNet, Inc. SNMPv3 Package2.2 for Java API to the SNMP agent. We give MIB definition for initial settings. There can be many varieties of management programs with different management tasks, and two typical management programs [11] are implemented, in addition to the CPU monitoring program as in Table 1.

These management programs can be generated easily by specifying a few parameters (Figure 5 (1)). The management application then approximates the bandwidth required for executing each management program and registers them with Management program tables in Figure 3 (b). The download and unload, start, suspend and resume of management program execution are realized by Java RMI (Remote Method Invocation). When execution starts (3), polling to the legacy SNMP agent is performed

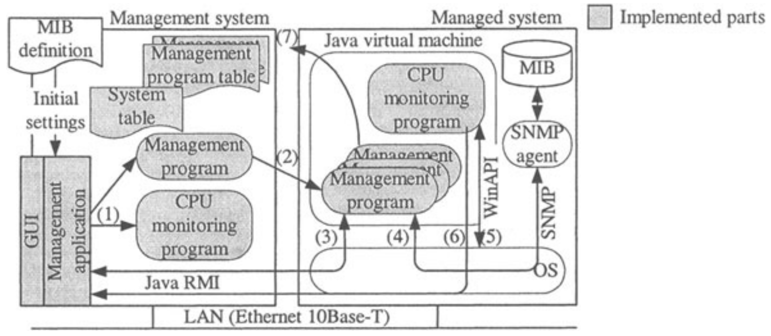


Figure 5. Prototype system diagram

Table 1. Management programs in prototyping.

Management program	Description	Parameters (not exhaustive)
A	notifies threshold violation to management system on result of polling	direct target managed system, names of management information for polling[a], polling interval[b], upper and lower thresholds[b]
B	sends aggregated management information collected by polling for every specified time to management system	direct target managed system, names of management information for collection[a], polling interval[b], sending interval[b]
CPU monitoring program	sends average CPU utilization collected by polling for every specified time to management system	direct target managed system, polling interval, sending interval

[a]Possible to specify multiple names.

[b]Possible to specify a different value for each piece of management information.

according to the specified parameters (4). Threshold violations and aggregated management information are also notified and sent by RMI (3). As CPU utilization is not defined in a standard MIB, it is collected through OS-dependent API (5) and sent by RMI (6). If heavy processing load on a system is detected, management program execution is suspended and the management program is moved to another less loaded system with its execution context (7).

6. Evaluations

6.1 Analytical Evaluations

This section evaluates two algorithms A_d and A_r, each obtained by applying Eq.(1) and Eq.(2) to the proposed method according to: 1) appropriate tolerable variation value Δ and tolerable bandwidth B for effective load balancing in Section 6.1.1, 2) capability of algorithms on their ability to perform load balancing where the sum of the absolute differences between CPU utilization of any two systems should be small in Section 6.1.2, and 3) time complexity of Eq.(1) and Eq.(2) in Section 6.1.3.

6.1.1 Tolerable Variation Δ and Bandwidth B.

Tolerable Variation Δ. The smaller the value of tolerable variation Δ, the smaller the absolute difference between CPU utilization of any two systems $c_{i,T}$ and $c_{j,T}$ $(1 \leq i, j \leq N, i \neq j)$, thus the algorithms can more closely strike a load balance among systems across an entire managed network. However, if the value of Δ is set too small, even a slight difference between two systems may trigger a load balancing process causing undesirable effects where the algorithms continue to move a management program from system to system. The optimal value of Δ preventing the effect should be the minimum one satisfying the following two conditions simultaneously.

Condition1	The value of Δ is greater than the maximum processing load of all management programs for all systems.
Condition2	The value of Δ is greater than the maximum processing load of all destination systems when a management program moves there.

Assuming a case where only a single management program is executed on a system across an entire network, and the average CPU utilization of all other systems is 0%, unless Condition 1 holds, the execution of the management program on any system triggers a load balancing process and causes the effect. It can be proven that Δ satisfying Condition 1 for the above case can prevent the effect in any other case. For algorithms A_d and A_r, such Δ is provided by Equation (3), where N and $c_{\max}$ denote the number of systems and maximum processing load of all management programs for all systems, respectively.

$$\Delta > \begin{cases} c_{\max}(1 - 1/N) & \text{for } A_d, \\ c_{\max} & \text{for } A_r. \end{cases} \tag{3}$$

Unless Condition2 holds, the increasing processing load of the destination system in moving a management program for the load balancing triggers another load balancing process and this also causes the effect. This can be avoided by a time window, for which the method is prohibited from a resulting load balancing process.

Tolerable Bandwidth B. Since user traffic and management traffic share bandwidth in LAN and WAN networks, it is generally desirable that the bandwidth for management traffic be restricted at most to 5% of the minimum bandwidth of the network [10]. Accordingly, for example, B is set to 100 Kbps, which is 5% of the effective bandwidth 2 Mbps of a 10Mbps Ethernet LAN.

6.1.2 Close Load Balancing Capability. Here, an evaluation function defined by Eq.(4) is introduced to show how closely the algorithms A_d and A_r can perform load balancing. The smaller the value of the evaluation function is, the closer load balancing an algorithm can perform, where the absolute differences between CPU utilization of any two systems is small.

$$\Sigma_{1 \le i < j \le N} \left| c_{i,T} - c_{j,T} \right| . \tag{4}$$

If the algorithms do not perform the load balancing process for a sufficient amount of time, Eq.(5) holds. With Eq.(5), the upper limits of the evaluation function Eq.(4) of the algorithms A_d and A_r are provided by Eq.(6) and Figure 6 for illustrative purposes.

$$\begin{aligned} \left| c_{i,T} - \left(\Sigma_{j=1}^{N} c_{j,T} \right) / N \right| &\le \Delta, && \text{for } A_d, \\ \max_i c_{i,T} - \min_i c_{i,T} &\le \Delta, && \text{for } A_r. \end{aligned} \tag{5}$$

Since equality of Eq.(6) holds for the algorithm A_r where half of the $c_{i,T} = \Delta$ and the rest of $c_{i,T} = 0$, and the upper limit of the algorithm A_r is tight, the algorithm A_d can perform load balancing more closely than the algorithm A_r.

The result shows that the mean deviation function is more suitable than the range function for the dynamic load balancing method striking a load balance across an entire managed network. Recall that the linear function defined in the existing method [5] is a variation of Eq.(2). This implies that even if the existing method [5] could be applied to an entire managed network, there would still be room for improvement, and it would be better to use the mean deviation function for closer load balancing.

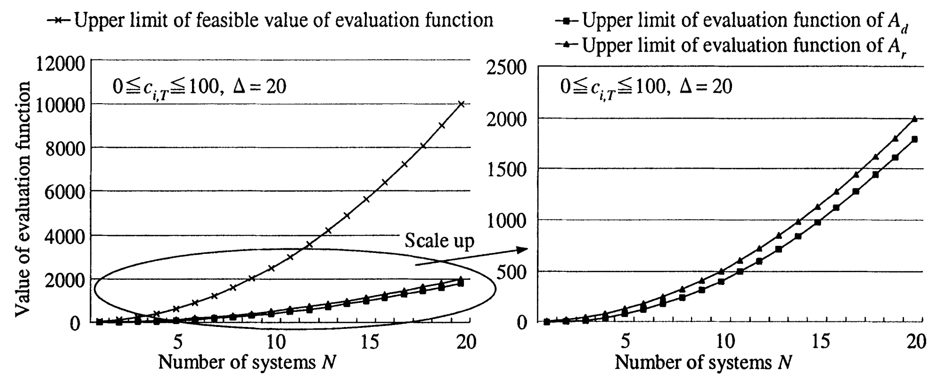


Figure 6. Capability of load balancing of algorithms

$$\Sigma_{1 \le i < j \le N} |c_{i,T} - c_{j,T}| \le \begin{cases} 1/4N(N-2)\Delta & \text{for } A_d, \\ 1/4N^2\Delta & \text{for } A_r. \end{cases} \tag{6}$$

6.1.3 Time Complexity of Criteria for Load Balancing.

Time Complexity of Mean Deviation Function. An evaluation of mean deviation function Eq.(1) requires $(2N + 1)$ additions and one division. The time complexity of the mean deviation function is $O(N \log |c_{i,T}| + (\log |c_{i,T}|)^2)$ as the time complexity of an addition and division are $O(\log |c_{i,T}|)$ and $O((\log |c_{i,T}|)^2)$, respectively.

Time Complexity of Range Function. An evaluation of mean deviation function Eq.(2) requires one max, one min and one additional operation. The time complexity of the range function is $O(N + \log |c_{i,T}|)$ as the time complexity of max and min operations over N elements is both $O(N)$.

The above results show that time complexity of the range function is better than that of the mean deviation function. However, since $|c_{i,T}|$ represents system i processing load and is limited by at most 100(%) in reality, the number of N systems dominates both complexities; thus, they can be considered the same as $O(N)$. Along with the result in 6.1.2, the algorithm A_d with the mean deviation function performs better than A_r at almost the same computational cost.

6.2 Empirical Evaluations

By applying the prototype system to an operational LAN, how well the proposed method can perform load balancing with a trivial overhead is evaluated. Based on the results in Section 6.1, the mean deviation function is used in this section. The system specification in the evaluation is shown in Table 2. The parameter values of management programs are set as in Table 3. All PCs are connected to the same LAN (10Mbps Ethernet) and the maximum tolerable bandwidth B is set to 100Kbps. The proposed method attempts to perform load balancing every 100 seconds.

6.2.1 Tolerable Deviation Δ in Operational LAN. First, the minimum value is derived of the tolerable deviation Δ satisfying Conditionl in Section 6.1.1. The processing load is 65.0% when management program A polling 20 managed ob-

Table 2. System specification[a] in evaluation.

Name	CPU	CPU frequency (MHz)	Memory (MB)	Description
Systeml	Intel Pentium	133	32	managed system
System2	Intel Pentium	150	96	managed system
System3	Intel Pentium II	266	64	managed system
System4	Intel Pentium II	450	64	managed system
System5	Intel Pentium II	450	128	management system

[a]OS is WindowsNT4.0 SP5.

Table 3. Parameter settings of management program A and CPU monitoring program.

Management program A	
Management information for polling	Object types of counter syntax in MIB-II ifEntry
Number of managed object instances for polling	1, 5, 10, 15 and 20
Polling interval[a]	5, 10, 15, 30, 60 and 180 seconds
Upper and lower thresholds	500 and 0
Priority	Common to all management programs
CPU monitoring program	
Polling interval	1 second
Sending interval	100 seconds

[a]Common polling interval when more than one object instance is specified.

ject instances for every 5 seconds and referred to as $A_{20,5}$ is executed on Systeml with the lowest processing capability of all systems. This implies that $c_{\max}$=65.0%. By Eq.(3), Δ>65.0×(1−1/5)=52.0(%). This value is obviously too greater. A smaller value can be obtained by more specific and finer-grained management programs. That is, 20 management programs each polling one managed object instance every 5 seconds and referred to as $A_{1,5}$, perform an almost equivalent management task together with that of an $A_{20,5}$. It brings a smaller value of Δ for closer load balancing. In this evaluation, there is hardly any difference betaween $A_{20,5}$ and 20 $A_{1,5}$'s processing loads for any other values for the number of managed object instances for the polling and polling interval. By executing the more specific and finer-grained management programs, i.e., 20 $A_{1,5}$'s on Systeml, $c_{\max}$ in turn becomes 4.2%, then the value of Δ is reduced to 4.2×(1−1/5)=3.36%.

Next, the minimum value of Δ satisfying Condition2 is derived. It takes 18.7, 21.5, 6.5 and 4.1 seconds on average to download the management program $A_{20,5}$ to Systeml, 2, 3 and 4. This time is almost the same for any other values of the number of managed object instances for the polling and polling interval. For every download, the CPU utilization marks 100% all the time and this implies that these download times correspond to the processing load per 100 seconds. The maximum processing load of all destination systems where a management program is downloaded is 21.5% and Δ > 21.5% when T is 100 seconds.

From the above discussion, the minimum value of Δ satisfying both conditions simultaneously is 21.5%.

6.2.2 Load Balancing by Proposed Method in Operational LAN. A heavy load is imposed on System2 by executing 20 management programs, each polling one managed object instance every 5 seconds, as well as 25% stationary processing load. Measurement is performed for the first time when there is no management program moved by the proposed method for the last 10 minutes, referred to as convergence time, and the maximum deviation of the average CPU utilization at convergence time, for some different values of Δ. The method attempts to move a management program every 100 seconds in sequence and all the 20 management pro-

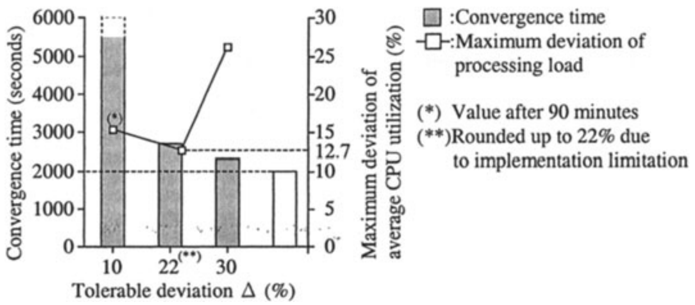


Figure 7. Convergence time and maximum deviation of average CPU utilization

grams being distributed to other systems is optimally desirable. Thus, it takes at least 2000 seconds (=20×100) for any value of Δ, shown as the white bar for comparison in Figure 7.

As shown in Figure 7, the convergence time shortens as the value of Δ is greater, whereas the maximum deviation of average CPU utilization increases from value 22% to 30%. The convergence time could not be measured within 90 minutes (= 5400 seconds) in the worst case when the value of Δ is 10%, smaller than the value of 22%. All of the above results show that setting the value of Δ = 22% can control the differences in average CPU utilization between systems within a small value of 12.7% as in Figure 7. Thus, the proposed method can strike a closer load balance for management tasks.

6.2.3 Bandwidth Consumption. The CPU monitoring program has small bandwidth consumption for sending CPU utilization messages. It is a one-way message at 60 bytes. It is smaller than the size of a minimum SNMP PDU of approximately 90 bytes, and becomes 180 bytes with a request and response pair. In this evaluation, since the sending interval is 100 seconds, the bandwidth consumption is 4.8bps. When the number of managed systems is 1000, then it becomes and is sufficiently small enough if compared with the tolerable 4.8Kbps bandwidth B (= 100Kbps) in Section 6.1.1.

6.2.4 Processing Load of Management and Managed Systems. The load balancing overhead of the proposed method on the management system is up to 16.2% and the process is completed within 1.4 seconds. This is approximately equivalent to that of a single polling for 12 managed object instances from the management system in the evaluation. Since the management system performs polling in the tens of managed object instances in general, the proposed method does not adversely affect the advantage of the distributed management model.

The CPU monitoring program processing load is 2.3%, 2.1%, 0.9% and 0.5%, respectively, when executed on Systeml, 2, 3 and 4. Each of them is less than or equal to the processing load of the management program A, polling a managed object instance every 5 seconds when executed on the corresponding systems. This brings hardly any impacts on the managed system in practical.

7. Conclusions

This paper proposed a new dynamic load balancing method for distributed network management and evaluated it both analytically and empirically. The proposed method strikes the load balance of management tasks across an entire managed network on the basis of 1) the processing load of management and managed systems, and 2) the bandwidth required for executing all management programs through the coexistence of centralized and distributed management models.

Since the criteria for the load balancing of an existing method is not at all associated with the processing load of management and managed systems, it cannot perform dynamic load balancing in considering network resource utilization. Although there is another existing method whose criteria are associated with the processing load, the method can perform only within a localized subnetwork and cannot strike a load balance across an entire managed network. Moreover, the existing method is less capable, as some load balancing functions are too simple to perform closer load balancing. The upper limit of absolute differences between CPU utilization of any two systems of the existing method is at most $N^2/4$, where N denotes the number of systems across an entire managed network. By introducing the mean deviation function as a new load balancing criteria, the proposed method can improve the capability and result in absolute differences between CPU utilization of any two systems at up to $N(N-2)/4$, with the same calculational cost $O(N)$.

A prototype system was also implemented based on the proposed method and evaluated through application in an operational LAN. The results showed that the proposed method could perform well in practice with only a slight overhead. An extension to wide area network and more evaluations in heterogeneous networks are left for further studies.

Acknowledgments

We would like to thank Mr. Tohru Asami, President, CEO of KDDI R&D Laboratories Inc., Dr. Yuichi Mastushima, Executive Vice President and Dr. Takeshi Mizuike Executive Director for their continuous encouragement with respect to this research.

References

[1] P. Bellavista, A. Corradi, and C. Stefanelli. An Open Secure Mobile Agent Framework for Systems Management. *Network System Management*, Vol.7, No.3, pp.323–339.

[2] A. Bieszczad, B. Pagurek, and T. White. Mobile Agents for Network Management. *IEEE Comm. Surveys*, Vol.1, No.1, 1998.

[3] M. Garey and D. Johnson. *Computers and Intractability, A guide to the Theory of NP-completeness.* W.H. Freeman and Co., 1979.

[4] D. Gavalas, D. Greenwood, M. Ghanbari, and M. O'Mahony. An Infrastructure for Distributed and Dynamic Network Management based on Mobile Agent Technology. In *Proc. of IEEE ICC* '99, pp.1362–1366, 1999.

[5] D. Gavalas, D. Greenwood, M. Ghanbari, and M. O'Mahony. Hierarchical Network Management: A Scalable and Dynamic Mobile Agent-based Approach. *Computer Networks*, Vol.38, No.6, pp.693–711, 2002.

[6] R. Giladi and M. Gat. Meta-management of Dynamic Distributed Network Managers. In *Proc. of IFIP/IEEE DSOM* 2000, pp.119–131, 2000.

[7] G. Goldszmidt and Y. Yemini. Evaluating Management Decisions via Delegation. In *Proc. of IFIP ISINM* '93, pp.247–257, 1993.

[8] G. Goldszmidt and Y. Yemini. Delegated Agents for Network Management. *IEEE Comm. Mag.*, Vol.36, No.3, pp.66–70, 1998.

[9] J. Grégoire. Models and Support Mechanisms for Distributed Management. In *Proc. of IFIP ISINM* '95, pp.17–28, 1995.

[10] IETF RFC 1157. *A Simple Network Management Protocol (SNMP)*, May 1990.

[11] A. Liotta, G. Knight, and G. Pavlou. Modelling Network and System Monitoring over The Internet with Mobile Agents. In *Proc. of IFIP/IEEE NOMS* '98, pp.303–312, 1998.

[12] T. Magedanz and T. Eckardt. Mobile Software Agents: A New Paradigm for Telecommunication Management. In *Proc. of IFIP/IEEE NOMS* '96, pp.360–369, 1996.

[13] Y. Yemini, G. Goldszmidt, and S. Yemini. Network Management by Delegation. In *Proc. of IFIP ISINM* '91, pp.95–107, 1991.

[14] M. Zapf, K. Herrmann, and K. Geihs. Decentralized SNMP Management with Mobile Agents. In *Proc. of IFIP/IEEE IM* '99, pp.623–635, 1999.

SESSION 6

Configuration Management

Chair: Danny Raz
Technion, Israel

SCALABILITY OF PEER CONFIGURATION MANAGEMENT IN PARTIALLY RELIABLE AND AD HOC NETWORKS

Mark Burgess
Faculty of Engineering, Oslo University College, Norway
Mark.Burgess@iu.hio.no

Geoffrey Canright
Telenor Research, Fornebu, Oslo, Norway
Geoffrey.Canright@telenor.com

Abstract: Current interest in *ad hoc* and *peer-to-peer* networking technologies prompts a re-examination of models for configuration management, within these frameworks. In the future, network management methods may have to scale to millions of nodes within a single organization, with complex social constraints. In this paper, we discuss whether it is possible to manage the configuration of large numbers of network devices using well-known and no-so-well-known configuration models, and we discuss how the special characteristics of ad hoc and peer-to-peer networks are reflected in this problem.

Keywords: Configuration management, ad hoc networks, peer to peer.

1. Introduction

Configuration management is about ensuring that the operational state of a device or host conforms to specifications lain down by a *site policy*. The configuration of a host ensures its efficiency, correctness and security in performing its function. System configuration is usually a specification of file or database contents, attributes, and process or service characteristics, including access rights, software customization and so on. A number of approaches has been devised for configuration management. For instance, the IETF model of configuration management revolves traditionally around the Simple Network Management Protocol (SNMP)[6]. This is read/write state based protocol for altering values in a management information database (MIB), and is used by a number of commercial software products. The 'Telecommunications Management Network' or TMN[12] is an alternative scheme designed for telecommunications networks and has a strong relationship with the OSI management model. These systems use an abstraction based on the concept of 'managed objects'. An different approach is used by systems like cfengine[2] and PIKT[13], which use descriptive languages to describe the attributes of many objects at the same time, and agents to enforce the rules.

The ability to send or receive messages is crucial to configuration management of network devices and hosts. Indeed, maintaining the configuration of hosts over time has many features in common with the problem of information transmission over a noisy channel[5]. Today, distributed systems sport a global geography, and are linked, both conceptually and physically, by a network infra-structure. Passing messages from one part of a system to another is subject to a plethora of uncertainties. For example, SNMP uses an unreliable transport protocol UDP for communication; any configuration scheme that relies on the availability of a resource or component at a specific moment has only a limited chance of being carried out. Systems can be unavailable due to power failures, physical breakages, absence of dependencies and so on. There is thus an ad hoc element to network connectivity even in an ostensibly permanent infra-structure. The additional complication of mobile services, with partial or intermittent connectivity adds to this problem.

An 'ad hoc' network (AHN) is defined to be a networked collection of mobile hosts, each of which has the possibility to route information. The union of those hosts forms an arbitrary graph that changes with time. The nodes are free to move randomly; thus the network topology may change rapidly and unpredictably. Clearly ad hoc networks are important in a mobile computing environment, where hosts are partially or intermittently connected to other hosts. While there has been some discussion of de-centralized network management using mobile agents[15], the problem of mobile nodes (and so strongly time-varying topology) has received little attention. However, we will argue below that ad-hoc networks provide a useful framework for discussing the problems surrounding configuration management in all network types, both fixed and mobile. This should not be confused with the notion of 'ad hoc management'[11], which concerns randomly motivated and scheduled checks of the hosts.

The plan for our paper is as follows. We begin by outlining how reliability can be discussed in terms of ad hoc connectivity in order to take advantage of its known scaling properties. Then noting how peer to peer communication implies decentralized policy, we estimate the required flow of configuration information as a function of the number of hosts, for a number of management models, in order to determine their scalability.

2. Availability of peers in a network

The probability that a host will be correctly configured is related to reliability of its communication with a policy source. As a simplest case, we assume that the reliability of each node and each link is independent of all others, so that the probabilities of availability are all independent random variables. In general this is not true, since some hosts/nodes depend on others for crucial services (e.g. the domain name service (DNS)), but this should suffice to gauge orders of magnitude.

DEFINITION 1 *A set of nodes or hosts is defined by a vector of probabilities* $\vec{h}^T = (p_1, p_2, \ldots, p_N)$, *where* $p_i (i = 1 \ldots N)$ *is the probability that node* i *is available. If the probabilities are 1, the hosts are said to be reliable, otherwise they are partially reliable.*

The nodes themselves may have any geographical location, and may be connected by any means. The connectivity between the nodes is represented by a matrix.

DEFINITION 2 *A network is defined by its adjacency matrix. By convention, the adjacency matrix of a network or graph is a symmetric matrix with zero leading diagonal. Zeroes denote no connectivity, while a 1 means a connection. The notation* $A(1)$ *distinguishes this (instantaneous) matrix, whose entries are binary-valued, from the time-averaged matrix discussed below. Owing to access and routing controls, this matrix need not be symmetrical in practice, but we shall not address that issue here.*

The properties of networks can be discussed in detail, using the adjacency matrix representation (see for instance, ref. [14]). It is not our intention to go into excessive detail here, but rather to distill a way of estimating the properties of networks. For this, we choose to look at the average properties of the networks.

We define a simple measure of the availability of a service, transmitted within a closed network, by an invariant scalar value χ:

DEFINITION 3 *The* connectivity, χ, *of a network $\mathcal{N}$, is the probability (averaged over all pairs of nodes) that a message can be passed directly between any two nodes. χ may be written as*

$$\chi = \frac{1}{N(N-1)} \vec{h}^{\mathrm{T}} A \vec{h}. \tag{1}$$

χ has a maximum value of 1, when every node is connected to every other, and a minimum value of zero when all nodes are disconnected.

For a fixed topology and time-independent node availabilities, χ is a constant characterizing the network. In general χ is time-dependent; one then obtains a static figure for the network by taking the long-time average.

$$\langle\chi\rangle = \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} \chi(t_i). \tag{2}$$

The utility of this measure is that it enables us to gauge and compare different network configurations on equal terms. It is also the vehicle by which we can map the problem of unreliable hosts in a fixed network onto a corresponding problem of reliable hosts in an ad hoc network.

3. Ad hoc networks

Ad hoc networks are networks whose adjacency matrices are subject to a strong, apparently random time variation. If we look at the average adjacency matrices, over time, then we can represent the probability of connectivity in the network as an adjacency matrix of probabilities.

DEFINITION 4 *An ad hoc network is represented by a symmetric matrix of probabilities for adjacency. Thus the time average of the adjacency matrix (for, e.g., four nodes) may be written as*

$$\langle A\rangle = \begin{pmatrix} 0 & p_{12} & p_{13} & p_{14} \\ p_{21} & 0 & p_{23} & p_{24} \\ p_{31} & p_{32} & 0 & p_{34} \\ p_{41} & p_{42} & p_{43} & 0 \end{pmatrix} \tag{3}$$

An ad hoc network is therefore a partially reliable network.

To motivate our discussion further, we note that:

THEOREM 1 *A fixed network of partially-reliable nodes, h_i, is equivalent to an ad hoc network of reliable nodes, on average.*

PROOF 1 *This is easily seen from the definition of the connectivity, using a matrix component form:*

$$\begin{aligned} N(N-1)\langle\chi\rangle &= \sum_{ij} h_i(p_i)\,\langle A_{ij}(1)\rangle\, h_j(p_j) \\ &= \sum_{ij} h_i(1)\,\langle A_{ij}(p_i p_j)\rangle\, h_j(1). \end{aligned} \tag{4}$$

This concludes the proof.

The proof demonstrates the fact that one can move the probabilities (uncertainties) for availability from the host vectors to the connectivity matrix and vice versa; for example

$$\begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} 0 & p_1p_2 & p_1p_3 \\ p_2p_1 & 0 & p_2p_3 \\ p_3p_1 & p_3p_2 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

Thus an array of hosts with reliability probabilities p_i, is equivalent to an array of reliable hosts in an unreliable network, where the probability of communication between them is the product of probabilities (assumed independent) from the reliability vector.

Superposed onto the routing problem is another problem of conceptual dependence. One is not merely dependent on connectivity to provide a route for messages, but one depends on trusted sources of information. Thus the arrows from source to receiver are not merely bytes exchanged but authorized policy instructions. We shall consider this issue below.

4. Peer to peer

The emergence of network file sharing applications such as Napster and Gnutella has focused attention on an architecture known as peer-to-peer, whose aim is to provide worldwide access to information via a highly de-centralized network of 'peers'.

DEFINITION 5 *A peer to peer network service is one in which each node, at its own option, participates in or abstains from exchanging data with other nodes, over a communications channel.*

Peer to peer has a deeper significance than ad hoc file sharing. It is about the demotion of a central authority, in response to the political wishes of those participating in the network. This is an issue directly analogous to the policies used for configuration management. In large organizations, i.e. large networks, we see a frequent dichotomy of interest:

- At the high level, one has specialized individuals who can paint policy in broad strokes, dealing with global issues such as software versions, common security issues, organizational resource management, and so on. Such issues can be made by software producers, system managers and network managers.
- At the local level, users are more specialized and have particular needs, which large scale managers cannot address. Centralized control is therefore only a

partial strategy for success. It must be supplemented by local know-how, in response to local environmental issues. Managers at the level of centralized control have no knowledge of the needs of specialized groups, such as the physics department of a university, or the research department of a company. In terms of configuration policy, what is needed is the ability to accept the advice of higher authorities, but to disregard it where it fails to meet the needs of the local environment. This kind of authority delegation is not catered for by SNMP-like models. Policy based management attempts to rectify some of these issues[8].

What we find then is that there is another kind of networking going on: a social network, superimposed onto the technological one. The needs of small clusters of users override the broader strokes painted by wide area management. This is the need for a scaled approach to system management[3].

5. Configuration management in ad hoc networks

Configuration management deals with the problem of establishing and maintaining a policy conformant configuration on workstations and other hosts distributed around a network. Policy is usually a set of rules and specifications about the software and resources of each host, defined by a central authority and disseminated to the individual hosts either on demand, or by common update.

Configuration management relies on two main things: i) the availability of trusted resources to each networked host, including a policy P, and ii) the consistency of the configuration specified by that policy. In an unpredictable environment one has potentially several problems: Critical dependencies, including the policy itself, can become unavailable or out of date; trust relationships are less certain if hosts cannot verify one another's' identity, location or integrity. Thus security and verifiable control, within specified time limits, are at stake.

Even in a fixed infrastructure network, with only partial connectivity, the availability of the resources is open to uncertainty. This means that the ability to correctly disseminate policy configuration is open to uncertainty. The framework of ad hoc networks thus encompasses a number of issues and offers a framework for discussing configuration strategies in general. In recent times, there has been a move towards self-configuring networks. Discovery protocols like JINI have to deal with the ad hoc nature of networks, and the protocols themselves will need to take the uncertainties in topology into account. Today, most protocols assume a fixed infra-structure.

One question that has been posed in this connection is whether a peer to peer strategy, for disseminating configuration policy, could provide a way of spreading information quickly about the network. If that were the case, then the temporary unavailability of a node to a central resource would not necessarily imply its isolation from fresh, critical data. This kind of data distribution has been discussed before[7] in connection with the scalability of software distribution. On the down side, peer to peer reliance is clearly an open invitation to engage in malicious activity.

6. Predictability and scaling

As networks grow, some configuration strategies do not scale well. They continue to be used, however, by force of habit. We are interested in examining the scaling properties of different configuration management schemes, especially in the context of network models that look to the future of configuration management.

We consider a number of cases, in order of decreasing centralization, or increasing delegation. Our basic 'constitutive' assumption is that there is a simple linear relationship between the probability of successful configuration and the rate(s) of communication with the policy- and enforcement-source(s). We look only at the coarsest averages over time, in order to determine the long-term behaviours of the models. We consider a change of configuration ("charge") ΔQ to be proportional to an average rate of information flow (current) I, over a time Δt; that is $\Delta Q = I\Delta t$. This equation is valid when I represents the time-averaged flow over the interval. Since we are interested in the limiting behaviour for long times, this is sufficient for our needs.

Now we apply this simple picture to configuration management for dynamic networks. We take the point of view of a 'typical' or 'average' host. It generates error in its configuration at the (average) rate I_{err}, and receives corrections at the rate I_{repair}. Hence the rate of increase of error for the average node is:

$$I_{\text{fail}} = (I_{\text{err}} - I_{\text{repair}})\,\theta(I_{\text{err}} - I_{\text{repair}}). \tag{5}$$

The Heaviside step-function is defined by $\theta(x) = 1$ if $x > 0$ and $\theta(x) = 0$ if $x <= 0$, and signifies the fact that, if the repair rate exceeds the error rate, then (on average, over long times) nothing remains outstanding and there is no net rise in configuration error. Thus this averaged quantity is never negative.

If random errors and changes to configuration occur at a rate I_{err} and the configuration agent is unavailable to correct them, then $I_{\text{fail}} = I_{\text{err}}$. If this holds during a time Δt, the configuration falls behind by an amount:

$$\begin{array}{c}\text{Bytes}\\ \text{missing}\\ (\Delta Q)\end{array} = \begin{array}{c}\text{bytes/sec}\\ (I_{\text{err}})\end{array} \times \begin{array}{c}\text{seconds}\\ \text{unavailable}\\ (\Delta t)\end{array}.$$

In the following we will use p to denote the average (over time, and over all nodes) probability that configuration management information flow (repair current) is not available to a node. This unavailability may come from either link or node unreliability. We can lump all the unreliability into the links (see above) and so write $p = (1 - \langle A_{ij}\rangle)$, where $\langle A_{ij}\rangle$ denotes both time and node-pair average. Each node then can only receive repair current during the fraction $(1 - p)$ of the total elapsed time.

The repair current is generated by two possible sources in our models: i) a remote source, and ii) a local source. In each case, the policy can be transmitted and/or enforced at a maximum rate given by the channel capacity of the source. We shall denote the channel capacities by C_R and C_L for remote and local sources for clarity, but we assume that $C_R \sim C_L$, since source and target machines are often comparable, if not identical. If the communication by network acts as a throttle on these rates, then one can further assume that $C_R < C_L$. In any case, the weakest link determines the effective channel capacity. Note that in the case of a confluence of traffic, as in the star models below, the channel capacity will have to be shared by the incoming branches. We now have a criterion for eventual failure of a configuration strategy. If $I_{\text{fail}} = \frac{\Delta Q}{\Delta t} > 0$, the average configuration error will grow monotonically for all time, and the system will eventually fail in continuous operation. Our strategy is then to look at the scaling behaviour of I_{fail} as the number of nodes N grows large.

Table 1. Comparison of models from the viewpoint of the different dimensions: policy dissemination, enforcement, freedom of choice, whether hosts can exchange chosen policy ideas with peers and how political control flows. A 'push' model implies a forcible control policy, whereas 'pull' signifies the possibility to choose. Model 3 lies between these two, in having the possibility but not the inclination to choose.

Model	Application Topology	Enforcement	Policy Freedom	Policy Exchange	Control Structure
1	Star	Transmitted	No	No	Radial push
2	Star	Transmitted	No	No	Radial push
3	Mesh	Local	No	No	Radial pull
4	Mesh	Local	Yes	No	Radial pull
5	Mesh	Local	Yes	Yes	Hierarchical pull
6	Mesh	Local	Yes	Yes	P2P pull

Star model

The traditional (idealized) model of host configuration is based on the idea of remote management (e.g. using SNMP). Here one has a central manager who decides and implements policy from a single location, and all networks and hosts are considered to be completely reliable. The manager must monitor the whole network, using bi-directional communication. This leads to an N : 1 ratio of clients to manager (see fig 1). This first model is an idealized case in which there is no unreliability in any

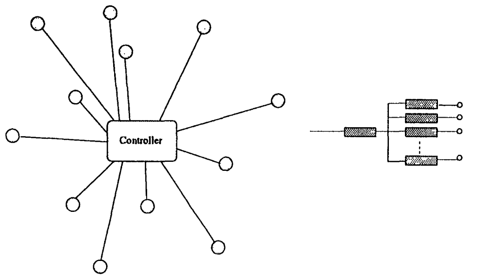


Figure 1 Model 1: the star network. A central manager maintains bi-directional communication with all clients. The links are perfectly reliable, and all enforcement responsibility lies with the central controller.

component of the system. It serves as a point of reference.

The topology on the left hand side of fig 1 is equivalent to that on the right hand side. We can assume a flow conservation of messages on average, since any dropped packets can be absorbed into the probabilities for success that we attribute to the adjacency matrix. Thus the currents must obey Kirchoff's law:

$$I_{\text{controller}} = I_1 + I_2 + \dots I_N. \tag{6}$$

The controller current cannot exceed its capacity, which we denote by C_S. We assume that the controller puts out repair current at its full capacity (since the Heaviside function corrects for lower demand), and that all nodes are average nodes. This gives that $I_{\text{repair}} = \frac{C_S}{N}$. The total current is limited only by the bottleneck of queued messages at the controller, thus the throughput per node is only $1/N$ of the total capacity. We can now write down the failure rate in a straightforward manner:

$$I_{\text{fail}} = \left(I_{\text{err}} - \frac{C_S}{N}\right) \theta \left(I_{\text{err}} - \frac{C_S}{N}\right). \tag{7}$$

As $N \to \infty$, $I_{\text{fail}} \to I_{\text{err}}$—that is, the controller contributes a vanishing repair current per node. The system fails however at a finite $N = N_{\text{thresh}} = C_S/I_{\text{err}}$. This highlights the clear disadvantage of centralized control, namely the bottleneck in communication with the controller.

Star model in intermittently connected environment

The previous model was an idealization, and was mainly of interest for its simplicity. Realistic centralized management must take into account the unreliability of the environment.

In an environment with partially reliable links, a remote communication model bears the risk of not reaching every host. If hosts hear policy, they must accept and comply, if not, they fall behind in the schedule of configuration. Monitoring in distributed systems has been discussed in ref. [1].

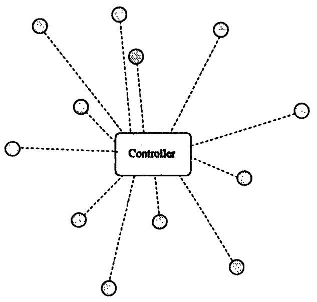


Figure 2 Model 2: a star model, with built-in unreliability. Enforcement is central as in Model 1.

The capacity of the central manager C_S is now shared between the average number of hosts $\langle N \rangle$ that is available, thus

$$I_{\text{repair}} = \frac{C_S}{N\langle A_{ij}\rangle} \equiv \frac{C}{\langle N \rangle} \,. \tag{8}$$

This repair current can reach the host, and serve to decrease its policy error ΔQ, during the fraction of time $(1-p)$ that the typical host is reachable. Hence we look at the net deficit ΔQ accrued over one "cycle" of time Δt, with no repair current for $p\Delta t$, and a maximal current $C_S/\langle N \rangle$ for a time $(1-p)\Delta t$. This deficit is then

$$\Delta Q(\Delta t) \;=\; I_{\text{err}} p \Delta t + \left(I_{\text{err}} - \frac{C_S}{\langle N \rangle}\right)(1-p)\Delta t \tag{9}$$

(here it is implicit that a negative ΔQ will be set to zero). Thus, the average failure rate is

$$I_{\text{fail}} \;=\; I_{\text{err}} p + \left(I_{\text{err}} - \frac{C_S}{\langle N \rangle}\right)(1-p) = I_{\text{err}} - \frac{C_S}{N} \,. \tag{10}$$

(Again there is an implicit θ function to keep the long-time average failure current positive.) This result is the same as for Model 1, the completely reliable star. This is because we assumed the controller was clever enough to find (with negligible overhead) those hosts that are available at any given time, and so to only attempt to communicate with them.

This model then fails (perhaps surprisingly), on average, at the same threshold value for N as does Model 1. If the hunt for available nodes places a non-negligible burden on the controller capacity, then it fails at a lower threshold.

Mesh topology with centralized policy and local enforcement

The serialization of tasks in the previous models forces configuration 'requests' to queue up on the central controller. Rather than enforcing policy by issuing every instruction from the central source, it makes sense to download a summary of the policy to each host and empower the host itself to enforce it.

There is still a centrally determined policy for every host, but now each host carries the responsibility of configuring itself. There are thus two issues: i) the update of the policy and ii) the enforcement of the policy. A pull model for updating policy is advantageous here, because every host then has the option to obtain updates at a time convenient to itself, avoiding confluence contentions; moreover, if it fails to obtain the update, it can retry until it succeeds. We ask policy to contain a self-referential rule for updating itself.

The distinction made here between communication and enforcement is important, because it implies distinct types of failure, and two distinct failure metrics: i) distance of the locally understood policy from the latest version, and ii) distance of host configuration from the ideal policy configuration. In other words: i) communication failure, and ii) enforcement failure.

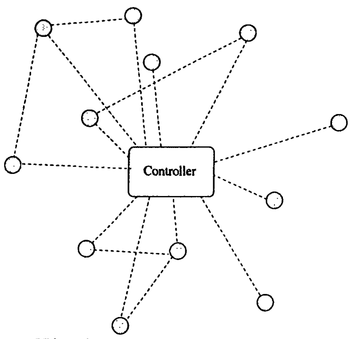


Figure 3 Model 3. Mesh topology. Nodes can learn the centrally-mandated policy from other nodes as well as from the controller. Since the mesh topology does not assure direct connection to the controller, each node is responsible for its own policy enforcement.

The host no longer has to share any bandwidth with its peers, unless it is updating its copy of the policy, and perhaps not even then, since policy is enforced locally and updates can be scheduled to avoid contention.

Let $I_{\rm update}$ be the rate at which policy must be updated. This current is usually quite small compared to $I_{\rm err}$, and was neglected in the previous models. Based on the two failure mechanisms present here, we break up the failure current into two pieces: $I_{\rm fail} = I_{\rm fail}(i) + I_{\rm fail}(ii)$. The former term is

$$I_{\rm fail}(i) \quad = \quad (I_{\rm err} - C_L)\theta(I_{\rm err} - C_L) \, ; \qquad (11)$$

this term is independent of N and may be made zero by design. $I_{\rm fail}(ii)$ is still determined by the ability of the controller to convey policy information to the hosts. However, the load on the controller is much smaller since $I_{\rm update} \ll I_{\rm err}$. Also, the topology is a mesh topology. In this case the nodes can cooperate in diffusing policy updates, via flooding (Note, flooding in the low-level sense of a datagram multicast is not necessarily required, but the effective dissemination of the policy around the network is an application layer flood.) .

The worst case—in which the hosts compete for bandwidth, and do not use flooding over the mesh—is that, for large N, $I_{\text{fail}} \to I_{\text{update}}$. This is a great improvement over the two previous models, since $I_{\text{update}} \ll I_{\text{err}}$. However note that this can be further improved upon by allowing flooding of updates: the authorized policy instruction can be available from any number of redundant sources, even though the copies originate from a central location. In this case, the model truly scales without limit, i.e. $I_{\text{fail}} = 0$.

There is one caveat to this encouraging result. If the (meshed) network of hosts is truly an ad-hoc network of mobile nodes, employing wireless links, then connections are not feasible beyond a given physical range r. In other words, there are no long-range links: no links whose range can grow with the size of the network. As a result of this, if the AHN grows large (at fixed node density), the path length (in hops) between any node and the controller scales as a constant times $\sqrt{N}$. This growth in path length limits the effective throughput capacity between node and controller, in a way analogous to the internode capacity. The latter scales as $1/\sqrt{N}$ [9, 10]. Hence, for sufficiently large N, the controller and AHN will fail collectively to convey updates to the net. This failure will occur at a threshold value defined by

$$I_{\text{fail}}(ii) = I_{\text{update}} - \frac{C_S}{c\sqrt{N_{\text{thresh}}}} = 0, \qquad (12)$$

where c is a constant. The maximal network size N_{thresh} is in this case proportional to $\left(\frac{C_S}{I_{\text{update}}}\right)^2$—still considerably larger than for Models 1 and 2.

Mesh topology with partial host autonomy and local enforcement

As a variation on the previous model, we can begin to take seriously the idea of distance from a political centre. In this model, hosts can choose not to receive policy from a central authority, if it conflicts with local interests. Communication thus takes the role of conveying 'suggestions' from the central authority, in the form of the latest version of the policy. For instance, the central authority might suggest a new version of widely-used software, but the the local authority might delay the upgrade due to compatibility problems with local hardware. Local enforcement is now employed by each node to hold to its chosen policy P_i. Thus communication and enforcement use distinct channels (as with Model 3); the difference is that each node has its own target policy P_i which it must enforce.

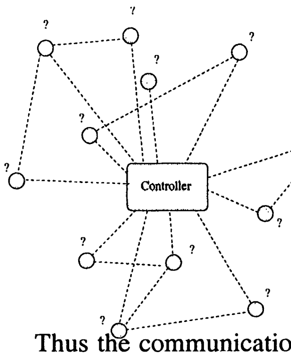


Figure 4 Model 4. As in Model 3, except the hosts can choose to disregard or replace aspects of policy at their option. Question marks indicate a freedom of hosts to choose.

Thus the communications and enforcement challenges faced by Model 4 are the same (in terms of scaling properties) as for Model 3: i.e. I_{fail} is the same as that in

Model 3. Hence this model can in principle work to arbitrarily large N. Model 4 is the model used by cfengine[2, 4]. The largest current clusters sharing a common policy are known to be of order 10^4 hosts, but this could soon be of order 10^6, with the proliferation of mobile and embedded devices.

Mesh, with partial autonomy and hierarchical coalition

An embellishment of Model 4 is to allow local groups of hosts to form policy coalitions, that serve to their advantage. Such groups of hosts might belong to one department of an organization, or to a project team, of even to a group of friends in a mobile network. Once groups form, it is natural to allow sub-groups and thence a generalized hierarchy of policy refinement through specialized social groups.

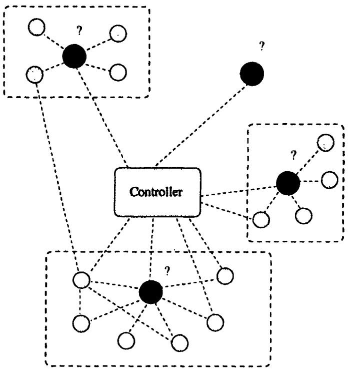


Figure 5 Model 5. Communication over a mesh topology, with policy choice made hierarchically. Sub-controllers (dark nodes) edit policy as received from the central controller, and pass the result to members of the local group (as indicated by dashed boxes). Question marks indicate the freedom of the controllers to edit policy from above.

If policies are public then the scaling argument of Model 3 still applies since any host could cache any policy; but now a complete policy must be assembled from several sources. Once can thus imagine using this model to distribute policy so as to avoid contention in bottlenecks, since load is automatically spread over multiple servers. In effect, by delegating local policy (and keeping a minimal central policy) the central source is protected from maximal loading. Specifically, if there are S sub-controllers (and a single-layer hierarchy), then the effective update capacity is multiplied by S. Hence the threshold N_{thresh} is multiplied (with respect to that for Model 3) by the same factor. This model could be implemented using cfengine, with some creative scripting.

Mesh, with partial autonomy and inter-peer policy exchange

The final step in increasing autonomy is the free exchange of information between arbitrary hosts. Hosts can now offer one another information, policy or source materials in accordance with an appropriate trust model. In doing so, impromptu coalitions and collaborations wax and wane, driven by both human interests and possibly machine learning. A peer-to-peer policy mechanism of this type invites trepidation amongst those versed in control mechanisms, but it is really no more than a distributed genetic algorithm. With appropriate constraints it could be made to lead to sensible convergent behaviour, or to catastrophically unstable behaviour.

One example of such a collaborative network that has led to positive results is the Open Source Community. The lesson of Open Source Software is that it leads to a rapid evolution. A similar rapid evolution of policy could also be the result from such

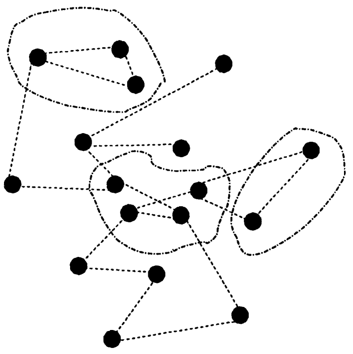


Figure 6 Model 6. Free exchange of policies in a peer-to-peer fashion; all nodes have choice (dark). Nodes can form spontaneous, transient coalitions, as indicated by the dashed cells. All nodes can choose; question marks are suppressed.

exchanges. Probably policies would need to be weighted according to an appropriate fitness landscape. They could include things like shared security fixes, best practices, code revisions, new software, and so on. Until this exchange nears a suitable stationary point, policy updates could be much more rapid than for the previous models. This could potentially dominate configuration management behaviour.

This model has no centre. Hence it is, by design, scale-free: all significant interactions are local. Therefore, in principle, if the model can be made to work at small system size, then it will also work at any larger size.

We note however that Model 6, of all the models presented here, has the greatest freedom to explore the space of possible policies. Hence an outstanding, and extremely nontrivial, question for this peer-to-peer model of configuration management is: can such a system find 'better' policies than centralized systems?

7. Summary and conclusion

We have presented several models for configuration management on networks. Our Models 3–6 depart from mainstream practice in various ways. The motivation for considering these models is the perception that highly centralized systems are not well adapted to networks that are too large, too heterogeneous, or too dynamic. Since current and future networks are taking on more and more of these three qualities, it is of interest to examine alternative models for configuration management.

We have held ourselves to a limited set of goals. The first of these is the definition of the models themselves. These models offer broad avenues for future research in configuration management; variants of one (or several) of them are likely to be important in future systems.

Our second goal has been to assess the scaling behaviour of these models with respect to two criteria: *communication* of the current policy to the hosts, and *enforcement* of the communicated policy. We have considered the various models' ability to meet these criteria, as the number of hosts N in the network grows large. We find, not surprisingly, that the highly centralized systems suffer from a communications bottleneck that limits the size at which they can function effectively. De-centralizing one or both of the two functions gives much better scaling behaviour—to the point that all of the Models 3–6 can, in principle (with some qualifications), implement policy communication and enforcement for very large systems.

Of course, de-centralization brings with it new problems, not addressed by the centralized system: problems of trust, of the quality of chosen policies, and of convergence to a stable regime. These new problems offer attractive issues for further research, due both to their intrinsic interest, and to their relevance to the future implementation of de-centralized network systems.

References

[1] H. Abdu, H. Lutfiya, and M. Bauer. A model for adaptive monitoring configurations. *Proceedings of the VI IFIP/IEEE IM conference on network management*, page 371, 1999.

[2] M. Burgess. A site configuration engine. *Computing systems (MIT Press: Cambridge MA)*, 8:309, 1995.

[3] M. Burgess. On the theory of system administration. *Submitted to J. ACM.*, 2000.

[4] M. Burgess. Cfengine's immunity model of evolving configuration management. *Submitted to IEEE Transactions on Software Engineering*, 2002.

[5] M. Burgess. System administration as communication over a noisy channel. *Proceedings of the 3nd international system administration and networking conference (SANE2002)*, 2002.

[6] J. Case, M. Fedor, M. Schoffstall, and J. Davin. The simple network management protocol. *RFC1155*, STD 16, 1990.

[7] A.L. Couch. Chaos out of order: a simple, scalable file distribution facility for intentionally heterogeneous networks. *Proceedings of the Eleventh Systems Administration Conference (LISA XI) (USENIX Association: Berkeley, CA)*, page 169, 1997.

[8] N. Damianou, N. Dulay, E.C. Lupu, and M. Sloman. Ponder: a language for specifying security and management policies for distributed systems. *Imperial College Research Report DoC 2000/1*, 2000.

[9] P. Gupta and P.R. Kumar. The capacity of wireless networks. *IEEE Trans. Info. Theory*, 46(2):388–404, 2000.

[10] J. Li, C. Blake, D.S.J. DeCouto, H.I. Lee, and R. Morris. Capacity of ad hoc wireless networks. *Proc. 7th ACM Intl. Conf. on Mobile Computing and Networking*, pages 61–69, 2001.

[11] J.P. Martin-Flatin. Push vs. pull in web-based network management. *Proceedings of the VI IFIP/IEEE IM conference on network management*, page 3, 1999.

[12] M. Matsushita. Telecommunication management network. *NTT Review*, 3:117–122, 1991.

[13] R. Osterlund. Pikt: Problem informant/killer tool. *Proceedings of the Fourteenth Systems Administration Conference (LISA XIV) (USENIX Association: Berkeley, CA)*, page 147, 2000.

[14] D.B: West. *Introduction to Graph Theory (2nd Edition).* (Prenctice Hall, Upper Saddle River), 2001.

[15] M. Zapf, K. Herrmann, K. Geihs, and J. Wolfang. Decentralized snmp management with mobile agents. *Proceedings of the VI IFIP/IEEE IM conference on network management*, page 623, 1999.

KHNUM - A SCALABLE RAPID APPLICATION DEPLOYMENT SYSTEM FOR DYNAMIC HOSTING INFRASTRUCTURES

Alain Azagury[1], German Goldszmidt[1], Yair Koren[2], Benny Rochwerger[1] and Arie Tal[3]
[1]*IBM Research*
AZAGURY@il.ibm.com, gsg@us.ibm.com, ROCHWER@il.ibm.com
[2]*Technion Israel Institute of Technology*
yair_k@cs.technion.ac.il
[3]*IBM Toronto Lab*
arietal@ca.ibm.com

Abstract: In a dynamically scalable hosting infrastructure for e-business computing, servers need to be quickly allocated in order to satisfy a sudden demand for increased computing power for a hosted site.

Khnum is the applications and data management component of Océano - a dynamically scalable hosting infrastructure for e-business computing utilities. It is responsible for server reconfiguration and for application deployment. Application deployment involves all services, configuration directives, executables and data of the application. A hosted site may include several applications.

Khnum enables Océano to rapidly deploy multiple applications to tens of servers simultaneously in just a few minutes. It uses AFS as the infrastructure for secure storage, automatically mapping files and directories onto the new servers' local filesystems and multicasting hot AFS cache content to the new servers. To avoid overloading the AFS servers during the deployment process, the hot cache content is multicasted to all the new servers, avoiding the boot storming (or "rushing") effect. This, in turn, improves the scalability of the deployment process; experimental results attest to Khnum's scalability in simultaneously deploying applications to tens of servers.

1. INTRODUCTION

The Océano project aims at providing a dynamically scalable hosting infrastructure for e-business computing utilities. Current co-location hosting environments use dedicated computing resources for every hosted site; these resources determine the capacity of the hosted site's applications.

In these environments, the process of adding new resources is complex, time-consuming and labor-intensive. This process normally involves the physical installation of the machines and supporting infrastructure (space allocation, racks or shelves, electricity, cooling, routers, etc.), and the deployment of the hosted site's applications, data and middleware. This process may also require some downtime while the site is being reconfigured for the added servers.

In addition, hosted sites increasingly require support for peak workloads that, in some cases, could be an order of magnitude larger than what they experience in their normal steady state. During promotions, a successful marketing campaign or a high seasonal demand, the number of accesses to an e-commerce site can increase significantly compared to its normal steady state, requiring much more resources for handling the larger workloads. In this model, enabling peak-load scale on demand would require large investments in standby, non-shared resources, which would be mostly under-utilized, occupy large amounts of physical space and require regular maintenance. Furthermore, a site may still go down if faced with a larger than expected workload and lacks proper access throttling. Clearly, such a model is not well suited to efficiently mitigate the differences between average and peak workloads. Thus a faster turnaround time in adjusting the resources (bandwidth, servers, and storage) assigned to each hosted site to the actual workload is needed.

Océano modifies the prevalent hosting model by increasing the sharing of resources, and by dynamically adjusting the amount allocated to each hosted customer according to the current observed demand. The Océano model assumes that all servers are clustered together within the same glass-house or campus, and the hosting environment is dynamically divided into secure, single hosted site domains. These domains are dynamic: the resources assigned to them may be augmented when workload increases and reduced when workload dips. This dynamic allocation of resources is controlled by flexible Infrastructure Service Level Agreement (ISLA) contracts with hosted customers. Océano administers the available resources, so that each hosted customer is provisioned as specified by its contract.

To do that, Océano maintains a pool of unassigned (free) servers that can be dynamically assigned to hosted sites. When a higher than normal workload is expected for a hosted site, a group of servers can be pre-allocated for the hosted site in advance. If however, during the live operation of the hosted site, Océano determines that the workload is going to increase beyond the hosted site's current capacity, more servers are then dynamically allocated for the hosted site (in accordance with the ISLA for the hosted site), while the site is handling the current workload. On the other hand, if Océano determines that the workload is going to decrease and that servers are going to be under-utilized, servers are de-allocated from the hosted site and returned to the pool of unallocated servers. Underlying subsystems provide the mechanisms for determining expected workloads, throttling access, managing resources and shifting them to and from a hosted site domain in pseudo real time (minutes) without compromising security requirements.

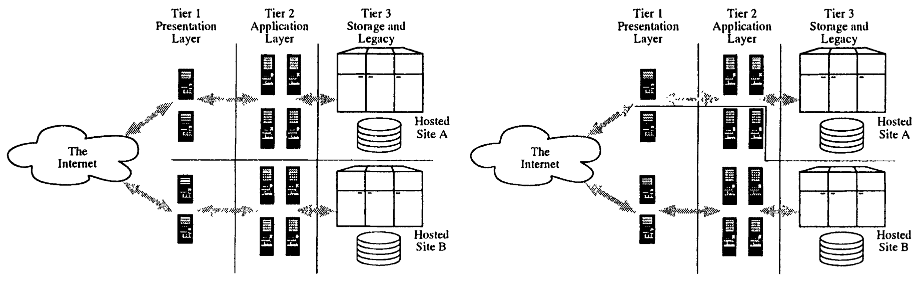
Figure1 illustrates this in a simple multi-tier scenario. Servers at Tier-1 and Tier-2 are dynamically reallocated as workload increases, while Tier-3 resources are assigned to hosted sites for very long periods of time. In Figure 1(a) 6 servers are allocated to hosted site A and 6 to hosted site B. Figure 1(b) illustrates the server allocation status following a detection of a significant increase in the workload of hosted site B and the reallocation of some of the machines. In the absence of "free" machines, to fulfill the needs of hosted site B, the Océano system took under-utlilized resources away from hosted site A[1].

Once a server has been allocated to a hosted site, it should be installed with the applications and data that will enable it to actively participate in the site's workload. In addition, the server's network configuration should be modified to identify it as belonging to the hosted site to which it was allocated. Setting up a new server involves time-consuming and labor-intensive operations such as application installation, configuration and tuning. Many techniques are available to ease these tasks (see Section 2).

In an Océano-hosted environment, the entire hosted site's data (including application binaries[2]) is kept in a shared file system; installation and configuration of applications is done off-line. *Khnum*[3], the applications and data management component of the Océano architecture, is responsible for the rapid deployment of applications (and data) on added servers. Application deployment, the process of setting up all the applications on a new server, is reduced to mapping a remote shared subtree (or several subtrees) to the local file system of the new server.

In the simplest case, the mapping involves only the creation of a few symbolic links, but it could also be relatively complex for applications that require system configuration changes where symbolic links are insufficient. In addition, Khnum is also responsible for configuring the machines to run the applications, and to pre-fetch application data and executables in order to bring them faster into the fully functional state.

Figure 1. Resources assigned are augmented when workload increases and reduced when workload dips



(a) Two hosted sites in the same Oceano hosted environment

(b) Due to sudden workload increase the servers are reallocated

In a multi-tier environment, any considerable change in the computing power (that is, number of servers) of any of the tiers can, if not managed properly, severely impact the adjacent tiers. For example, in an Océano-like hosting environment, a sudden increase in incoming requests may cause a quick buildup of computing power at Tier-1 and Tier-2; however, the number of Tier-3 servers stays the same, hence they may become a bottleneck when multiple new Tier-1 servers and Tier-2 servers all request access simultaneously to the same data from the Tier-3 servers. Over-provisioning Tier-3 for the maximum expected workload, as a way to avoid this problem, would be expensive and probably impractical. Khnum avoids choking the Tier-3 servers by pre-fetching "hot" cache data from a running Tier-1 server and a running Tier-2 server into newly allocated Tier-1 servers and Tier-2 servers, respectively. Furthermore, Khnum uses multicast to "push" this hot data to all new servers being allocated to a hosted site simultaneously. The Khnum model proved to be a very efficient mechanism for rapidly deploying applications on new servers. Using this model, we have successfully deployed multiple applications, including Apache, Jakarta-Tomcat, Real Media, and iPlanet.

The Khnum model requires AFS distributed filesystem support and a local filesystem supporting symbolic links. Therefore, although our implementation was based on RedHat Linux v6.2, other UNIX variants could be used for implementing this model.

The rest of this paper is organized as follows. Section 2 describes related work in application installation, distributed file systems, and cache pre-loading schemes. Section 3 presents Khnum's data sharing model, its components, its file system mapping method, and its cache pre-fetching. Section 4 describes experimental results, and conclusions and future directions are presented in Section 5.

2. RELATED WORK

2.1 Installing and Cloning

The RPM utility by Red Hat Linux [2] reduces the installation complexity by packing the applications with installation scripts and a list of dependencies. The **rpm** utility performs the dependencies check, unpacks applications and runs the installation scripts. Disk cloning products, such as *Symantec Ghost* [3], tackle the installation problem by copying entire disks from a pre-configured machine into one or more new machines. All these approaches assume that applications and data are installed on each machine as independent copies, which makes the task of "content management" hard since there is a need to maintain many synchronized copies of the same data. On the other hand, shared file systems have been used to store application binaries in a common place. However, architectures using this approach (for example, [4]) limit the use of shared data to a small set of predefined locations to reduce the complexity of managing mount points or symbolic links, or both.

2.2 Distributed File Systems for clusters

Khnum relies on the Andrew File System (AFS) [5], for sharing the hosted site's content between its multiple servers. Contrary to the weak caching semantics of the popular NFS [6], AFS provides robust caching mechanisms that allow access to large amounts of cached data with speeds comparable to those of local file system accesses. As extensions to AFS's efficient file sharing model, which significantly reduces the workload on the file servers, Coda [7] and Disconnected AFS [8] also allow access to the cached data even when the file server is inaccessible, while keeping their respective file system semantics. In the JetFile [9] multicast-based distributed file system, most operations, which are usually managed by the server, have been moved to the clients. JetFile also supports large caches (in the order of gigabytes), and uses dynamic replication as a means to localize traffic, contrary to AFS's static read-only replication.

2.3 Cache pre-loading schemes

The Khnum model is based on the assumption of a symmetric relationship between servers, that is, we assume that all the servers for a certain hosted site, serve more or less the same content[4], and hence should have very similar cache contents. Therefore, when adding new servers to a hosted site, the cache content of an active server is *multicasted* to the newly added servers, under the assumption that the new servers will be asked to serve roughly the same content. This model is different from SEER [10] and MFS [11], which propose sophisticated hoarding mechanisms for detecting which files should be stored in the cache, as a preparation for disconnected operation, under the assumption that different servers need different content.

Dedicated to caching Web content, LPC [12] employs multicasting for pushing cache content to cache replicas, and automatic tracking of popular Web pages for determining hot cache items. When popular Web pages are determined, their content is pushed to the Web servers, effectively minimizing the time it will take a server to serve that page for the first time. This is rather different from our approach, in which all servers, other than newly added ones, are quite independent of each other. However, in our model, we assume all servers are configured with a large enough AFS cache, so that all popular static Web pages should eventually be stored in the local cache of every server without imposing any additional complexity on the server [13]. On the other hand, if the Web pages are continuously generated every few minutes (for example weather reports, stock reports, sports results), LPC may be more scalable with large numbers of servers, significantly reducing the workload on the shared file system server. TriggerMonitor [14] uses a different, but related, approach by "sensing" changes in database entries, which are used to compose the dynamically generated Web pages. Generated pages are stored and used until a change is detected that requires a page regeneration. The newly generated pages can then be "pushed" to the participating Web servers that serve them as "static" pages.

3. THE DATA SHARING MODEL

To reduce the time and complexity of the *application deployment* process on new servers, all application data (applications executables, configuration files and data[5]) reside on a shared file system and the local disk is used only for temporary data (swapping, caching, and so on), machine specific configuration, and the basic operating system. Essentially, the servers on each cluster become *almost "data less" machines*. When a server either fails or is removed from a hosted site, all the data on its local disk gets wiped out (except for the basic operating system and machine specific configuration)[6].

Ideally, a single symbolic link into a subdirectory in the AFS tree would be sufficient to fully enable applications on the cluster nodes. However, this is doable only for a limited set of applications. In many cases applications require files in system directories such as /etc. Moreover, symbolic links to a shared directory should not be used for directories and files that are, by definition, local to a particular machine. For example, the installation of the Apache Web server creates the /var/log/http subdirectory to keep a local log of http activity. To overcome these problems and still keep the applications on the shared file system, the Khnum model suggests a three-phase process for deploying applications on an Océano-hosted environment:

1 Standard installation - The application is installed locally using the standard application procedure on an off-line machine designated as the *installation staging server*.

2 Analysis and relocation - Once an application has been installed, configured and tested, it is relocated to an area on the shared file system that mirrors the local disk of the installation staging server. Some applications can be directly installed on the shared file system, while others need manual classification of all its files according to their access:

Read-only files which can always be relocated to the shared file system as long as the actual path to them is kept (through symbolic links). In most cases this includes configuration files since these are typically modified once (or sporadically).

Instance read/write files contain data relevant to a particular instance of the application, and hence cannot be shared. Log files are a good example of this type of files. When the application is relocated, these files are separated from the application subtree into a local subtree (by modifying the application configuration files accordingly).

Application-wise read/write files contain information relevant to all instances of an application. To avoid inconsistencies, applications may choose to lock entire files or only portions of them, and to lock only during the write operations or during the entire "life" of the application (in which case the application becomes non-shareable).

Some applications may require an additional effort of creating customized configuration scripts. The purpose of these configuration scripts is to modify configuration files and perform additional tasks that cannot be achieved by simply replacing the existing files with application-specific ones (for example files containing server-specific configuration information). These configuration scripts will be invoked at a post-mapping phase (see below).

3 Mapping - The final step of bringing an application up includes: (a) stopping the processes running on the server undergoing application deployment; (b) creating the necessary links and directories on the server's file system; (c) running configuration scripts (when needed); and (d) restarting the system and application processes.

The application analysis process may be difficult at first, but the knowledge acquired on each application can be re-applied. Although the file hierarchy structure varies from application to application, the ongoing efforts to standardize the file system structure [15] will eventually simplify the process. Note that the time-consuming parts of this process (installation and analysis) are done off-line, and hence they are not part of the rapid deployment of applications on new servers. In addition, when a new server is allocated to a hosted site, previously allocated servers continue to work normally without interruption.

3.1 System Overview

As with most Océano components, Khnum's functionality is divided between a management sub-component - the *Khnum Manager* (**KhnumM**), which oversees and coordinates the application deployment process; and an execution sub-component - the *Khnum Daemon* (**KhnumD**), which is present on all servers and is responsible for file system mapping, cache pre-fetching and configuration. **KhnumM** waits for "application deployment instructions" from either *eClams*, the Océano component responsible for the management of servers [1], or directly from a user when working in standalone mode. These application deployment instructions consist of a hosted site identifier, a list of servers and a "command": either add the servers to the hosted site, or remove the servers from the hosted site, that is, restore the servers to the unallocated state. As a response to these instructions, **KhnumM** initializes the application deployment process on the new servers by selecting from the servers already allocated to the desired hosted site (the "old servers") a *cache pre-fetch source*[7] and then sending a START-DEPLOYMENT message to all new servers. We currently pick an arbitrary old server as the cache pre-fetch source. However, it is not clear whether we should pick older servers whose cache might have reached a level of stability, or newer servers that may still hold the files required for initialization.

Figure 2. The life cycle of a server

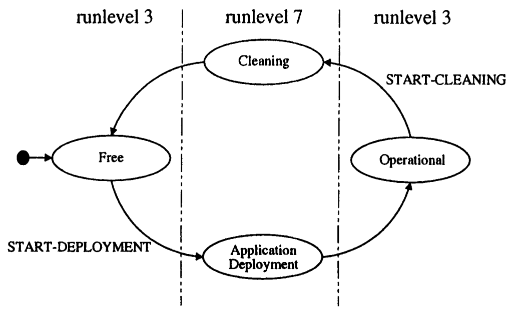


Figure 2 describes the life cycle of a server from Khnum's perspective. **KhnumD** on new servers is in the *Free* state where it waits for the START-DEPLOYMENT message from **KhnumM.** When **KhnumD** receives a request from **KhnumM** it responds by changing the system's *runlevel*[8] to a specially tailored runlevel 7[9]. This transition will cause all the services on the server undergoing applicationed deployment to be stopped and the initialization program *KhnumSetup* to run. *KhnumSetup* will create symbolic links to the shared file system, initialize the cache, optionally run configuration scripts[10], and initiate another runlevel transition, which will cause the new services to be started. A START-DEPLOYMENT message includes information on new AFS cell configuration files, a designated cache pre-fetch server identifier and a setup file (Khnum.setup) containing information on how to setup the required applications.

Upon receiving a START-DEPLOYMENT message, **KhnumD** enters the *Application Deployment* state, and performs the following procedure:

1 Stop all services on the server undergoing deployment (by switching to runlevel 7).
2 Stop AFS daemon[11] services.
3 Copy new AFS configuration files to a standard location.
4 Restart AFS daemon with the new configuration (to gain access to the new hosted site).
5 Find and set up all the symbolic links and directories on the hosted site's AFS cell required by the server (by running KhnumSetup) and run configuration scripts according to the definitions in Khnum.setup.
6 Stop AFS daemon (to prepare for cache pre-fetching).
7 If there are other servers undergoing application deployment for this hosted site, pre-fetch the AFS cache content (from the server designated by **KhnumM** as part of the START-DEPLOYMENT message).
8 Switch back to runlevel 3, which also restarts the AFS daemon and (system and application) services (new services are now started because in step 5 symbolic links were created in /etc/rc.d/init.d and /etc/rc.d/rc3.d).
9 Send a DEPLOYMENT-COMPLETE message back to KhnumM.

AFS cache pre-fetching (step 7) is only needed as a performance optimization. After application deployment, the server automatically enters the *operational state* in which it listens for START-CLEANING or PREPARE-CACHE-SNAPSHOT (not shown in Figure 2) messages. The "cleaning procedure" that **KhnumD** performs when it enters the *cleaning* state is very similar to the application deployment procedure:

1 Switch to runlevel 7, which stops all services.

2 Stop AFS daemon.

3 Clean up AFS cache.

4 Remove all the files that were created locally by applications since the applications were deployed on the server.

5 Remove all symbolic links and directories created by **KhnumD** when the server underwent application deployment.

6 Copy administrative AFS configuration files to standard location.

7 Restart AFS daemon.

8 Find and set up all the files and directories on the administrative AFS cell.

9 Switch back to runlevel 3, which restarts the system services (at this point only basic services will be started)

10 Send a CLEANING-COMPLETE message back to **KhnumM**.

When servers are in the "free" pool, they can potentially go though hard-drive re-imaging (that is, operating system re-installation) to produce a totally "clean" filesystem. As mentioned before, the time limitations during the cleaning stage are not as tight as during the application deployment stage, and therefore more time-consuming processes such as hard drive re-imaging and machine reboot are possible in the time allowed.

3.2 File System Mapping

The Khnum mapping process automatically creates: (a) symbolic links for read-only and application-wise read-write data, and (b) entire subtrees needed for instance read-write data. This process is driven by a configuration file consisting of *mapping 4-tuples* **(SharedDir, LocalDir, policy, script)** where:

SharedDir: specifies the remote root of the mapping, that is, where in the shared file system the image of additions to the local file system is rooted.

LocalDir: specifies a subdirectory on the local file system where the links/directories found in SharedDir are to be created (recursively).

Policy: specifies what to create on the local file system: subdirectories only (mktree), subdirectories and symbolic links to remote files (mkdir), or symbolic links to remote subdirectories and remote files (mklink).

Script: points to a post-mapping configuration script, that is, after the line in the configuration file is processed, this script will be called.

The different policies together with the post-mapping script allow for maximum flexibility with minimal changes to the local file system. The essence of the process is to create an image of the remote file system structure on the local file system using symbolic links as the preferred mechanism and creating entire subtrees whenever symbolic links are not appropriate. At the end of the process, the minimal number of new subdirectories will have been created, while most of the data will be accessible through symbolic links and the post-mapping script handles the special cases of files that need to be modified instead of replaced.

3.3 AFS Cache Initialization

The AFS aggressive caching mechanism ensures that in the "steady state" accessing frequently used *read-mostly*[12] files is almost as fast as if the files were local to the machine[13] [16]. However, the penalty for reaching this steady state can be high in terms of performance (for example, network traffic), in particular when many machines try to initialize their AFS cache simultaneously. Hence, a method to get many machines simultaneously to a known steady state, without choking the AFS file servers, is needed.

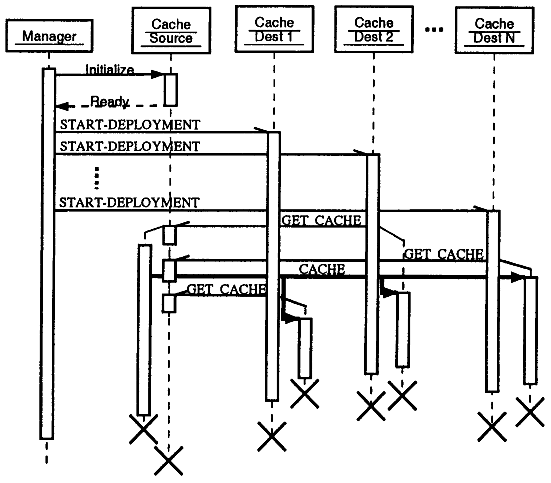
When the AFS daemon is started, it validates all the entries in the local cache. This means that if the cache contents are replaced with a valid content before the daemon is started, this new content will be used as if it was received by the daemon itself. Based on this observation and on our desire to achieve a steady state quickly, the AFS cache on new servers is initialized, before the daemon is started, with the contents of the AFS cache of a server already in the cluster (the *cache source*). Furthermore, to reduce network traffic and application deployment time, this pre-fetching is done by multicasting the cache contents to all the new servers simultaneously, using a multicast-enabled version of the TFTP protocol [17]. Figure 3 shows the message flow of the cache pre-fetch protocol, which is divided into the following three phases:

1 **KhnumM** selects a server as the *Cache Source* and sends it a INITIALIZE-CACHE request message to initialize the *cache service* and waits until the cache source is ready; the Cache Source server takes a snapshot of the cache content, starts the multicasting daemon, and sends a message back to **KhnumM** notifying that it is ready for multicasting the cache content.

2 **KhnumM** sends a START-DEPLOYMENT request (with the IP address of the Cache Source server) to each joining server. Each server then requests from the Cache Source the cache content. Once all requests are received (or a time-out expires), the Cache Source multicasts its cache contents to the new servers. Multicast TFTP multicasts cyclically all nonreceived blocks until the cache has been transmitted successfully to all the joining servers. Since pre-fetching is done for performance, we also considered multicasting in a *best effort* manner, that is, populating potentially only parts of the caches. **KhnumD** on the new servers performs the application deployment procedure, creating the directory structures and links according to the hosted site's definition as described in the previous section.

3 The new servers request the cache snapshot from the cache source and use it to initialize the AFS cache and resume the application deployment procedure.

Figure 3. Cache Pre-fetching



4. EXPERIMENTAL RESULTS

As mentioned previously, the prevailing hosting model at the time of writing is that of statically allocated servers. Adding additional servers to the initial setup is both expensive and time-consuming (a matter of days for adding additional machines, provided that they are physically available). Therefore, there is no value in a direct experimental comparison of dynamic allocation times and static allocation times.

On the other hand, we mentioned that adding many servers at once in our model may stress the AFS server (or servers), and we proposed a cache multicast pre-fetch model as a possible solution to this problem. We also conducted experiments that show the difference in "total application deployment time" (the time it takes to add a number of servers to a hosted site) with or without using multicast cache pre-fetching. The results of our experiment show that multicast cache pre-fetching significantly improves the scalability of our solution (in terms of the number of servers that can be added "at once" in single-digit number of minutes).

Our experiments were conducted using a set of 20 servers. Each server was a 600 MHz IBM IntelliStation with 512 MB of RAM running Linux RedHat 6.2. The machines were interconnected with a 100 Mbps fast Ethernet switch. In these experiments the *total application deployment time* was measured, that is, the time from when a START-DEPLOYMENT message was sent to the first server, to the time the last server has replied with a DEPLOYMENT-COMPLETE message.

Therefore, as described earlier, all the servers, that undergo application deployment, stop their current services and clear their "old" AFS caches before the application deployment for the newly hosted site begins. There are a number of factors that may considerably affect the measured results:

Number of servers: Impacts the total amount of data that, under normal circumstances, needs to be sent over the network. Every server independently accesses the shared file system. Thus, when a file is requested, that is not in the requester's AFS cache, the file's content will need to be retrieved from the shared file system server.

Network bandwidth and load: In many occasions, data from different sources may traverse the network, and affect the experiment's results. In our setup we use a dedicated private network for these experiments, minimizing the possibility that data other than our experiments' data will traverse the network (besides the usual "network noise").

Data set size: The total size of all the content for a given installation. We assume a server is already installed with the base operating system services and utilities, thus an installation refers only to any additional data that may be needed by the server, in order to serve as an equal member of the set of servers it is joining (for example Web servers of a specific Web site).

Cache size: The use of a cache depends heavily on the design and implementation of the shared file system. In our experiment, we use AFS. To *simplify our experiment*, we define our entire data set as "hot" (that is the most needed data, that needs to reside in the cache)[14]. That is, we define the cache size to be equal to or larger than the data set size, which should give us the effect of having our entire data set in the cache.

Shared file system server capacity: The more servers that need to be served from a shared filesystem server the greater the potential workload on the shared server, more so when new servers undergoing application deployment try to access the same content simultaneously. The capacity of the shared filesystem server could easily become a bottleneck when dealing with many servers.

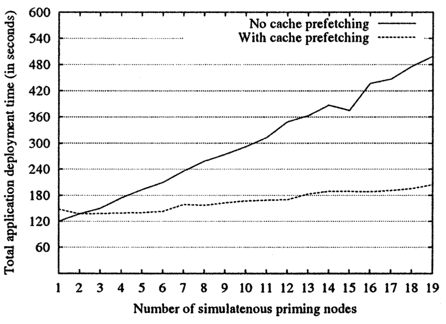
4.1 Description of the experiments

In order to get a clearer picture on the improvement that is provided by multicast cache pre-fetching, we tested our environment using two different data sets:

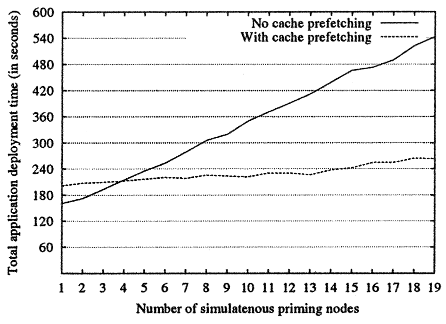
- The "Mayflower" experiment provides an example of a graphic intensive Web site. The site consists of 5421 files of small size (3 KB to 10 KB), adding up to 112 MB. The AFS cache size was set to 120 MB.
- The "ShowBoat" experiment is an example of a streaming video site, with a few relatively long movie clips (3 minutes to 4 minutes long). The site consists of 1855 files adding up to 162 MB. In this case the AFS cache size was set to 180 MB.

The experiments consist of deploying applications and data on up to n servers simultaneously ($0 < n < 20$) and measuring the total application deployment time as defined above. Our primary interest was to measure the effects of the multicast cache pre-fetching; hence tests were done with and without pre-fetching. From the hosted customer's perspective, the interesting measure is the time it takes for a new server to serve its first request (that is, the server becomes fully operational). To approximate this value, we "forced" all new servers to retrieve the entire content from the AFS server such that the AFS cache fills up. Since in both experiments the cache size was set to be larger than the data set size, the data set is exhausted before the AFS cache is

Figure 4. Application deployment experiments



(a) "Mayflower"



(b) "Showboat"

completely full, and thus, the entire data resides in the AFS cache (nearly simulating the effect of having the data set locally installed).

The measured results show that the multicast cache pre-fetching significantly reduces the total application deployment time (see Figures5(a) and5(b)). Both figures show that the application deployment time is a linear function on the number of servers simultaneously undergoing application deployment, where pre-fetching imposes a somewhat costlier setup time (that is preparing the cache content to be sent and sending it to a large number of nodes). However, the difference in slope (roughly a 6:1 ratio) shows that we easily overcame the somewhat higher setup penalty. Note that even though cache data is multicasted, each AFS client still needs to validate the AFS cache content with the AFS server, which causes a slight delay that increases as the number of servers undergoing application deployment increases, hence the small slope. The multicast cache pre-fetching model considerably improves the scalability of the application deployment process. This is especially important for "bursty environments"[15], where the system is expected to react fast to sudden peaks in demands, by adding a large number of servers simultaneously. It is also interesting to note that there is a certain threshold (which varies depending on the data size and number of files) under which it would be much better to avoid using multicast cache pre-fetching in order to obtain the best results. However, since the time difference in these numbers of servers is not large compared to the total time it takes to deploy applications on the servers, a simpler approach may mandate the use of multicast cache pre-fetching on any number of servers. The main reason for this approach being the fact that other than conducting experiments similar to the ones we have conducted, it is quite hard to estimate the threshold for each and every setup. Thus it may be practical to always use multicast cache pre-fetching, even for a small number of servers.

5. CONCLUSIONS

Océano manages a single collection of servers to provide hosting for multiple hosted sites, and dynamically reconfigures these servers to suit the current demand. Khnum manages the application deployment and server reconfiguration of each hosted site. An additional feature is Khnum's cache management, which supports fast concurrent addition of multiple servers for the same hosted site. By using multicast, Khnum achieves significant performance gains relative to earlier methods. We believe this model is flexible enough to host large families of applications that can serve requests independently on separate servers, for example, search engines that access a shared index from the shared file system.

So far, Khnum has focused on management of file system data. In the future, we would like to investigate similar pre-fetching techniques for databases as well. Another interesting direction for future investigation is geographical distribution of server farms, where the Océano system might dynamically allocate servers in the farm closer to the end-users (for example, Web sites transparently migrate from the US to Japan when the level of activity from Japanese users increases).

Trademarks

IBM and IntelliStation, are registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

Other company, product or service names may be trademarks or service marks of others.

Acknowledgments

We would like to thank Liana Fong and Srirama Krishnakumar for their helpful comments and ideas.

Notes

1. The methods by which to shut off a server from incoming requests until it becomes inactive, and the ways by which to determine whether a server is active or not are beyond the scope of this paper.

2. The discussion on sharing binaries ignores any licensing constraints.

3. All Océano component names allude to the ocean. According to Egyptian mythology, *Khnum is* the lord of the cool waters.

4. Our notion is that for many Web sites that serve static (or pseudo-static) data, there is a portion of the data which is hot (i.e. accessed by most users). With a good enough workload balancing component, most of the participating servers will need to access that hot data.

5. In the rest of the paper, we use interchangeably "applications" and "data", since both are stored as files in the shared file system, and treated in the same manner by Khnum.

6. Contrary to the requirement of rapid application deployment due to increased workload on a working site, removing a server from a hosted site does not require the same speeds. Therefore, re-imaging the server's disks is one option for a thorough "cleaning" of any residues or for installing a different version of the operating system, or a different operating system.

7. A *cache pre-fetch source* is an active server in the hosted site, which is instructed by Khnum to multicast the contents of its AFS cache to new servers being added to the hosted site. For more information on cache pre-fetching in Khnum, see section 3.3.

8. A UNIX System V term for the set of services and kernel state (single/multi user).

9. Every Khnum controlled server is configured to include the additional runlevel 7, which is essentially similar to runlevel 6 (reboot) with few modifications for running the setup scripts and automatically switching back to runlevel 3 (multiuser).

10. Some applications may require an additional configuration, which is achieved by customized configuration scripts. This is needed in those cases where there is a need for server-specific configuration.

11. The afsd daemon is stopped last as services run off AFS and afsd refuses to shutdown when there are remote files open.

12. We are assuming, for simplicity, that *read-mostly–* files may be updated by Tier-2 servers (or services) such as a daily forecast update, etc. Updating these files by a Tier-1 server can be handled by AFS, however, it may cause unpredictable results when applied to applications that are ill equipped to deal with sharing of files by multiple instances.

13. We assume that read-write data is stored on shared databases, which are beyond the scope of this paper.

14. In practice, data sets for hosted sites are much larger than the cache size.

15. Examples of such "bursty environments" are an on-line stock reports site during a stock rally, or a news site during a large scale event.

References

[1] K.Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalantar, S. Krishnakumar, D.P. Pazel, J. Pershing, and B. Rochwerger. *Océano - SLA Based Management of a Computing Utility.* In Proc. of the 7th IFIP/IEEE International Symposium on Integrated Network Management, May 2001.

[2] *The Official Red Hat Linux Reference Guide,* chapter 6: Package Management with RPM. `http://www.redhat.com/support/manuals/RHL-6.2-Manual/ref-guide/ch-rpm.html`.

[3] *Symantec Ghost - Product Information.* `http://www.symantec.com/ghost`.

[4] Z. Wensong. *Linux virtual server for scalable network services.* In Linux Symposium, Otawa, Canada, July 2000. `http://www.LinuxVirtualServer.org/ols/lvs.ps.gz`.

[5] R. Campbell. *Managing AFS: The Andrew File System.* Prentice Hall PTR, 1998.

[6] Sun Microsystems. *NFS:Network File System Version 3 Protocol Specification,* February 1994.

[7] P. J. Braam. *The coda distributed file system.* Linux Journal, June 1998.

[8] L.B. Huston and P. Honeyman. *Disconnected Operation for AFS.* In Proceedings of the USENIX Mobile and Location-Independent Computing Symposium, August 1993.

[9] B. Gronvall, A. Westerlund, and S. Pink. *The design of a multicast-based distributed file system.* In Proceedings of the Third Symposium on Operating Sytsems Design and Implementation, Feb 1999.

[10] G. Kuenning. *The Design of the SEER Predictive Caching System.* In Proceedings of the Workshop on Mobile Computing Systems and Applications, Santa Cruz, California, December 1994.

[11] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer and C. H. Hauser. *Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System.* In Proceedings of the 15th ACM Symposium on Operating Systems Principles, December 1995.

[12] J. Touch. *The LSAM proxy cache: a multicast distributed virtual cache.* In Proceedings of the Third International WWW Caching Workshop, Manchester, England, June 1998.

[13] T. T. Kwan, R. E. McGrath, and D. A. Reed. *NCSA's World Wide Web Server: Design and Performance.* pages 28(11):68-74, November 1995.

[14] J. Challenger, A. Iyengar, and P. Dantzig. *A Scalable System for Consistently Caching Dynamic Web Data.* In Proceedings of IEEE INFOCOM '99, March 1999.

[15] D. Quinlan. *Filesystem Hierarchy Standard (FHS) - Version 2.1.* April 2000. `http://www.pathname.com/fhs`.

[16] M. Spasojevic and M. Satyanarayanan. *A Usage Profile and Evaluation of a Wide-Area Distributed File System.* In Proceedings of the USENIX Winter 1994 Technical Conference, 17–21, San Fransisco, CA, USA, 1994.

[17] A. Emberson. *TFTP Multicast Option.* RFC 2090, Lanworks Technologies Inc., February 1997.

ENABLING PREOS DESKTOP MANAGEMENT

Tiago Cruz, Paulo Simões
CISUC – Dep. Eng. Informática, University of Coimbra
3030 Coimbra – Portugal
{tjcruz, psimoes}@dei.uc.pt

Abstract: Desktop management is probably the most resource-consuming task for the typical operations and support team, regardless of being frequently overlooked as *not as complex or specialized* as core network operations and management. Nowadays this scenario is even worst, since the increasing number and complexity of desktop systems was not matched by satisfactory management solutions – despite the relative success of products such as Intel's Landesk or Microsoft's SMS.

In order to address this problem, we are exploring a different approach to desktop management, through the design and implementation of the OpenDMS management framework. This open source framework differs from available products in several points, such as earlier remote management mechanisms (prior to operating system load), incorporation of existing open standards, a network-centric architecture, operating system neutrality and tighter integration between traditional PCs, Thin Clients and Network PCs.

In this paper we discuss the current status of desktop management solutions and we present an overview of the OpenDMS approach, including its most relevant technical foundations and an application scenario.

Key words PC networks, Desktop Management, OSS, Open-Source Tools

1. INTRODUCTION

Processing power was gradually taken off from central mainframes and distributed over each user's desktop, in the form of personal computers (PCs). However, this paradigm shift does not come without a price tag: more PCs to manage; hardware and software with ever-increasing complexity; more users requiring support; and more network requirements. Dropping prices and increasing user-friendliness made desktop PCs ubiquitous. Nevertheless, acquisition prices represent just a small fraction of the total cost of ownership (TCO) of such systems.

Among other factors, end-user training, helpdesk support and maintenance represent determinant contributions to heavy TCO costs [1-2]. Desktop management probably consumes the main share of human resources of the common IT department, easily outpacing server and network management altogether. Even considering that desktop management tasks are often less complex and specialized (and therefore less expensive) their cost is definitely not negligible.

There were several attempts to reduce these costs, mostly lead by the industry: commercial desktop management suites [3-4]; the DMTF initiative [5]; Thin Clients [6-7]; Microsoft's *Zero Administration for Windows* and Intel's *Wired for Management* [8]. However, despite these efforts, the current practice of desktop management is still not satisfactory.

Desktop management is usually based on some kind of *runtime agent* that supports remote operations such as monitoring, configuration and inventory management. This agent is then complemented with a centralized console used by the operator to access and manage several PCs.

One problem with this model is that only relatively healthy PCs can be managed. In the case of hardware, file system or operating system (OS) boot failure the agent will not load and local intervention will be required.

Another problem is related with the lack of integration. Management suites tend to be designed for specific environments (generally complete Windows PCs connected to local Windows Domain Controllers) and their functionality for alternative configurations is either reduced or non-existent at all.

Based on our own experience in the management of small PC networks with tenths or hundreds of PCs per location, we feel the current model might cover the basic needs in the management of groups of healthy and homogeneous Windows PCs. However, it provides no recovery solutions for problems prior to OS load and it lacks flexibility in the management of more heterogeneous environments, such as diskless PCs, Unix desktops, Windows Thin Clients and terminals based on Citrix Metaframe [9] or X-Windows.

In the OpenDMS project [10] we are exploring a different approach: the desktop becomes remotely manageable right after the initial Power On Self Test (POST) procedures; lighter PC configurations, such as Thin Clients, are explicitly supported and integrated in the management model; and mainstream standards and open tools are used to build an integrated network environment for the managed desktops.

The rest of this paper is organized as follows. The current status of desktop management is discussed in Section 2. The general architecture of the OpenDMS project is presented in Section 3. The next two Sections detail two of the most distinctive features of OpenDMS: PreOS management (Section 4) and the support for Thin Clients (Section 5). Section 6 presents a deployment scenario and Section 7 concludes the paper.

2. DESKTOP MANAGEMENT

Unlike other computer systems, for traditional PCs – "the most crash-prone computers ever built" [11] – reliability is less important than cost, performance or functionality. This approach opened the way for the *PC in everyone's desk* era and accelerated the spread of paradigms like client/server, distributed and workgroup computing. However, it is also truth that replacing mainframe-based computing by the current decentralized architecture, where the PC is the dominant specie, failed to fully deliver some of its promises, such as a dramatic reduction of TCO costs [1-2].

Over the years we have witnessed several industry-lead attempts to physically redesign PCs, to standardize PC management services and even to replace the classical full-blown PC desktop by new desktop paradigms, such as Thin Clients and Network PCs.

TCO and ecological concerns motivated several attempts to physically redesign the PC, with initiatives like the IBM PS/2 Energy Desktop [12]; Intel's Micro ATX and Flex ATX [13-16], and VIA Technologies' ITX and Mini-ITX [17-19] reference design prototypes. All these designs, as well as the PC Design Guides [20], from Intel and Microsoft, share the same core ideas: simpler PCs with reduced power consumption, enhanced ergonomics, flexibility, legacy-free design, small size and aesthetics.

Thin Clients and Network PCs were also proposed as a solution for the TCO problem. Reducing user freedom and moving data and programs back to a central system seemed the logical solution, but the paradigm failed to reach critical mass. In our opinion this was probably due to bandwidth limitations, the need to rewrite a large number of applications and the "proprietary/open-disguised" vendor approach in which software and hardware were actually closed and very platform-specific. This happened to projects like the X-Terminal, Microsoft's NetPC [6] and Oracle's NC [7]. Ironically, now that the hype has vanished, increasing network bandwidth and emerging computing technologies are finally creating the conditions for successful widespread deployment of truly open Thin Clients.

The Distributed Management Task Force (DMTF) [5] took the leading role in determinant initiatives for desktop management standardization, such as the desktop management interface (DMI [21]), the system management BIOS (SMBIOS [22]) and the Common Information Model (CIM [23]). DMTF standards provided ground support for initiatives like Microsoft's Zero Administration for Windows and Intel's Wired for Management, which tried to create a set of tools and technology resources to help systems administrators [8].

As already mentioned, desktop management products are based on OS-dependent management agents that directly support a predefined set of management operations, as well as direct remote desktop access for more unusual operations. Some of these tools also provide some level of integration with Windows management mechanisms, allowing integrated software distribution and configuration management for Windows domains. However, interoperability between products of different vendors is still reduced. Solutions like Intel's Landesk suite [4] and Microsoft's SMS [3] are effective only when managing Microsoft-only networks and following very product-specific management guidelines that difficult interoperability.

3. THE OPENDMS APPROACH

`OpenDMS` is a project where we try to complement the traditional desktop management model, addressing issues not covered by current practices. Two of the most relevant issues are the remote desktop management in the PreOS stage and the integrated support of alternative desktop models, such as thin client configurations and UNIX workstations.

To better understand how those issues interact with classical management technologies, the project also comprises a complete management framework (built using open standards and already available tools) that includes, for instance, a *runtime management agent* with many similarities with typical desktop management agents. However, it should be stressed that OpenDMS is not competing with tools like SMS or Landesk. In fact, in some situations it is possible and worthwhile to complement the OpenDMS framework with more sophisticated management agents from commercial suites.

3.1 PreOS Management and Platform Neutrality

The OpenDMS target platform is the common PC built with of-the-shelf hardware components. With very few exceptions (e.g. Apple machines or computers with very exotic hardware) this practically covers every PC sold nowadays.

Minimizing OS-dependency was also possible because many procedures are performed in an OS absent state: the so-called PreOS instant of the PC initialization sequence (see Figure 1). This opposes to typical desktop management solutions whose pre-boot capabilities are non-existent or, at most, limited to rudimentary OS installation or image distribution.

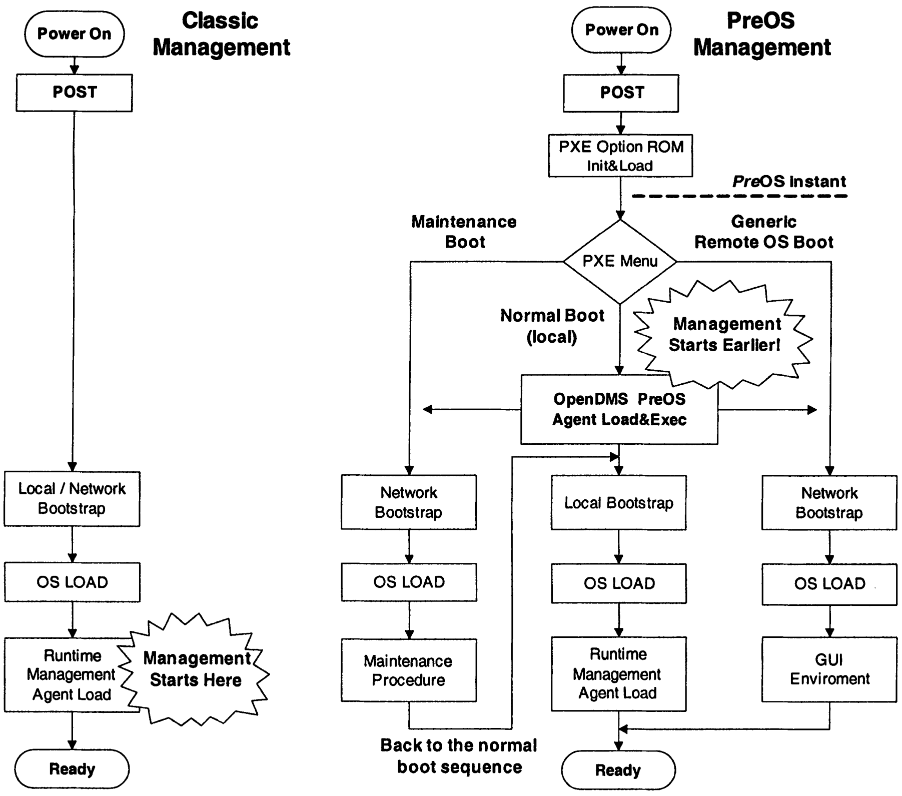


Figure 1. PreOS Management vs. Classic Desktop Management

With OpenDMS a PC becomes remotely manageable right after the initial POST procedures. Making a remotely controlled detour in this precise instant, in order to download and execute a special PreOS Agent or to boot an OS over the network, it is possible to manage a PC without a working OS. It is enough to have a network connection, a core set of operational hardware components (power supply, motherboard, memory, processor, network adapter) and standard remote boot firmware extensions.

OS-present operations are contemplated through the use of an additional management agent which is similar to those normally found in classical management suites, although somewhat different because several tasks normally executed at OS runtime are now performed by the PreOS Agent. OS load time and runtime load are considerable reduced, since the management agent imposes little load on the client system: there are no integrated web-servers or hardware asset management tools, with the only exception being the support for hot-pluggable devices.

The OS runtime component of OpenDMS was also designed to be as platform neutral as possible. In order to take advantage of specific OS features, the management agents are obviously OS-dependent. However, their management service interfaces follow a common specification. Furthermore, the whole environment – distributed file systems, remote terminal services, thin-client support services, etc. – is based on widespread standards and open tools available in a wide number of multi-platform versions, including at least the Windows family, FreeBSD and the most popular Linux variations.

3.2 Architecture

Figure 2 shows the OpenDMS client architecture. The first layer consists of PC hardware (of-the-shelf PC components) and related resources, such as Wake-On-LAN [24] (for remote off-hours maintenance tasks like backup and virus scanning) and system instrumentation.

The second layer consists of native firmware and standard extensions, including Preboot Execution Environment Boot ROM (PXE [25]), PC BIOS and DMI/SMBIOS. All these components are already available in current PCs, and their role in PreOS management will be discussed in Section 4.

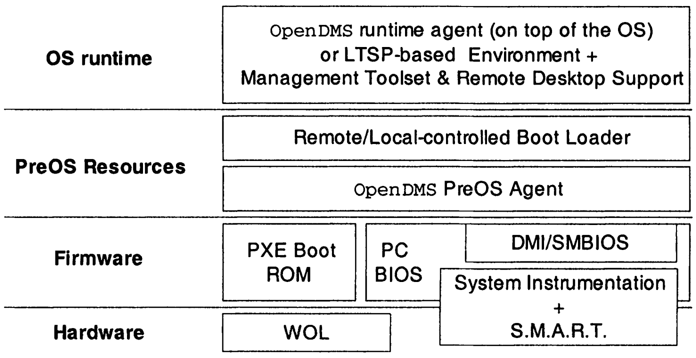

OS runtime
OpenDMS runtime agent (on top of the OS) or LTSP-based Environment + Management Toolset & Remote Desktop Support
PreOS Resources
Remote/Local-controlled Boot Loader
OpenDMS PreOS Agent
Firmware
PXE Boot ROM
PC BIOS
DMI/SMBIOS
System Instrumentation + S.M.A.R.T.
Hardware
WOL

Figure 2. OpenDMS Architecture (Client Side)

The PreOS Agent is the key component of the third layer. This agent is dynamically downloaded from the OpenDMS server and performs several management tasks in the absence of an operating system. After that, the PreOS agent passes control to the standard Boot Loader, which goes on with the boot of the operating system (either from the network or from local storage devices).

After OS load there are two possible configurations. If we have a traditional operating system, remote management is assisted by a classic runtime agent controlled by the management server. On the other way, it is also possible to build a thin-client configuration using additional resources provided by the OpenDMS framework. These resources are organized in the form of a modular thin-client platform specification built from commodity PC hardware, with support for network file systems and multiplatform connection capabilities that include Microsoft RDP-based systems [26], X/Windows servers, Citrix Metaframe [9] (client not available under GPL) and character-based protocols. Supporting access to local multimedia and storage, if needed, this platform is flexible enough to perform adequately in many usage scenarios, from desktop hybrid NCs-PCs to multimedia kiosks. Thin Client support will be discussed in Section 5.

On the server side one there are two distinct servers (Figure 3). The first is the core desktop management server. This server – built on top of GNU/Linux – uses PXE and either the classic TFTP (Trivial File Transfer Protocol) or its multicast-enhanced version (MTFTP) to upload the PreOS agent to the managed desktop. It remotely controls the execution of this PreOS agent (e.g. validating user authentication), as well as the runtime management agent. The systems manager uses a Web-based user interface and processed management information is organized using OpenLDAP [27].

The second OpenDMS server is dedicated to thin-client support. This server exports to the managed desktop a Linux-based thin-client environment built on top of distributed file systems (NFS [28], Samba/SMB [29] and NBD [30]), eventually existing local file systems and remote desktop clients (X-Windows, character-based protocols, Windows Terminal Server/RDP [26], VNC [31] and Citrix Metaframe). PXE and TFTP (or MTFTP) provide support to remotely boot an OS image. This set of services is either provided by a single machine or spread across several servers. In the specific case of Windows Terminal Server and Citrix Metaframe the service has to be provided by a native Windows server. To configure and maintain this second OpenDMS server the systems manager also uses OpenLDAP and a Web interface.

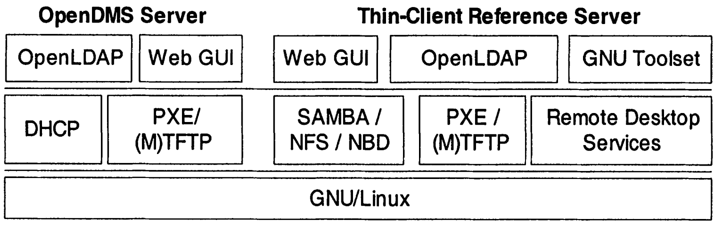


Figure 3. OpenDMS Architecture (Server Side)

4. BUILDING PREOS MANAGEMENT

The PreOS stage of the PC initialization process still remains a sensitive area for the great majority of available desktop management tools, probably because until a few years ago there was no built-in firmware support for PreOS management mechanisms. However, modern PCs support a specific kind of special-purpose firmware extensions through the use of option ROMs, normally embedded in hardware such as Network Boot ROMs. Those extensions were originally designed just for remote deployment of operating systems over the network, but there is no reason not to use them to force the download of more specific software, such as a PreOS Management Agent.

Ordinary Boot ROMs are incapable of dealing with the needs of PreOS management due to design limitations. Standard TCP/IP Boot ROMs are built around unicast-based protocols with remote OS load from a single point in mind. Without any kind of load-balancing or fault-tolerant features, they are too unreliable to use as a PreOS management tool. Furthermore, implementations from different sources tend to use non-standard downloadable OS image formats, leading to incompatibilities. These issues were solved by PXE Boot ROMs, designed with interoperability and reliability concerns in mind. They extend the basic remote boot functionality, providing a basic OS-absent program execution environment.

PXE Boot ROMs were created in the context of the Intel Boot Initiative [32], circa 1998, and meanwhile they made its way into almost every kind of PC network interface currently produced. At the time of this writing Intel is working in the specification of an Extensible Firmware Interface (EFI), as a replacement for the currently available BIOS architecture for IA-32 systems (already available for IA 64 systems), that will include a PXE-compliant facility [33]. PXE is the standard for remote boot ROMs.

Figure 4 presents the modular structure of the PXE specification. Integrated in the system's firmware as an option ROM, the PXE BIOS extension provides four APIs designed to be used by any kind of Network Boot Program:

- PreBoot API. This API provides the means to control the entire PXE environment, from deactivation/activation of the embedded TCP/IP stack to access to DHCP packet data received during the boot negotiation process.
- (M)TFTP API. This API supports TFTP or MTFTP-based file transfer functions.
- UDP API. This API provides basic UDP-supported I/O functions.
- UNDI (Universal Network Device Interface). This API is an abstraction layer that works as a universal device driver, allowing the other APIs to work independently of the available network interface hardware. This layer also allows the creation of universal device drivers partially supported by the PXE UNDI API.

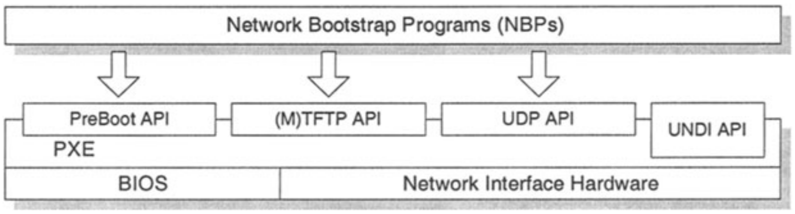


Figure 4. The PXE Boot ROM

It should also be stressed that PXE provides fault-tolerance and load-balancing mechanisms: there are several boot servers, allowing the client to boot from one of them according to several load-balancing techniques.

Once the system is turned on and the POST routines are executed, the system's BIOS begins searching and initializing all option ROMs. The PXE code will then force the download and execution of the PreOS Agent.

Figure 5 shows the key execution steps of the PreOS Agent. Once initiated, it tries to establish communication with the management server in order to receive enough information to proceed with the remaining tasks. Next, if the server informs the agent that the workstation is allowed to boot, it will make a system integrity check and will report the status together with the hardware configuration detected (using DMI/SMBIOS or direct access methods). If the system status is satisfactory and the hardware inventory list is coherent with what the server has in its own database, the agent will be authorized to proceed to the next step. Otherwise the system may lock the boot process (avoiding further progress and shutting down the system, if needed). After an optional user authentication step, the PreOS Agent will pass control to the bootstrap loader, allowing the system to proceed with its normal boot sequence. Since the specific OS image to be loaded may be selected by the OpenDMS server, it becomes possible to have a PC with multiple personalities (Windows desktop, Unix workstation, thin-client, etc.) according to the user and the circumstances. This feature can also be used to load an OS-environment specifically designed for recovery operations.

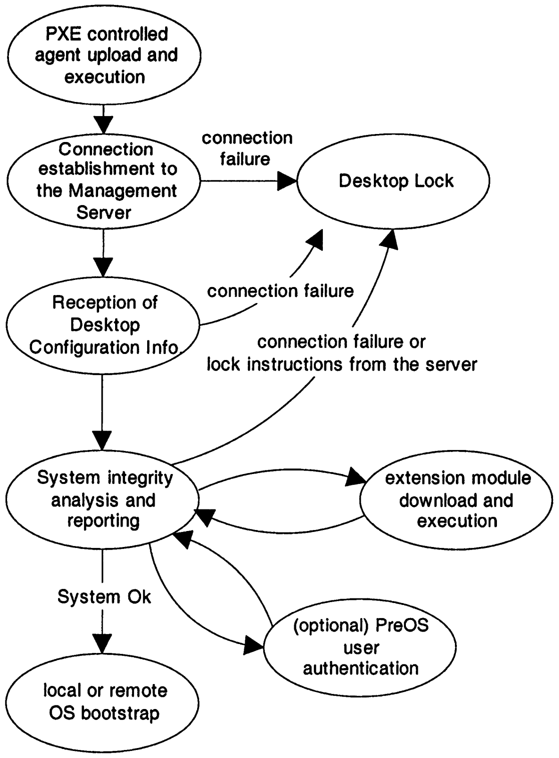

PXE controlled agent upload and execution
Connection establishment to the Management Server
connection failure
Desktop Lock
Reception of Desktop Configuration Info
connection failure
connection failure or lock instructions from the server
System integrity analysis and reporting
extension module download and execution
System Ok
(optional) PreOS user authentication
local or remote OS bootstrap

Figure 5. PreOS Agent Execution

Thanks to the PreOS Agent, common tasks like user authentication and hardware inventory can be performed in the PreOS stage of the PC boot sequence, without depending on a runtime OS environment. Furthermore, the PreOS Agent is developed around a modular approach in which functionality can be extended by adding third-part downloadable modules. Two of the most interesting modules already developed are *memtest86* and *S.M.A.R.T.* (System Monitoring, Analysis and Report Technology), which provide extensive memory and hard drive diagnostic features accessible to the PreOS management toolset.

If the normal boot sequence is followed and an OS is loaded, the OpenDMS runtime agent (or eventually a more feature-rich runtime agent from a commercial third-party provider) will ensure that the system is still remotely manageable during normal operation.

5. THE OPENDMS THIN CLIENT APPROACH

As already mentioned, Thin Clients and Network PCs were once seen as an interesting paradigm to reduce the complexity of the user desktop and move data and programs back to a central system. However, the proprietary approach followed by products like the X-Terminals, Microsoft's NetPC and Oracle's NC, as well as technological limitations, prevented the paradigm from reaching critical mass.

But nowadays such weaknesses are addressable using a different approach: why not build a Thin Client based on common PC-hardware with the same capabilities of its proprietary counterparts? Bandwidth is now available and current PC hardware is far more sophisticated than the one found in most proprietary solutions. PXE also fits these purposes: despite being used mostly for OS deployment (booting a small mini-environment that performs OS-deployment procedures), PXE supports the direct boot of an OS through the network. Therefore, its role is not limited to the execution of the PreOS Agent (which is also important in the selection of the correct "desktop personality"), since it is possible to use PXE and TFTP to load a remotely-provided Thin Client environment.

OpenDMS includes several modules – most of them adapted from the LTSP Project [34] – that can be combined to build custom-tailored Thin Client configurations. Four distinct operations modes were considered:

– Diskless GUI-based Thin Clients, in which the system does not need any kind of local fixed mass-storage device, booting from a cluster server, accessing a remote file system using NFS, and using a small amount of local memory as a *ramdrive* to cache frequently used data. It uses a locally executed X-server to provide display facilities to applications ran on the remote cluster server or to provide a graphic environment in which an RDP or Citrix client can be used to access a Windows Terminal Server system. Access to local audio, serial, parallel and removable storage is possible on such configurations, thanks to the use of smart redirector methods (as the network block device protocol). This mode can also be use to build intelligent, remotely managed multimedia kiosks or point-of-information terminals. Normal PCs may also benefit from this mode of operation, allowing them to behave like they had an alternate Thin Client personality, depending on a user-selectable boot option in a PXE-provided menu. Additionally, this operation mode provides a special remote helpdesk

mode in which a normal PC can boot directly into a browser window pointing to a helpdesk request service page.
- Network Computer, a natural extension of the Thin Client mode in which the system is allowed to possess local fixed mass-storage facilities used to store frequently used programs and configuration data.
- Network Appliance. In this mode, a PC boots from the network with a special purpose, self-contained mini-Linux distribution in order to transform a simple PC in a network appliance, such as a print server with no local storage facilities, a remotely manageable Network Attached Storage system or a basic Router/Firewall.
- Recovery Mode, a special operation mode devised to allow the remotely initiated recovery of a PC with a seriously damaged file system. A normal PC can boot into this mode in order to download and restore a previously-uploaded file system image from a cluster server. This procedure depends on a set of GPL licensed tools (like the parted partition manipulation tool) executed under the control of automated scripts on a minimal self-contained Linux environment booted from the network.

Thin-clients do not require top-notch hardware. Even a fifth generation *x86* system (e.g. systems based on the old 90 MHz Intel Pentium, with a PCI bus, a PXE-capable network interface card and 32MByte of RAM) is able to provide a reasonable hardware platform to build an OpenDMS Thin Client. This way hardware lifetime is significantly extended.

6. DEPLOYMENT SCENARIO

Figure 6 shows an example of an OpenDMS managed environment. In this environment there are Linux Workstations and Windows Desktops with a "normal" configuration based on a local OS. There are PCs that have multiple personality, dynamically selected according to the circumstances. There are also diskless PCs, used as thin clients, and one managed network appliance (a small router/firewall). There is one OpenDMS server dedicated to the management tasks and another server for Thin Client support.

For PCs with local OS and fixed configuration (Windows and Linux desktops) the PreOS Agent is primarily used to diagnostic hardware or file system failures. In the case of severe hardware failure the system may be locked waiting for local intervention. In the case of file system or OS failure the system might boot a special recovery environment. This special mode is a lightweight Linux installation, provided by the "Thin Client Server", with recovery tools from the GNU Toolset and means to allow the user to perform helpdesk request operations from the damaged PC itself, even in the case of hard drive failure or corruption. It is also possible to upload and locally install a previously prepared OS image of the system stored in the "OS Images Database" (eventually overwriting the corrupted OS). After successful OS load, the runtime management agents provide the means to perform remote management on these desktops.

Users of multiple-personality desktops are authenticated by the PreOS Agent. Based on the user ID and the adopted policies, the OpenDMS server enforces one of several available configurations: a complete Windows/Linux machine booting from a local OS; a Network PC booting from the network but still using local storage devices; or a Thin Client working like a diskless system in GUI mode and allowing access to the Windows Terminal Server or a more generic Application Server. Remotely loaded OS images are provided by the Thin Client Server, and runtime network file systems might be provided by an additional Network Attached Storage device using SAMBA or NFS.

Diskless PCs are a simplification of the multiple-personality desktops: although it is also possible to choose from several available OS images, only diskless configurations are possible (e.g. web browsers, X terminals or Windows diskless terminals). Nevertheless, using NBD [30], it is still possible to use local devices such as CD drives, microphones or speakers.

The network appliance loads a task-specific remote OS image each time it is booted. There are no local file systems to maintain, and router hardware failures are easily solved: get a new computer (or replace the faulty hardware), change the appliance ID in the OpenDMS servers and turn on the new computer. It will automatically load the router software and start working.

The OpenDMS server includes the DHCP server – to manage the IP pool reserved to the client systems and to provide PXE discovery services – the PXE server and a TFTP service to upload the PreOS Agent. This server controls both the PreOS Agent and the runtime management agents.

The Thin Client Server is used just to maintain OS images that may correspond to "local OS images" (i.e. uploaded to the desktop and locally booted) or operating systems directly booted from the network: Thin Client configurations, special recovery environments and network appliances designed for specific tasks.

Network file systems and remote desktop services are conceptually part of the Thin Client Server (see Figure 3). However, in the presented scenario they are provided by three distinct specialized servers: a file server using Samba and/or NFS, a generic application server (based, for instance, on X-Windows or Web interfaces) and a Windows Terminal Server or a Citrix Metaframe Server to run Windows based applications (e.g. Microsoft Office) from thin clients.

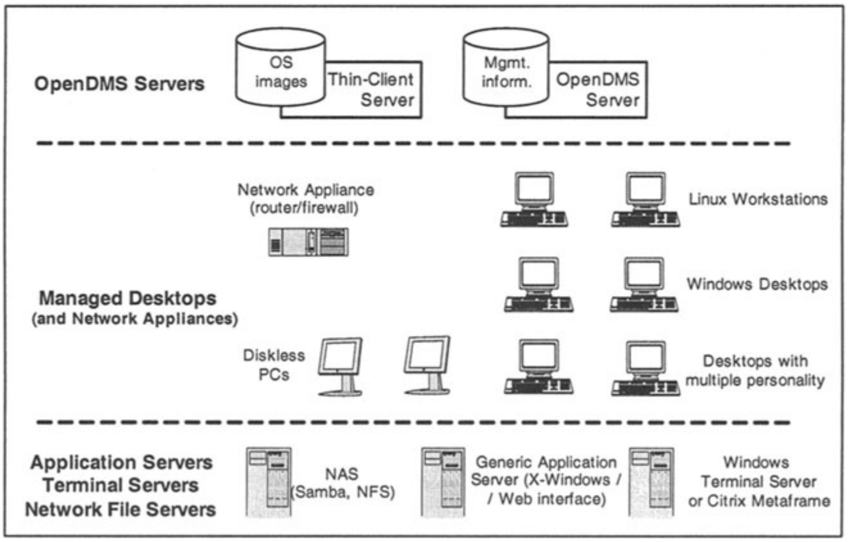

OpenDMS Servers
OS images
Thin-Client Server
Mgmt. inform.
OpenDMS Server
Network Appliance (router/firewall)
Linux Workstations
Windows Desktops
Managed Desktops (and Network Appliances)
Diskless PCs
Desktops with multiple personality
Application Servers
Terminal Servers
Network File Servers
NAS (Samba, NFS)
Generic Application Server (X-Windows / / Web interface)
Windows Terminal Server or Citrix Metaframe

Figure 6. Example of an *OpenDMS* Managed Environment

There are several relevant issues not explicitly exposed by the presented scenario: the total amount of servers required to build an operational environment; the potential usage of load-balancing mechanisms; and integration with commercial desktop management suites.

Figure 6 presents two OpenDMS servers and three additional servers providing network file systems and remote desktop services. Five servers might be overkill for small networks and not enough for larger infrastructures. Our experience shows that it is possible to concentrate the support of 10 to 20 thin-clients in one desktop management server and one thin-client server. Even using just these two servers (two basic Celeron 400 MHz/256 MByte machines, in our specific testbed) it is possible to provide a basic degree of resilience and load-balancing, since several services (e.g. NFS, PXE, TFTP, MTFTP and recovery support) are easily and naturally replicable in both machines. In larger systems this natural resilience might be combined with the distribution of services across specialized servers located in strategic network locations, increasing the number of servers to manage but achieving a higher level of performance, availability and scalability.

So far we have not invested much effort in the potential integration with already existing desktop management platforms. Nevertheless, PreOS management and specific support for thin-clients are orthogonal to current approaches, and therefore it is possible to use OpenDMS and other platforms at the same time without overlapping each other (except for the runtime management agents). High-level integration could eventually be achieved using the LDAP-organized management information and policies kept by the OpenDMS server. Tighter integration would require adapting current platforms in order to control the PreOS Agent, whose interface is simple and well documented. Extending this agent is also simple, since it comprises an interface for additional software modules.

7. CONCLUSIONS AND FUTURE WORK

PC ubiquity, as well as PC increasing complexity, keeps desktop management as one of the most resource-consuming areas of operations and support. Despite several industry-lead initiatives the truth is that classical desktop tools target just a fraction of the whole problem.

In the OpenDMS framework we are searching for alternative solutions for this problem. Rather than directly competing with the best features of commercial products – such as their suitability for integrated management of healthy Windows domains – we are more interested in complementing these tools with less common approaches.

One of these approaches is PreOS management. PreOS management allows more complete control of user-available computing resources. With PreOS management it becomes possible, for instance, to perform early hardware checkup, user-authentication procedures and basic file system verification. Furthermore, PreOS makes possible to flexibly select the PC personality (full-blown PC, thin client, network PC, network appliance, etc.) according to the identity of the user and the status of the system. Relatively recent industry standards, such as PXE, DMI BIOS and SMBIOS finally made possible the development of capable PreOS management agents. However, to the best of our knowledge, OpenDMS is the first tool that provides such functionality.

The thin client approach also differentiates OpenDMS management. This approach corresponds to the notion that thin clients are indeed the right answer to many situations, with considerable advantages over traditional PCs: increased control, increased reliability and less demanding hardware requirements. Although this might sound as *déjà vu*, considering the vanishing hype around previous thin client proposals, our approach to the subject does bring some new elements. First, we believe that some of the problems that affected previous approaches are now less constraining than before. LAN bandwidth is becoming cheaper and cheaper, for many uses a modern Web Browser in sufficient and even remote desktop technology has been remarkable improved in the last few years. Second, we are not going after a proprietary or "100% thin client" approach. Instead, we are using regular PC hardware and open source software to build flexible thin clients that may fit many configurations, from diskless computers to thin clients or even network appliances. In fact, it becomes even possible to select the "desktop personality" at boot time, according to the intended use.

OpenDMS is not limited to these two aspects, and its architecture also embraces more classical management mechanisms, such as runtime management agents. However, at least for now, we just use those runtime management mechanisms to better understand the potential implications of PreOS management in classic management practices. Implemented runtime agents (for Windows and Linux) just provide remote desktop access (built upon VNC), basic health monitoring, software inventory, troubleshooting and log facilities. Systems administrators needing additional features can use OpenDMS just for PreOS management and thin client deployment, while keeping commercial products for runtime management of Windows PCs.

OpenDMS already includes fully functional implementations of the PreOS Agent and the reference Thin Client environment. The *proof-of-concept* server is also developed but still needs to be productized (e.g. with simpler installation procedures and user friendlier interfaces). We are currently testing the platform using a few dozens of classroom PCs, in order to study scalability, robustness, and hardware and bandwidth requirements. Further work on security is also planned: complying with the PXE specification, the current implementation uses potentially dangerous protocols such as TFTP, requiring either the usage of more careful network configurations or the evolution for more secure network services.

REFERENCES

[1] Gartner Consulting Group, "TCO Analyst: A White Paper on GartnerGroup's Next Generation Total Cost of Ownership Methodology", 1997, pp. 19-20

[2] S. Heilbronner, R. Wies, "Managing PC Networks", IEEE Communications Magazine, October 1997

[3] Microsoft Systems Management Server (SMS) Homepage, http://www.microsoft.com/smserver/

[4] Intel Landesk Homepage, http://www.intel.com/network/products/landesk/

[5] DMTF, Distributed Management Task Force Homepage, http://www.dmtf.org

[6] Microsoft Corporation, "Network PC System Design Guidelines v1.0b", 1997 (available at http://www.eu.microsoft.com/hwdev/archive/netpc.asp)

[7] Willian Blundon, "Deciphering the NC World", JavaWorld, March 1997 (available at http://www.javaworld.com/javaworld/jw-03-1997/jw-03-blundon.html)
[8] L. Weller, "Reducing TCO with Intel WFM and Microsoft ZAW initiatives", Intel Developer Update Magazine Issue 17, February 1999
[9] Citrix, Citrix Metaframe Homepage, http://www.citrix.com/
[10] Tiago Cruz, Paulo Simões, "Rethinking Desktop Management", Proceedings of APNOMS'2002, Jeju, South Korea, September 2002
[11] Tom Halfhill, "Here's why today's PC's are the most crash-prone computers ever built – and how you can make yours more reliable", Byte Magazine, April 1998
[12] IBM Corporation, "IBM PS/2 E product information", IBM DC 00210-00, 1993
[13] N. Sumrall, "High Concept Comes to the PC: Form, Function and Fashion", Report from Intel Developer Forum, fall'99, Intel Developer Update Newsletter
[14] Intel Corporation, "MicroATX Motherboard Interface Specification", 1997
[15] Intel Corporation, "FlexATX Addendum V 1.0 to the microATX Specification", 1999
[16] Intel Corporation, "Ease of Use Initiative – Concept PC", 2002
[17] VIA Technologies, "ITX Mainboard Specification White Paper", 2001
[18] VIA Technologies, "Mini-ITX Mainboard Specification White Paper", 2002
[19] VIA Technologies, "Information PC reference design", 2001
[20] Intel Corp, Microsoft Corp, "PC Design Guide Website", http://www.pcdesguide.org/
[21] DMTF, "Desktop Management Interface (DMI) v2.0s", 1998
[22] DMTF, "System Management BIOS Reference Specification v2.3.2", 2001
[23] DMTF, "Common Information Model (CIM) Specification v2.2", 1999
[24] AMD Corp., "Magic Packet Technical White Paper", 1995
[25] Intel Corp, "Preboot Execution Environment (PXE) specification version 2.0", 2000
[26] Mark Chapman, "Rdesktop project", http://rdesktop.sourceforge.net
[27] OpenLDAP, http://www.openldap.org
[28] Hal Stern, "Managing NFS and NIS", O'Reilly & Associates, 1992
[29] Samba Project, http://www.samba.org
[30] NBD Project, http://nbd.sourceforge.net
[31] AT&T Cambridge Labs, "Virtual Network Computing", http://www.uk.research.att.com/vnc/
[32] Intel Corp, "Intel Boot Initiative program", 1998
[33] Mike Henry, "PXE Manageability Technology for EFI", Intel Developer Update Magazine, October 2000
[34] LTSP, "Linux Terminal Server Project", http://www.ltsp.org

SESSION 7

Peer-to-Peer and Overlay Networks

Chair: James Won-Ki Hong
POSTECH, Korea

PEER-TO-PEER OVERLAY NETWORK MANAGEMENT THROUGH AGILE

Jan Mischke[1] and Burkhard Stiller[1,2]
[1]ETH Zurich, Switzerland; [2]University of Federal Armed Forces, Munich, Germany

Abstract: Currently, state of the art peer-to-peer (P2P) lookup mechanisms actively create and manage a peer application layer overlay network to achieve scalability and efficiency. The proposed mechanism AGILE (Adaptive, Group-of-Interest-based Lookup Engine) extends this management approach, adapting the overlay network such as to bring requesting peers and desired lookup items close together, reducing the number of hops and, thus, latency as well as bandwidth requirements for a lookup. At the same time, AGILE introduces mechanisms to build a fair system.

Key words: Peer-to-peer (P2P) Lookup Services, Overlay Network Management, Scalability

1. INTRODUCTION

Peers in a P2P system communicate on a logical overlay network among them. Some existing systems, *e.g.*, Gnutella, build this overlay network at random, adding (or removing) links and nodes in an uncontrolled way through arbitrary ping requests and pong responds. Unfortunately, the orderless structure requires a non-scalable flooding mechanism for lookup, and the path lengths and node degrees can become large. More sophisticated approaches, like Tapestry [16], Pastry [4], or Chord [1], actively manage the overlay network such as to ensure robustness and alleviate lookup and request routing. These systems, however, pay little attention to the heterogeneity of peers with respect to their interests and capabilities.

The proposed mechanism AGILE (Adaptive, Group-of-Interest-based Lookup Engine) creates and maintains an overlay network according to specific topological requirements for P2P lookup. It additionally adapts the network over time so that groups can form according to common interests, improving the lookup performance, while at the same time ensuring fairness.

Essential topological requirements are derived in Section 2, while Section 3 discusses related work and identifies major gaps to those requirements. Section 4 introduces and evaluates the proposed approach AGILE, before Section 5 concludes.

2. REQUIREMENTS

It is straightforward to require that a P2P system be scalable and make efficient use of system and peer resources, namely *memory*, *processing power*, *bandwidth*, and *time/latency*. With up to 96% of local peer node resources being idle [2], bandwidth and user time, or latency in the technical system, are most crucial and will be considered in more detail in the next subsection. Furthermore, the system should ensure a proper *load balancing* in that it be *fair*, involving peers according to their use of the system and in that it pay attention to the *heterogeneous capabilities* of peers. Finally, a P2P system has to be *robust* to frequent node joins and leaves and link failures.

In general, network topologies can be characterized through their degree of symmetry, the network diameter, the bisection width, the average node degree, and the average wire length [6]. The functional and performance requirements (see above) determine the desired target characteristics.

- **Symmetry:** Only symmetric topologies are appropriate for true peer-to-peer systems as only in this case all peers are equal from a topology point of view. At the same time, symmetry assists load balancing. Examples of symmetric topologies include rings, buses, hypercubes, complete meshes, cube-connected circles, or k-ary n-cubes. Measurements as stated in [13], however, prove a huge heterogeneity among peer nodes in terms of their uptime, average session duration, bottleneck bandwidth, latency, and the number of services or files offered, so that server-like roles in a P2P network may be advantageous.
- **Network Diameter (D):** The diameter of a network is defined by the number of hops required to connect from one peer to the most remote peer. It strongly influences latency and bandwidth.
- **Bisection Width (β)**: The number of connections from one part of the overlay network to the other define its bisection width. Assuming proper load balancing, the maximum throughput of the network is proportional to the bisection width, and there is a direct relation with fault tolerance: the bisection width determines the number of links that have to break before the system goes down or, at least, operates only as two partial systems.
- **Node Degree (d):** The node degree is defined as the number of links that each peer has to maintain. The node degree can be a significant inhibitor for scalability. The node degree determines the size of the routing table on each peer with the proportional impact on memory consumption and processing power.
- **Wire Length (τ):** The wire length is the average round trip delay of a connection, contributing to the latency in the system.

It is particularly important to have a look in detail at latency and bandwidth consumption for a lookup request. The latency L for a lookup request is defined as

$$L = \bar{\tau} \cdot n_h = \bar{\tau} \cdot D \cdot (1 - f_p)$$

where n_h is the number of hops for a request and the *pruning factor* f_p denotes the average percentage of the maximum number of hops that a request does not need to travel, because it has been pruned off before. The pruning factor can be calculated from the pruning probability at each hop $p_{p,i}$ (i.e. the probability that the requested item is found at that hop) through

$$fp = 1 - \frac{1}{D}\sum_{i=1}^{D}\prod_{k=0}^{i-1}(1-p_{p,k}); \qquad i, k \in \aleph$$

The pruning probability $p_{p,0}$ at node 0, the requesting node, will usually be zero. Hence, three important factors determine the latency time: the network diameter, the average round trip delay, and the pruning probability. It is possible to increase the pruning probability in a topology by exploiting knowledge on the peers' interests.

In addition, the total bandwidth B required for a lookup request is

$$B = B_{RP}n_h(d-(d-1)\varepsilon_{route}) = B_{RP}(d-(d-1)\varepsilon_{route}D(1-f_p)$$

where B_{RP} denotes the bandwidth or size of one request package, n_h (as above) the number of hops, d the node degree, and ε_{route} the *routing efficiency*. The routing efficiency is defined to be 1 if only one node has to be contacted at each hop and 0 if all nodes have to be contacted. In that sense, Gnutella with its flooding approach has a routing efficiency of 0, whereas consistent hashing algorithms like Chord [1] have a routing efficiency of 1.

As for the latency, the network diameter and the pruning probability influence the bandwidth requirements (and scalability) in a major way. Furthermore, the routing efficiency plays a significant role. The equation also suggests that the node degree be kept low. However, this applies only if the routing efficiency is smaller than 1, as a lower node degree automatically entails a larger network diameter.

3. RELATED WORK

Tapestry [16], Pastry [4], Chord [1], and CAN [9] determine the systems most closely related to AGILE. Their common theme is that they arrange lookup items or *keys* (such as content files, services, or peer node addresses) and peer nodes in the same identifier space. Subsequently, they hand over the responsibility for holding a key with a certain identifier to a peer with a numerically close identifier. This enables them to simply route a lookup request message at each node towards a neighboring node with a closer node ID, achieving a routing efficiency of 1. All of these lookup services propose hashing to map lookup item names and nodes (IP addresses) onto the identifier space. Firstly, the hash function is globally known, ensuring the same mapping for each request for or insert of a key. Secondly, hashing results with high probability in unique IDs. Thirdly, the pseudo-randomness of the hash function uniformly distributes keys and nodes in the identifier space.

The main difference between these approaches is the topology they build to arrange peers properly so that they can route closer to the desired ID, while meeting major requirements to a good topology (cf. Section 2). Furthermore, they apply different algorithms to constructing, maintaining, or managing this topology.

- *Tapestry:* Tapestry builds a Plaxton mesh. IDs are represented as numbers with a sequence of digits to a base b. At each hop, a request is routed toward a node, whose ID matches the search key in one digit more than the previous node's ID did, starting at the last digit (suffix-based routing). The management of the overlay network focuses on fault tolerance: soft stating, time-outs, and republishing to ensure accuracy of the information, triple redundancy and back-pointers in the routing tables, use of several "root" servers, i.e. redundancy in the nodes responsible for a key.
- *Pastry:* The basic concept and topology is the same as for Tapestry, except that prefix-based routing instead of suffix-based routing is applied. The fault-tolerance focus is replaced by an apparently more light-weight scheme.
- *Chord:* Chord arranges keys and nodes around an identifier circle. The node with the largest number preceding the search key is responsible for holding it. Nodes maintain overlay links to a couple of successors and fingers as chords in the circle in exponentially increasing distances from the respective node, enabling to halve the remaining ID search space at each routing step. This becomes very similar to Tapestry and Pastry when choosing a base of 2 in the latter ones.
- *CAN:* CAN is based on a d-dimensional Cartesian coordinate space (or d-torus) separated into bins of varying size to implement a distributed hash table. Other than Tapestry, Pastry, and Chord, the node degree is thus fixed.

HyperCuP [14], takes a different approach. Like in Gnutella, flooding is used for the lookup. However, the overlay network is actively managed as a hypercube with good symmetry, diameter, and bisection width properties. It seems to be possible to also use a hashing scheme to improve routing efficiency. Furthermore, the authors propose an ontology-based routing scheme for the same reason.

Table 1 compares these systems (including AGILE) with respect to major requirements from Section 2 according to the developers' information or information deduced from algorithm descriptions. For all systems N denotes the number of nodes in the system, b and d are design parameters, where b is the base value for a digit representation of hash keys (where used) and d is the dimensionality of the CAN torus.

All mechanisms except CAN achieve logarithmic scalability with respect to the path length of a routing request or the network diameter. Chord does not allow to trade off the node degree for a lower number of hops by choosing a base higher than 2. Particularly for PC nodes, a higher node degree can easily be accommodated while allowing to reduce bandwidth and latency. While the existing algorithms only have a statistically inherent pruning probability related to their base b, they all achieve a routing efficiency of 1 - HyperCuP with its flooding mechanism being the obvious exception. The node degree scales logarithmically except for CAN, where it even remains constant. However, this limits the flexibility when a network grows. As to the wire length, Pastry, Tapestry, and CAN introduce optimization schemes. The methods and simulations to obtain figures for the stretch (*i.e.*, the relative latency of overlay routing compared to IP routing) are too different to base a good comparison on them. Several further proposals have been made to address the issue of wire length separately [3], [17], [18], [11], and [10].

Table 1 compares fault tolerance in terms of two dimensions, key redundancy and link redundancy. While replication can be controlled by the application, the lookup algorithms propose different mechanisms to conveniently place k replicas. For increased fault tolerance with respect to routing, Pastry and Chord keep redundant state information for closest neighbors or successors in the ring, respectively, whereas CAN and Tapestry set up (3 or r, respectively) independent entire routing tables. Tapestry further increases fault tolerance through soft-stating and heart-beat protocols.

Maintenance complexity, which is the number of messages per node join or leave, scales logarithmically for all systems but CAN. More detailed quantitative information is not available, but it is obvious that Tapestry with its surrogate routing and routing table redundancy will exhibit a higher complexity than the other mechanisms. As all algorithms build a probabilistic but fairly symmetric topology heterogeneity is only partly addressed by Tapestry through the BROCADE extension [17], and by CAN through load-dependent bin splitting.

Table 1. Comparison of Lookup Mechanisms

Characteristic	**Tapes-try**	**Pastry**	**Chord**	**CAN**	**Hyper-CuP**	**AGILE**
Network diameter	$O(\log_b N)$	$O(\log_b N)$	$\approx \log_2 N$	$O(dN^{1/d})$	$O(\log_b N)$	$\approx \log_b N$
Pruning probabilit.	1/b	1/b	1/b=1/2	n/a	n/a	1/b+37%[†]
Routing efficienc.	1	1	1	1	0	1
Node degree	$O(b^* \log_b N)$	$O[(b-1)^* \log_b N]$	$O(\log_2 N)$	O(d)	$O(\log_b N)$	$O[(b-1)\log_b N]$
Wire length/stretch	(≈ 2-4)	(≈1.3-1.4)	n/a	(≈ 2-3)	n/a	(≈ 2-4[‡])
Key/replica redundancy	k salt values	k closest nodes	k succ. nodes	k hash functions	n/a	k salt values[‡]
Link redundancy	tripl. table entries	r closest neighbors	r succ. nodes	r realities	n/a	triple table entries[‡]
Maintenance complexity	$O(\log_b N)$	$3b^*\log_b N$	$O(\log_2 N)$	$O(N^{1/d})$	$O(\log_b N)$	$O(\log_b N)$[‡]
Fairness measures	none	none	none	none	none	virtual nodes
Symmetry / heterogeneity	symm.	symm.	symm.	symm./ bin split	symm.	symm./ GoI

AGILE creates a topology where each node can be the root of a tree. It exhibits similar network diameter and node degree characteristics as Tapestry. It adopts the advantages of Tapestry in terms of fault tolerance and wire length as well as its overlay maintenance scheme. However, AGILE considerably improves the pruning probability by applying an adaptive algorithm that brings requestors and requested keys stochastically closer together. Furthermore, it introduces fairness into the lookup mechanism by imposing the highest routing burden on those peers making the most frequent requests.

[†] For large b, otherwise 37%/(1-1/b); for assumptions, cf. Section 4.5

[‡] Tapestry mechanism adopted

4. THE AGILE ALGORITHM

The AGILE algorithm proposed has been derived from the requirements presented above and combines the advantages of a scalable, hashing-based algorithm and topology with the efficiency and fairness of an interest- and usage-based group topology. The basic algorithm of the lookup inseparably combines the overlay topology and the lookup request routing.

For the subsequent discussions, consider the following scenario, where a peer node (the requestor) tries to find a certain service or content in the P2P network. It has to specify what it is looking for and the P2P system should return the content or service or a link to the content or service, *e.g.*, the IP address of a peer where it can be found. The desired and returned object is termed a lookup key (or simply key) and the specified request a lookup identifier (ID). Peer nodes in the network are characterized by their node ID, the node holding the lookup key is called provider node. Routing is the process of finding a path from the requestor to the provider node (which is usually unknown to the requestor) in a distributed way by forwarding lookup requests from one peer to another. The overlay network defines the structure on which request routing can take place.

4.1 ID Space and Arrangement of Nodes and Keys

A proper assignment of IDs to nodes and keys can be derived from the routing efficiency requirement. In order to avoid any kind of flooding and achieve a routing efficiency of 1, the P2P system is required to have global knowledge on the translation of search request or lookup key into lookup ID and on the association of the lookup ID with the provider ID. The use of hash functions, *e.g.*, based on SHA-1 [5] or MD-5 [12], to translate the search request, *e.g.*, the file name, into the lookup ID solves the first problem. The second problem is solved by arranging peer nodes in the same identifier space as the lookup IDs, *e.g.*, by applying the same hash function to nodes' IP addresses. The node with an ID numerically closest to the lookup ID will be the provider peer.

Figure 1 illustrates the identifier space in AGILE with peer nodes and lookup keys arranged in the same space. Note that due to the pseudo-randomness of the hash function distances of peers and the number of keys associated to a provider can vary. Stochastically, however, their distribution will be uniform. Figure 1 also introduces a hierarchy of types and genres in the identifier space. This hierarchy is derived from the requirement to achieve a good pruning factor. Assuming that request routing takes place along the identifier space (which, even though not linearly, is the case for AGILE), a good pruning factor requires that providers (or lookup keys, respectively) and potential requestors be located close to each other. AGILE achieves this through a clustering of keys and nodes into Groups of Interest (GoIs).

For a detailed illustration, assume a segmentation of lookup keys (content or services) as described by the following meta-information:

- Type, *e.g.*, music files, news information, or storage services.

– Genre, *e.g.*, rock, pop, classic, or house.
– Name, *e.g.*, RollingStones_Satisfaction or Beethoven_9.

Note that the specifics of the segmentation are purely illustrative and not focus of this work. One could as well apply a two-level hierarchy only, or subdivide the music genre further into different styles as done at allmusic.com or iuma.com, or use even higher level genre hierarchies [8].

Peer nodes have to be arranged in the same segmentation as content keys; in the illustration: type, genre, name, or, for nodes, IP address. The type and genre of a peer refer to its pre-eminent interests (its GoI). Section 4.4 below discusses how to determine the GoI of a peer and how to handle multiple interests. Hashing is then applied to each of the hierarchy levels. The lookup ID becomes TypeID.GenreID.NameID while the node ID will be TypeID.GenreID.AddressID.

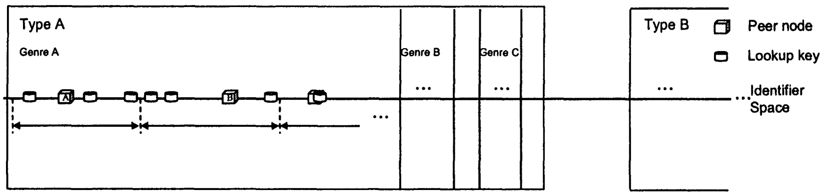


Figure 1. Identifier Space in AGILE

A total identifier space of 128 bit will be sufficient for most P2P systems. A distribution of bits to type, genre, and name/address, respectively, depends on the expected number of different types, different genres within a type and names/addresses within a type and genre. It is assumed that 32 bit each for type and genre and 64 bit for name/address will meet most demands.

4.2 Overlay Network Structure and Request Routing

Within the identifier space defined above, lookup requests have to be routed towards a node with the corresponding ID. It would be possible to route a request directly from one node to an adjacent one in the ID space in the direction of the lookup ID, who forwards it to its neighbor and so on until it finally reaches the provider. As this is highly inefficient and not scalable nor robust, an overlay network of virtual links needs to be constructed according to the requirements in Section 2, enabling every peer to route a request to any other peer in the identifier space with as few hops as possible.

A tree topology yields a good trade-off between node degree and network diameter. The tree is an efficient structure for searching or lookup, and both the node degree as well as the diameter scale logarithmically. For symmetry reasons and also to increase the bisection width of the graph, however, the simple tree structure needs to be extended: every peer has to be allowed to become the root of the tree or be on any other level, rather than maintaining links only to one level in the tree hierarchy.

Figure 2 (left) shows an AGILE overlay lookup tree. The lookup key segmentation defines the high-level tree hierarchy. As a root node, each peer maintains links to peers from all different types. Within its own type, each peer maintains links to peers from all different genres. Within its own type and genre, each peer maintains links to all peers. This enables an efficient hierarchical lookup request routing from the more generic type to the more specific genre and, eventually, name.

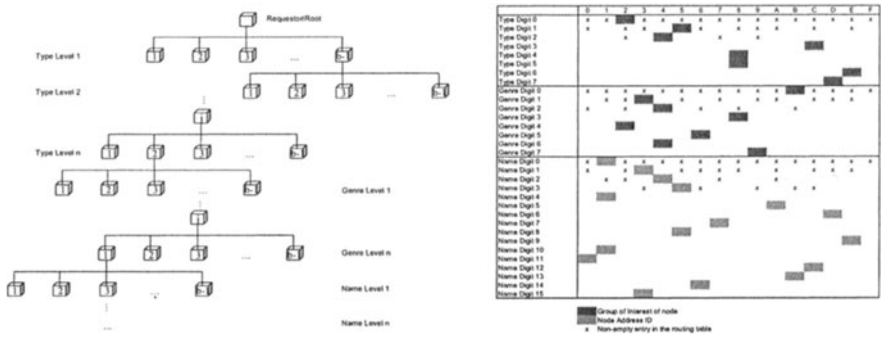


Figure 2. Left: An AGILE Overlay Lookup Tree; Right: Illustrative Routing Table

As the number of nodes in a genre or type can potentially become very large, a subordinate hierarchy is introduced to reduce the node degree, with a maximum of b nodes on each tree level. It is straightforward to associate b with the base of a numerical representation of the node or lookup ID. The position of a node (or key) in the tree is then determined by the succession of digits of its ID.

The resulting overlay network graph is defined through the virtual links on each peer, i.e. the routing tables. Figure 2 (right) illustrates a peer node routing table for a base b=16. The first row corresponds to the node being the root in a lookup tree. It has each one entry for peers with a different first digit in their ID. The second row holds entries for a lookup tree where the peer node is on the second level pointing to peers with identical first but different second digits. In general, the i-th row in the table points to peer nodes who have (i-1) digits in common with the peer in consideration and span the entire value space (b values) for the i-th digit, if all such nodes exist in the system.

Once the overlay topology is created, it is important to define how lookup requests can be routed from the requestor to the provider. This becomes very straightforward and efficient in the AGILE structure. Figure 3 illustrates the approach. At each hop, the routing peer forwards the request to a peer such as to match one more digit of the node ID, starting at the first digit. To simplify the illustration, Figure 3 only represents the first three digits.

For example, consider a peer requesting a key with an example ID 12345678.12345678.1234567890ABCDEF. The requesting peer looks into the first row of its routing table for a peer with "1" as a first digit and sends the request. The contacted peer looks into the second row of its routing table and forwards the request to a peer with "2" in the second digit, while the routing entries in the second row automatically ensure that the first digit of all entries is "1". The process continues until the type ID is matched or the search is stopped. The same mechanism runs for the genre ID. Finally, for the name ID, the process stops, when it reaches a peer with an empty corresponding row in the routing table. This peer holds the key, if it exists, or returns an error message. It is obvious that a requestor directly starts with the search for the name ID, if it itself belongs to the corresponding GoI. Similarly, a request may progress several digits at a time if the lookup ID matches more than one further digit with the processing peer. The pseudo-code for AGILE routing can be found in [7].

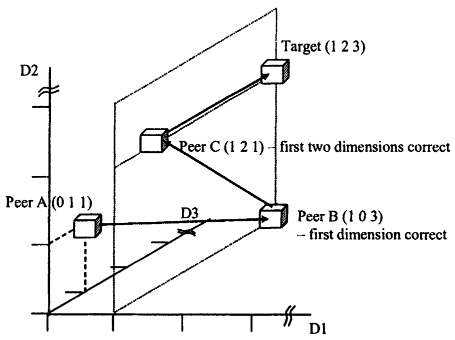

Target (1 2 3)
D2
Peer C (1 2 1) - first two dimensions correct
Peer A (0 1 1)
D3
Peer B (1 0 3)
– first dimension correct
D1

Figure 3. Illustration of Topology and Routing in AGILE

4.3 Insertion and Removal of Keys and Nodes

In order for the mechanisms described in the previous paragraph to work, it is necessary to first insert keys into the system and onto the node with the numerically closest ID. Furthermore, the topology (*i.e.*, the routing tables) have to be maintained as peers join and leave the network.

The insertion of keys into the system works exactly reciprocal to the lookup of a key. The peer node wishing to offer new content or services initiates an insert request with the according lookup ID. The request is routed just in the same way as a lookup request until it reaches the designated provider peer node which stores the key. For the removal of a key, the peer that stops to offer certain content or services sends a removal request with the according lookup ID into the network. The provider peer deletes the key.

The insertion of nodes into the system also works along the routing path. The new node contacts any known node. A node insert request is routed according to the usual routing procedure with the joining node's ID as lookup ID. At each hop in the path, the existing node learns about the new node. The joining node, in turn, can copy a row (row i at the i-th hop) from the forwarding node's routing table to initialize its own routing table. The insertion of nodes becomes more intricate once one wants to optimize wire length and achieve proximity in the underlying network for all or most nodes in the routing table. We have adopted the Tapestry [16] and Brocade [17] mechanisms including the algorithms for node removal, redundancy creation and fault management and the replication strategy.

4.4 Group Management and Adaptiveness

Groups of Interest (GoI) have been introduced to achieve a good pruning probability or "tunneling", since the first hops are avoided through GoIs. The goal of adaptive GoI management is to establish a process for peers joining and leaving GoIs such as to improve pruning or tunneling while keeping the overhead for group management itself reasonable.

A peer first joins a GoI by explicitly choosing categories of interest during the installation phase. Afterwards, requests for content will automatically make it join the requested GoI. That means, a peer can join more than one GoI. For each GoI, it carries a different node ID, derived from its GoI and IP address as discussed before. When joining a GoI and creating a new node ID, the peer effectively creates a new virtual node. It has to maintain a complete routing table for the virtual node that corresponds to its ID. The insertion takes place just as for a real node.

Two mechanisms help keep the overhead incurred by introducing virtual nodes and catering for more than one ID on a single node minimal: *thresholding* and *time filtering*. Thresholding means that a node only joins a new GoI, if the number of requests to that GoI exceeds a certain value. Time filtering means that the accounting of requests towards the threshold will be attenuated over time. Effectively, a node will leave a GoI, if it no longer makes requests to that group over a period of time - the corresponding virtual node is removed. Initially, time filtering will be a simple windowing; subsequent improvements are possible using adaptive filtering to predict future request behavior. It is assumed that the observation of a peer's past behavior leads to reasonable predictions as the change rate of likes and dislikes will be slow compared to the request rate. In addition to thresholding and time filtering, a third mechanism, *aggregation*, may be required in designs choosing a higher-level hierarchy than the three level type-genre-name example, where requests to presumably very small leaf-GoIs occur only infrequently. Requests not only to leaf-GoIs in the hierarchy will be counted, but all requests within a higher- (up to second-) level hierarchy will be aggregated. Once the threshold for the aggregated requests is exceeded, a virtual node will be created at the leaf-GoI most requests have been made to.

Through the introduction of GoIs, their automated update, and the consequent introduction of virtual nodes, AGILE makes the lookup topology adaptive. Nodes eventually move toward the content they like and request.

The pseudo-randomness of the hash function in AGILE ensures load balancing, as nodes as well as content items are spread uniformly over the key space with respect to their type, genre, and name. However, GoIs in AGILE allow hot spots in the key space to form. If many nodes share a popular common interest, the key space will become far more populated in the respective type/genre area than in the areas corresponding to less popular interests. This, however, is a natural process. As the GoIs of these nodes coincide with their requests, the degree of node agglomeration is proportional to the degree of request agglomeration. Proper load balancing in the system is ensured.

Peers that have joined several GoIs, however, do have to carry a significantly higher routing load than others. This meets the system's fairness requirement. Peers requesting many content items from many GoIs and, thus, consuming many network resources also have an increased routing burden themselves. Peers making very infrequent requests to GoIs are not affected as the time filtering and thresholding makes them eventually leave the GoI in concern, releasing the additional routing burden.

Peers with frequent requests to the same GoI also carry a higher routing load in AGILE. New virtual nodes within the same GoI are automatically created when the number of requests per time interval exceeds a certain threshold. The pseudo-code for GoI-management can be found in [7].

4.5 Evaluation

A detailed evaluation of the node degree and the average number of hops for a lookup request is given here to show the impact of adaptive, group-of-interest-based overlay management on performance.

For the node degree, the routing table is considered. The routing table is densely populated in the first rows for type, genre, and name/node ID, depending on the number of nodes. As GoIs are spread uniformly across type ID and genre ID, respectively, it is unlikely that one GoI will have many identical digits with another GoI - the table becomes very sparse in the bottom rows. The same holds true for the name ID. More precisely, the probability that entry j in row i of the type, genre, or name area is populated is,

$$p_{i,j} = 1 - (1 - b^{-i})^{N_{t,g,n} - 1}$$

where $N_{t,g,n}$ denotes the number of different types, the number of different genres within a type, or the number of nodes within a GoI, respectively. Note that the counting of rows starts from 0 for each of the areas type, genre, and name. This yields for the total population of the table, the node degree d

$$d = (1 + n_v)(d_t + d_g + d_n); \quad d_{t,g,n} = (b-1) \sum_{i=1}^{R_{t,g,n}} \left[1 - (1 - b^{-i})^{N_{t,g,n}-1}\right]$$

where n_v is the number of virtual nodes and $R_{t,g,n}$ is the number of rows for type ID, genre ID, and name ID, respectively. The node degree is plotted in Figure 4 (left) for a base b=16, R_t=R_g=8, R_n=16, n_v=0. Two curves show the node degree for 50, and 5 different types and different genres within a type, respectively. Except for very low number of nodes, both curves lie well below the logarithmic curve $(b-1)\log_b N$ as well as below the reference curve without grouping (R_t=R_g=0, R_n=32), which can be regarded as an approximation for algorithms without grouping like Tapestry and Pastry.

The average number of hops for a lookup request n_h is approximated as follows:

$$n_h = (1 - p_{GoI,t})\, n_{h,t} + p_{success,t}\,(1 - p_{GoI,t,g})\, n_{h,g} + p_{success,t,g}\, n_{h,n} + p_{success,t,g} \cdot 1$$

where $n_{h,t}$, $n_{h,g}$, $n_{h,n}$ are the number of hops needed to match the type, genre, and name of the lookup key, respectively, if the lookup key exists, but does not fall within the requestor's group of interest. $p_{GoI,t}$ and $p_{GoI,t,g}$ denote the probabilities that the lookup refers to the requestor's group of interest type or genre, respectively.

$p_{success,t}$ and $p_{success,t,g}$ define the probabilities that the request is successful with respect to the type and genre.

The additional hop is an approximation for the hops that occur when the next digit cannot be matched, but when nevertheless closer nodes are available in the routing table. As type, genre, and address IDs are uniformly distributed, it is unlikely that more than one such hop occurs.

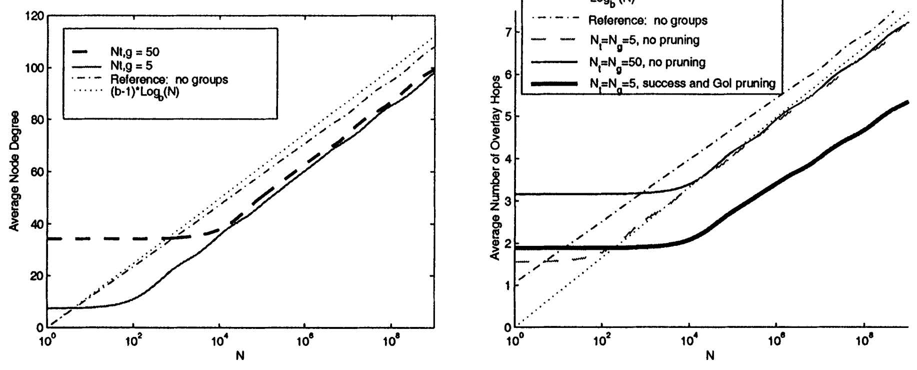


Figure 4. Left: Node degree; Right: Number of hops

Based on the likelihood that a node exists sharing i digits with the lookup ID, $p_{exist,i}=1-(1-b^{-i})^{N_{t,g,n}}$, it is:

$$n_{h,t,g,n} = \sum_{i=1}^{R_{t,g,n}} i(p_{exist,i} - p_{exist,i+1})TF_i$$

where $p_{exist,R+1}$ is defined to be zero. As some of the hops from one row to the next one happen on one and the same node and do not represent actual hops on the overlay network, the tunneling factor TF_i is introduced. It represents the ratio of hops on the overlay network to advances in routing table rows up to row i and can be derived to be

$$TF_i = \frac{b^{-i}}{i}\sum_{k=1}^{i}\binom{i}{k}k(b-1)^k = (1-b^{-1})$$

The additional pruning factor achieved through the introduction of GoIs becomes

$$f_{p,GoI} = 1 - \frac{n_h}{n_{h,t} + n_{h,g} + n_{h,n} + 1}$$

The average number of hops is plotted in Figure 4 (right) for b=16, R_t=R_g=8, R_n=16 as for the node degree. As before, for comparison, $\log_b N$ and a reference curve without grouping (R_t=R_g=0, R_n=32), which should show similar results as algorithms like Tapestry or Pastry are also shown. Two curves for 50 and 5 different types and different genres within a type, respectively, show the expected number of hops, when there is no pruning due to a node requesting a lookup key within its own GoI or due to quick abortion of the lookup when the type or genre is not available ($p_{success}$=100%, p_{GoI}=0%). For a reasonably large number of nodes, both curves are close to $\log_b N$ and exhibit a slight gain compared to the reference case. The last curve represents the average number of hops for 50 different types and genres within a type when there is pruning; the scenario assumes $p_{GoI,t}$=70%, $p_{GoI,t,g}$=30%, $p_{success,t}$=95%, and $p_{success,t,g}$=85%. In this case, the pruning factor becomes 37% compared to the reference case when there are 1 million nodes in the network.

Additional pruning gains can be achieved using a higher-level hierarchy for GoI structuring. Effects on the node degree will be limited and beneficial as long as no additional virtual nodes are created; this, however, is highly dependent on the thresholding and aggregation parameters applied.

The effects of an increase in amount and type of information in the system can be studied by comparing the graphs for $N_{t,g}$=5 and $N_{t,g}$=50 in Figure 4: the effect is limited and mostly visible in small networks only.

5. CONCLUSIONS AND FUTURE WORK

AGILE is a new lookup and routing algorithm bringing requestors and providers close together in Groups-of-Interest (GoI), tackling important scalability and performance concerns about the overlay network management for lookup and routing, while at the same time promoting system fairness. It can be applied to all peer-to-peer applications requiring such lookup services, as diverse as file sharing, distributed search and indexing, and, with some adaptations, distributed storage or file systems and distributed computing.

In future versions of the algorithm, searches with regular search expressions will be investigated. As a first step, search within a GoI can be replaced by controlled flooding rather than hash-based routing. Subsequently, a globally known semantic closeness operation will be needed to replace the hashing scheme, combined with proper load balancing, as the pseudo-random uniform load distribution due to hashing will be lost.

ACKNOWLEDGEMENTS

This work has been performed partially in the framework of the EU IST project MMAPPS 'Market Management of Peer-to-Peer Services' (IST-2001-34201), where the ETH Zürich has been funded by the Swiss Bundesministerium für Bildung und Wissenschaft BBW, Bern under Grant No. 00.0275.

REFERENCES

[1] H. Balakrishnan, M. Kaashoek, D. Karger, R. Morris, I. Stoica: *Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications*; ACM SIGCOMM, San Diego, August 27-31, 2001.

[2] E.A. Brewer: *Lessons from Giant-Scale Services*; IEEE Internet Computing Vol. 5 Nr. 4, pp. 46-55, July/August 2001.

[3] M. Castro, P. Druschel, Y. C. Hu and A. Rowstron: *Exploiting network proximity in peer-to-peer overlay networks*; International Workshop on Future Directions in Distributed Computing (FuDiCo), Bertinoro, Italy, June 2002.

[4] Druschel, Rowstron: *Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems*; IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, 2001.

[5] FIPS 180-1, *Secure Hash Standard*; U.S. Department of Commerce/NIST, National Technical Information Service, Springfield, VA, April 1995.

[6] K. Hwang: *Advanced Computer Architecture;* McGraw-Hill Series in Computer Science, p.77, 1993.

[7] J. Mischke, B. Stiller: *Peer-to-peer Overlay Network Management Through AGILE: Adaptive, Group-of-Interest Based Lookup Engine*; Extended Version, ETH Zürich, Switzerland, TIK-Report No. 149, August 2002.

[8] F. Pachet, D. Cazaly: *A Classification of Musical Genre*; Proceedings of Content-Based Multimedia Information Access (RIAO) Conference, Paris, France, 2000.

[9] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker: *A Scalable Content-Addressable Network*; ACM SIGCOMM '01, San Diego, 2001.

[10] S. Ratnasamy, M. Handley, R. Karp, S. Shenker: *Topologically-Aware Overlay Construction and Server Selection*; IEEE INFOCOM, New York, June 2002.

[11] S. Rhea, J. Kubiatowicz: *Probabilistic Location and Routing*; IEEE INFOCOM, New York, June 2002.

[12] R. Rivest: *The MD-5 Message Digest Algorithm*; RFC 1321, 1992, http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc1321.html in August 2002.

[13] S. Saroiu, P. Gummadi, S. Gribble: *A Measurement Study of Peer-to-peer File Sharing Systems*; Technical Report # UW-CSE-01-06-02, Department of Computer Science & Engineering, University of Washington, Seattle, 2002.

[14] M. Schlosser, M. Sintek, S. Decker, W. Nejdl: *HyperCuP - Hypercubes, Ontologies and Efficient Search on P2P Networks*; International Workshop on Agents and Peer-to-Peer Computing (AP2PC), Bologna, Italy, July 2002.

[15] K. Sripanidkulchai, B. Maggs, H. Zhang: *Enabling Efficient Content Location and Retrieval in Peer-to-Peer Systems by Exploiting Locality in Interests*; ACM SIGCOMM, Computer Communication Review Vol. 30 Nr. 1, January 2002, p. 80.

[16] B. Zhao, J. Kubiatowicz, A. Joseph: *Tapestry: An infrastructure for fault-tolerant wide-area location and routing*; Technical Report UCB/CSD-01-1141, Computer Science Division, U.C. Berkeley, April 2001.

[17] B. Zhao, Y. Duan, L. Huang, A. Joseph, J. Kubiatowicz: *Brocade: Landmark Routing on Overlay Networks*; First International Workshop on Peer-to-Peer Systems (IPTPS), Cambridge, MA, March 2002.

[18] B. Zhao, A. Joseph, J. Kubiatowicz: *Locality Aware Mechanisms for Large-scale Networks*; International Workshop on Future Directions in Distributed Computing (FuDiCo), Bertinoro, Italy, June 2002.

WEB SERVICES MANAGEMENT NETWORK
An Overlay Network for Federated Service Management

Vijay Machiraju, Akhil Sahai, Aad van Moorsel
Hewlett-Packard Laboratories
1501 Page Mill Road, Palo Alto, CA 94034
{vijaym, asahai, aad}@hpl.hp.com

Abstract: We introduce the architecture, object model, components, and protocols of a management overlay for federated service management, called Web Services Management Network (WSMN). WSMN targets management of web services that interact across administrative domains, and therefore typically involves multiple stakeholders (examples are business-to-business, service provider interconnections, help desks). The architecture is based on (implicit) SLAs to formalize relations across domains. It relies on a network of communicating service intermediaries, each such intermediary being a proxy positioned between the service and the outside world. WSMN also exchanges control information to agree on what to monitor, where to monitor, and whom to provide visibility.

Key words: management, service management, SLA, web services, web service management network

1. INTRODUCTION

By packaging software applications as 'services' that are accessible over the Internet or intranet, enterprises achieve new and better means to utilize their own and each other's applications. Services[1] can be accessed through manual user activities (e.g., browser-based IT help desk), and increasingly through other services. In the latter case, conglomerations of interacting services emerge, which

[1] We use the terms 'service' and 'web service' interchangeably, but prefer the term 'service,' since it stresses that we discuss the management of applications exposed as *services*, instead of the fact that we assume these services to communicate through *web services* technology (SOAP, XML, WSDL).

access one another through more and more automated means. Examples can be found in business-to-business computing (web services, supply-chain processes, payment services), service providers (utility or grid computing) or in enterprise applications (payroll applications, remote IT services).

In the emerging world of Internet services, operational management becomes exceedingly important and challenging (and thus interesting). Services often directly impact the business process execution, and mishaps may directly be reflected in the bottom line of a business. This puts a premium on fault and performance management capabilities for services. Moreover, the services paradigm increases the complexity of run-time operations. Services communicate across monitoring domains [1], include different business partners, and are likely to rely on third parties to complete a service offering. As a consequence, service management has to deal with multi-party interactions, has to collect a large amount of data and synthesize it to understand the health of relationships between partners, and must resolve the limited end-to-end visibility and control one has over each other's services.

From the above we conclude that traditional application management must evolve into 'true' service management, and ultimately into service 'relationship' management. First, we must manage a service 'as a whole,' that is, as it is provided to a partner. Contrast this with application management, for which it is necessary to understand how the service is internally implemented through a set of objects and what exceptions each object generates. Instead, 'true' service management manages the interactions a service has with other services or consumers, for which we need visibility at the service interface between an application and its users. Secondly, we must manage relations, not only through local management, but also through sharing data between partners as needed, and, more importantly, through exchanging signaling information about monitoring, service levels and control actions. In other words, we need to be concerned with federated management [2].

In this paper, therefore, we propose Web Service Management Network, a management architecture for federated service management. Since we believe it is safe to assume Internet services will be implemented using web service technology (SOAP, XML, WSDL), WSMN is based on such technology. The critical concept in WSMN is that of SLAs. If SLAs are explicitly defined we make them manageable, and if no SLAs are actually agreed upon between services, WSMN manages towards SLAs imposed specifically for management purposes ('implicit SLAs'). The SLA concept allows us to frame and solve many problems rather elegantly and effectively, as we discuss further in Section 2.1, and illustrate using a WSMN prototype implementation in Section 4. WSMN, then, is (1) a network of cooperating intermediaries, each such intermediary implemented as a proxy sitting between a service and the outside world, and (2) a set of protocols to manage service relationships expressed through SLAs.

One can regard WSMN as a logical signaling network for management purposes, a concept well known from telecom (SS7 [3]). In that sense it is quite different from traditional management protocols such as those supporting SNMP and CIM based monitoring. It is closer to various overlays that are emerging throughout the various layers of the Internet stack, to establish quality guarantees that the Internet stack alone cannot create. Examples exist for instance for Internet telephony (SIP [4]) and streaming media content delivery [5]. Also of interest are the various emerging solutions to provide properties and features such as security, transactionality and change management to business-to-business web services. Examples are Flamenco Networks [6], Kenamea [7] and Talking Blocks [8]. All these companies use networks of collaborating intermediaries, often including a third-party play (repositories as well as services). However, none of these solutions addresses service management, as we do in this paper. Recently, Gartner surveyed and put in perspective these architectures, which they term web service networks [9]. Our term 'WSMN' is therefore extra appropriate, since our approach uses a web services network architecture for purposes of service management.

We believe that future web services management technology and standards must be based on WSMN as the architectural underpinning. The primary objective of this paper, therefore, is to introduce and explain WSMN and argue its value as core architecture for service and service relationship management. To this end, we introduce the main concepts behind WSMN in Section 2 (SLAs and protocols, respectively) and discuss the details of the intermediaries in Section 3. Section 4 then demonstrates multi-party SLA management using a WSMN prototype implementation.

2. WSMN DESIGN CHOICES

In one sentence, Web Services Management Network is a logical overlay network for SLA management between services, constituted of communicating intermediaries. Figure 1 illustrates WSMN, each intermediary being a proxy between a service (providing the interface to one or more applications) and the outside world. The most critical and interesting aspects of our design are (1) the choice to base all management on SLAs, and (2) the protocols for intermediaries to collaborate. We will argue in detail why we made these two design decisions, and discuss their consequences for the components in the intermediaries. We will not further elaborate why service management logic is placed in intermediaries, since we think that is an obvious enough choice—the reader can find a discussion in [9] and [10]. We note that we assume that services interact using web services technology, that is, XML, SOAP and WSDL [11].

2.1 SLAs as a Management Tool

SLAs are, of course, well-established in management [12], and one can argue that especially in service relationship management, SLAs will be of increasing and singular importance [10] [13]. However, our primary incentive to use SLAs is different: it is not that the existence of SLAs introduces a management problem we must deal with, but that voluntary introduction of SLAs can be a tool to manage service relationships. Hence, if SLAs are not sufficiently detailed, or are not

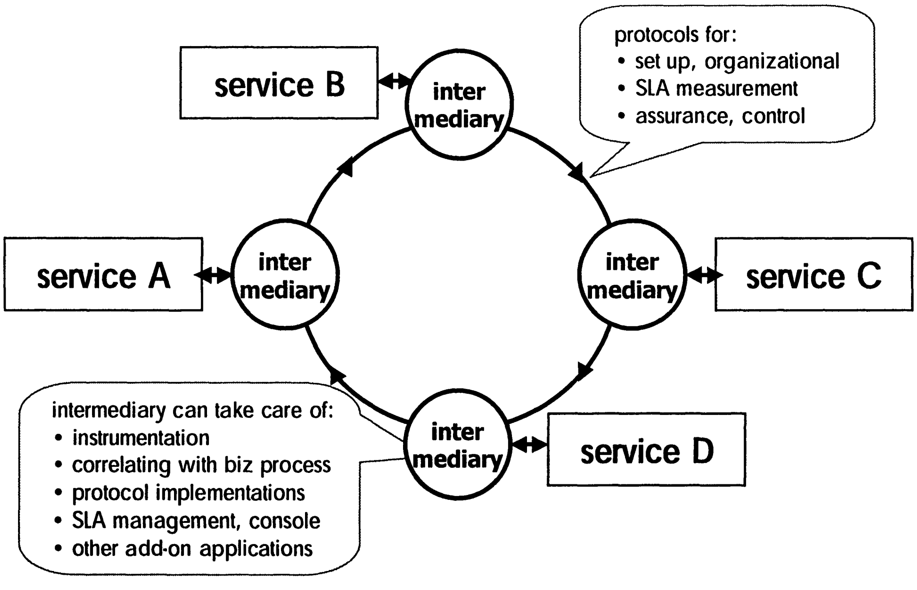


Figure 1. Web Services Management Network

explicitly agreed upon between services, we introduce SLAs for the purpose of management. We call these SLAs 'implicit SLAs.'

A service level agreement defines a basic abstraction through which partners can understand each other's capabilities, negotiate on service parameters, and manage to agreed levels. SLAs are a clean way to separate management concerns between the partners. In a situation where multiple partners interact with each other to accomplish a goal/task, an SLA is defined between every pair of interacting partners with well-defined expectations. This allows management of the overall task to be broken up up-front into smaller problems of managing each interaction. An alternate architecture is one that requires centralized intelligence, which makes sure that the overall task completes. However, such an architecture requires management data to be massively shared between all partners and is not scalable to a large number of partners. Moreover, in situations where each of the managed services runs within its own management domain, a centralized architecture will not be feasible.

Many problems in service relationship management can be solved through SLA management. For example, the problem of discovering and selecting right partners translates to the problem of querying for or negotiating an SLA. Similarly, the problem of managing a relationship translates to the problem of monitoring and assuring an SLA. Finally, solutions for rating partners and optimizing partnerships can be built by optimizing SLAs based on the cost of meeting them and penalties of not meeting them. Due to this close association, we use the words SLA management and relationship management interchangeably in the rest of this paper.

In WSMN, once an intermediary learns about an SLA (either input through a console or as the result of a semi-automated SLA negotiation protocol), it determines what elements to monitor, at what intermediary to monitor them, and how often to set alarms (see also Section 4). To establish such automated SLA management, the SLA must be specified unambiguously. In [14], we represent the SLA specification language we developed for that purpose. This specification language relies on a managed object model for web services presented in [15].

2.2 Protocols

The intermediaries interact amongst each other through a set of management protocols. These protocols range from basic ones that involve establishment and sustenance of the network to higher-level protocols that implement higher-level functionalities (partner selection, SLA assurance). We find it convenient to distinguish three classes of protocols: life-cycle protocols, measurement protocols and assurance protocols.

Life-Cycle Protocols. The life-cycle protocols are basic protocols that deal with initiation and sustenance of the WSMN. In our implementation, the initiation protocols are commenced as soon as two web services start communicating with each other. It is assumed that if the web service supports WSMN, the intermediaries are reachable at the address obtained by conjugating "/WSMN" at the end of the service URL. The intermediary that detects a web service communication initiates an establishment protocol that acquaints the intermediaries of each other's capabilities. After the establishment phase, the synchronization protocol is executed. A reasonably accurate clock synchronization protocol is needed because out of sync clocks could lead to erroneous results. Once a WSMN is established, unless it is torn down through an explicit teardown protocol, the intermediaries exchange keep-alive messages as part of the keep-alive protocol. Through all these protocols, the WSMN manages the various phases of its life cycle.

Measurement Protocols. We feel it is important to argue why service management requires measurement protocols between intermediaries, because it is not necessarily obvious such protocols are required. The key question is, if party *A* guarantees an SLA to party *B*, why is it not sufficient to monitor the compliance to this SLA at the intermediary of *A*?

There exist scenarios in which not all data required for managing SLAs can be measured locally. For example, consider a web service *A* that promises certain goods to be delivered to a customer *B* within five days. However, the web service uses a third service *C* for shipping the goods. As a result, *A* will not know when the goods were in fact delivered to *B*. The only way that *A* can measure its SLA is by consultation with *B* or *C*. We will see this example arise in the prototype discussion in Section 4. There, *A* monitors order arrival, *B* monitors delivery, and they exchange the results using the measurement exchange protocol. Another common example involves, say, a payment service provider *P* promising a certain perceived (end-user) transaction throughput to its customers. However, there are several service providers in the flow of execution between a customer and *P* – *P*'s data center *Q*, *Q*'s Internet service provider *R*, *R*'s carrier service provider *S*, and customer's Internet service provider *T* – all of which have an influence on the perceived throughput.

A common theme in all these examples is that the SLA dependency relations between web services do not exactly match the execution dependencies (in the first example above, *A* has an SLA dependency with *B*, but the execution dependencies also include *C*). An SLA can exist between two partners even though there are other players that influence the outcome of the SLA. Hence, SLA monitoring between web services requires an infrastructure where limited management information can be exchanged between partners in a trusted and secure manner. For this purpose, we developed a measurement exchange protocol, the details of which are described in Section 3.2.

Assurance Protocols. Assurance protocols are higher-level protocols executing more complex interactions, related to run-time optimization and control of SLAs. SLA negotiation requires negotiation protocols to be executed between the intermediaries. SLA assurance can be done by dynamically changing partners with different levels of service as suppliers. In another example, we developed a trouble-ticket exchange protocol, which forwards warnings between intermediaries as soon as SLAs are not met or are in danger of not being met. These trouble tickets are created, their status checked and are closed once the problem is satisfactorily fixed. Assurance protocols may very well depend on additional services, possibly offered through third parties—e.g., a UDDI repository helps discovering web services, third-party negotiation services help setting SLAs, rating services (which maintain records on web service performance) help identifying the right partners, etc. This paper does not focus on such assurance protocols.

3. MONITORING AT THE INTERMEDIARIES

Figure 2 illustrates the various components of the intermediary. The intermediary is embedded in the SOAP router, and has three main groups of components: (1) WSMN engines for measurement and SLA management, (2) WSMN protocol implementations, and (3) applications exploiting the engines and protocols. We discuss the first two items in detail, in particular with respect to instrumentation issues, and refer to Section 2 as well as the example in Section 4 for various applications that utilize WSMN.

3.1 Monitoring Engines

SOAP routers receive the messages from SOAP clients and submit them to the receivers (the *payload* layer in Figure 2). SOAP routers are the obvious candidate to support the intermediaries, since they already act as proxy for the web service interactions between a collection of services and the outside world. The WSMN intermediaries add management capabilities to SOAP routers, by capturing SOAP messages, potentially modifying the SOAP headers and extracting information from those messages. This can be done without any modifications to existing applications, and without re-compilation of the existing SOAP toolkit installation.

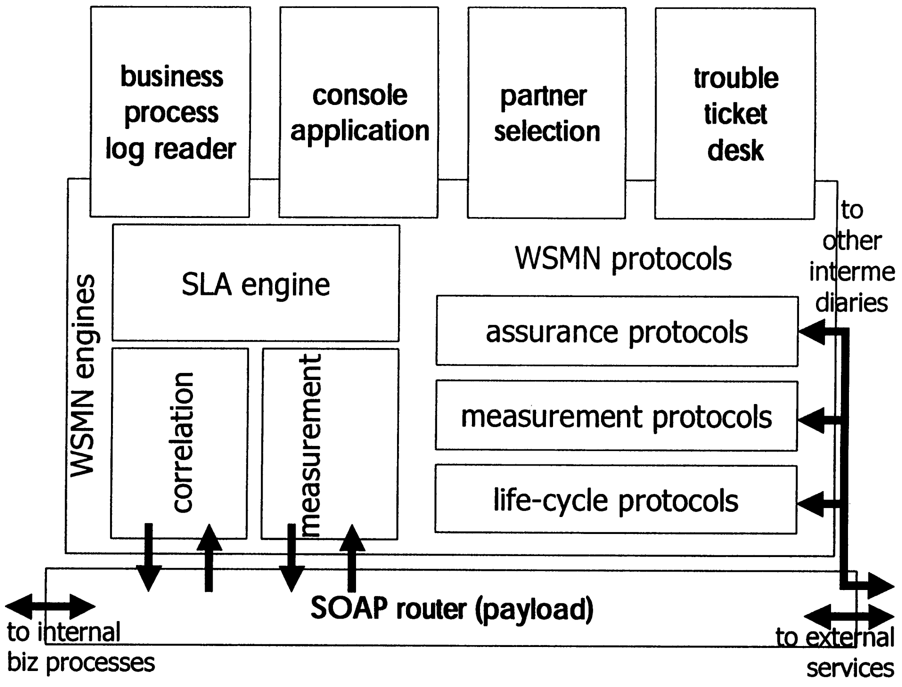


Figure 2. Components of WSMN intermediary

Figure 2 demonstrates two measurement engines that intercept the SOAP messages and manipulate them for measurement purposes: *measurement engine* and *business process correlation engine.* The measurement engine deals with measuring the interactions with the outside world, while the business process correlation engine deals with measuring the interactions with a process engine that maintains the state of conversations with partners (as specified through standards such as RosettaNet or ebXML [11]). The arrows between the *SOAP Router* and the *WSMN engines* signify that the engines intercept the SOAP message and detour the control flow.

Both measurement engines utilize a message tracking protocol, which allows one to correlate delays and other information over all segments a transaction traverses [16] [17]. A notion of global flow is introduced by this protocol. Messages in the same global flow use a unique ID (GUID) as identifier either as defined in some protocols (RosettaNet) or injected in SOAP headers by the measurement engine. The measurement engine checks every time it catches a message whether a GUID is present, and, if no GUID exists, it inserts a GUID into the SOAP header of the message. All SOAP routers propagate the GUID in their communications, and, consequently, all intermediaries are able to figure out which SOAP message is sent in the context of which previous messages. The details of the message tracking can be found in [16] and extensions to deal with the process engine are explained in [17].

In addition to the message data, one may very well be interested in gathering other information about the business, and correlate the activities with external message exchanges. To provide for that, one needs to add an application to the intermediary, such as the *business activity monitor* in Figure 2. For example, HP Process Manager logs execution data into a raw file, which can then be uploaded into database tables by the dedicated application. Alternatively, the add-on application uses the Java API provided by HP Process Manager. In combination with the GUID-based message tracking this provides a rich business activity monitoring solution [1].

3.2 Measurement Exchange Protocol

All WSMN protocols use web services transport, that is, the WSMN messages pass through the SOAP router just like messages with execution payload. However, as the arrows in Figure 2 indicate, WSMN messages do not 'merge' with the payload (as with interception methods), but are treated independently.[2] We assume that the necessary life-cycle protocols have executed as described in Section 2.2., to establish a well-functioning WSMN. As we argued in Section 2.2, it may then be required to combine the measurements of various intermediaries to determine if an SLA has been met. Therefore, we developed the measurement exchange protocol.

The measurement exchange protocol has been designed with the following objectives in mind: (a) minimize the amount of data that is transmitted between the two intermediaries, and (b) transfer the data in time for the evaluation of SLA to take place when triggered. Based on these two goals, WSMN intermediaries must agree on (a) what measurements need to be transferred and at what level of aggregation, and (b) how frequently must they be transferred. This is determined by the SLA specification. The details of the SLA specification are given in [14], but the important attributes are *evalFunc, evalWhen* and *measuredAt. measuredAt* specifies which service (and thus which intermediary) collects the data, and *evalWhen* specifies at what moments in time to collect the data. The attribute *evalFunc* allows us to be smart in aggregation of the measurements, using typical sampling functions such as *count (t)*, *totaled*, *averaged*, *movingAvg(lastN)*, *minN*, *maxN*, *threshold* (see [18] for possible strategies in data aggregation). In the case when the sampling function cannot be determined from the *evalFunc*, we transfer all the measurements from one side to the other.

The resulting measurement exchange protocol makes sure that there is agreement on the level of aggregation and the frequency of transferring data. This results in five different types of messages, which together form the protocol. We explain the primitives in terms of a scenario in which a 'provider' obtains data from a 'customer' (see Section 4). The primitives are:

[2] Note that the three protocol types (*life-cycle*, *measurement* and *assurance*) in Figure 2 are just a classification and do not form a stack in the sense of the ISO reference model. All WSMN protocols execute independently from each other, but have in common that they communicate using SOAP (that is, taking the stack perspective, all WSMN protocols sit one layer above SOAP).

- *Init*: sent by the customer to the provider for clauses whose measurement data need to be exchanged. The *init* message carries possible choices of sampling function, interval, duration and reporting interval details that the consumer supports.
- *Request*: The provider decides the exact measurement specification (sampling function, sampling parameters and reporting parameters) that it chooses and specifies it in its request message.
- *Agreement*: The customer sends this message if it agrees to the request
- *Start*: message from provider to commence the reporting
- *Report*: actual measurement report messages
- *Close*: message to terminate the reporting

4. APPLICATION OF WSMN

To demonstrate and test WSMN, we built a prototype WSMN intermediary, and created a test environment based on a business-to-business scenario. In this scenario, a PC vendor wants to manage SLAs with its customers and suppliers. Figure 3 shows the various players in this scenario: the vendor *PCMaker*, its supplier web services *ChipSupply*, *Assembly*, *Payment*, *Delivery* and a customer service, namely *PCBuyer1*, thus resulting in a WSMN with six intermediaries. We have no illusions that this scenario is particularly close to reality, but it serves our purposes of demonstration and testing. A typical message exchange sequence between the various players is shown in Figure 4. Effectively, the (potential) buyer logs in with the vendor and asks for a quote from the PC vendor. *PCMaker* first checks with one of its suppliers and then returns a quote to the buyer. In this case, the buyer decides to order, and the PC vendor executes the order through its providers. Note that no doubt a lot of manual work is involved at various stages, but that the interactions in Figure 3 and Figure 4 only refer to electronic messages. *PCMaker* agrees on an SLA with the buyer, stipulating that delivery will not take more than some number of days—the two parties agree in their SLA that this corresponds to the time from the moment that *PCMaker* receives order message 7 until *PCBuyer1* acknowledges delivery through message 16.

Executing life-cycle protocols to set up WSMN. The WSMN is created as soon as the service to service communication is detected by the intermediaries. For that, we implemented the life-cycle protocols mentioned in Section 2.2. We also implemented a third-party synchronizer service. This service is used by all the intermediaries to synchronize their clocks.

We added a console as an add-on application to the intermediaries (as denoted in the second application box in Figure 2). Later we see how we use that for visualizing various aspects of SLA management, but we also use it to show what services are running. In our prototype, a console is available for *PCMaker*. It depicts the services that are known to *PCMaker*, resulting in an up-to-date 'run-time version' of Figure 3. The latest interaction that the measurement engine intercepted is colored blue—as one can see, the latest communication at the moment of this snapshot was with the synchronizer service.

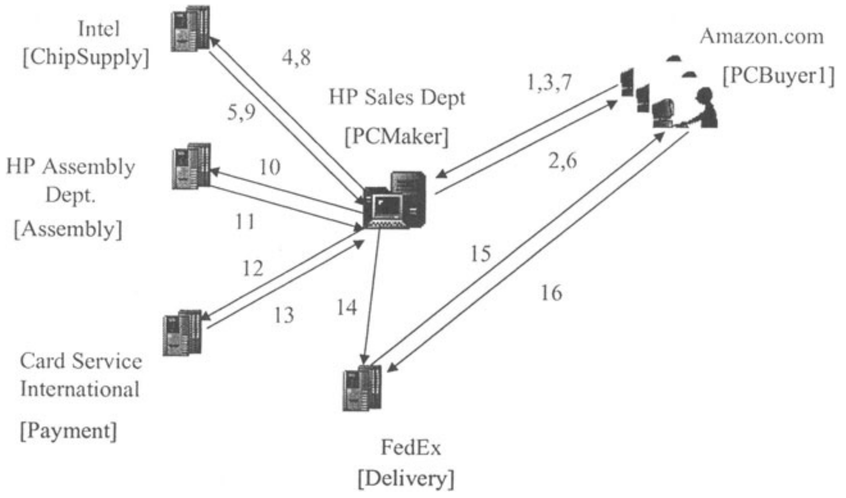


Figure 3. Example interactions

#	MSG_TYPE	SENDER	RECEIVER
1	SubmitLoginmsg	PCBuyer1	PCMaker
2	ConfirmLoginmsg	PCMaker	PCBuyer1
3	SubmitQuoteRequestmsg	PCBuyer1	PCMaker
4	RequestChipQuotemsg	PCMaker	ChipSupply
5	SendChipQuotemsg	ChipSupply	PCMaker
6	SendQuotemsg	PCMaker	PCBuyer1
7	SubmitPORequestmsg	PCBuyer1	PCMaker
8	SendChipPOmsg	PCMaker	ChipSupply
9	RespondChipPOmsg	ChipSupply	PCMaker
10	SendAssemblyPOmsg	PCMaker	Assembly
11	RespondAssemblyPOmsg	Assembly	PCMaker
12	SendPaymentPOmsg	PCMaker	Payment
13	RespondPaymentPOmsg	Payment	PCMaker
14	SendDeliveryPOmsg	PCMaker	Delivery
15	SendDeliveryNotificationmsg	Delivery	PCBuyer1
16	SendReceiptNotificationmsg	PCBuyer1	Delivery

Figure 4. Interactions for the example

```
<sla>
   <slaId>3</slaId>
   <startDate>Fri Feb 15 00:00:00 PST 2002</startDate>
   <endDate>Mon Jul 15 00:00:00 PDT 2002</endDate>
   <slo>
      <sloId>1</sloId>
      <dayTimeConstraint>Mon-Fri: 9-17</dayTimeConstraint>
      <measuredItem>
         <item>
            <constructType>message</constructType>
            <constructRef>PCMaker.com/SubmitPORequestmsg</constructRef>
            <measuredAt>PCMaker.com</measuredAt>
         </item>
         <item>
            <constructType>message</constructType>
            <constructRef>
                PCBuyer1.com/SendReceiptNotificationmsg
            </constructRef>
            <measuredAt>PCBuyer1.com</measuredAt>
         </item>
      </measuredItem>
      <evalWhen>6PM</evalWhen>
      <evalOn>all</evalOn>
      <evalFunc name ="averageResponseTime"
                operator ="LT" Threshold ="5" unit ="days"/>
   </slo>
</sla>
```

Figure 5. XML specification of an SLA

Setting up an SLA. In order to demonstrate SLA management in the overlay network we defined an SLA between *PCMaker* and *PCBuyer1*. Figure 5 provides the details of one agreed upon SLA, in XML format: over the specified period, the average time from when *PCMaker* receives the order until *PCBuyer1* acknowledges its delivery, must be less than five days (this is the time between message 7 and 16 in Figure 4). The SLO (service level objective) in the agreement requires measurement at each end-point. The WSMN intermediaries at *PCBuyer1* and *PCMaker* utilize the measurement exchange protocol to agree on sending measurements for *SendReceiptNotificationmsg* everyday just before 6 PM and keep sending the reports from *startDate* to *endDate*. In our prototype, the console allows one to load the SLA in the intermediary at both at *PCBuyer1* and *PCMaker*—once loaded, the intermediary immediately starts the processes necessary for SLA management.

Implementation of the intermediary. All services in our prototype have been implemented using Apache SOAP toolkit, and we extended the SOAP toolkit to collect the message correlation and instrumentation data. We use WSFL (Web Services Flow Language [11]) as flow language, use HP Process Manager (HPPM) for orchestration of conversations between services. Since HPPM provides a Java API to control process executions by other software components, the business process correlation engine can use this API to feed a unique ID into HPPM process instances and retrieve it when necessary. The measured data is all stored and modeled in a mySql database for short-term storage and in an Oracle9i data warehouse for long term archiving. Figure 6 illustrates these and other components; notice that Figure 6 is a more detailed version of Figure 2.

The *model generator* in the intermediary receives the WSDL/WSFL specifications and creates a model of the web service in the *model repository*. All the measurements collected from the service (e.g., ongoing conversations, performance measurements, etc) are attached to this model. The instrumentation in the web service is responsible for collecting these measurements and passing them on to the *measurement handler* to be stored in the model repository. If the measurements are collected on the client side (as determined by the *measuredAt* components of the items in SLA clauses), then the *communicator* is responsible for receiving the measurements and storing them into the repository. If it is required that management data is transferred between intermediaries, then the *management protocol handler* executes the measurement exchange protocol (see Section 3.2).

SLA engine. As soon as the SLA engine *management process controller* receives the SLA it executes a *monitoring process flow* and accordingly informs the *SLA customizer*, which in turn customizes the alarms at the *Event Manager* (depending on the *evalWhen* and *dateconstraint* components). The *Event Manager* comprises of the *SLO Validity Period Monitor* and triggers (time based and event based). The *SLA customizer* also creates an *SLO object* in the *SLA repository* and registers it as the call back handler of the events. The *SLO object* maintains the state of the SLO (*valid*, *active*, *invalid*). If a registered event for start-date of an SLO arrives the state of the SLO is changed from *init* to *valid*. The SLO is invalidated when the *end-date trigger* arrives. While the *evalWhen* events are triggered (because of a time or an event happening) the *SLO evaluator* evaluates the SLO. The *SLO evaluator* obtains the required management information (based on *evalOn*, *daytime constraint* and the *evalFunc* constituent of the specification) from the high performance database in memory. The *SLO evaluator* determines compliance/violations. The *SLA violation engine* maintains the record for violations, their timestamps, the levels of violation, and the clauses that are violated (both in memory and in log files). The violation records will also be used by the *SLA violation engine* for triggering actions specified by *evalAction* constituent of the SLO.

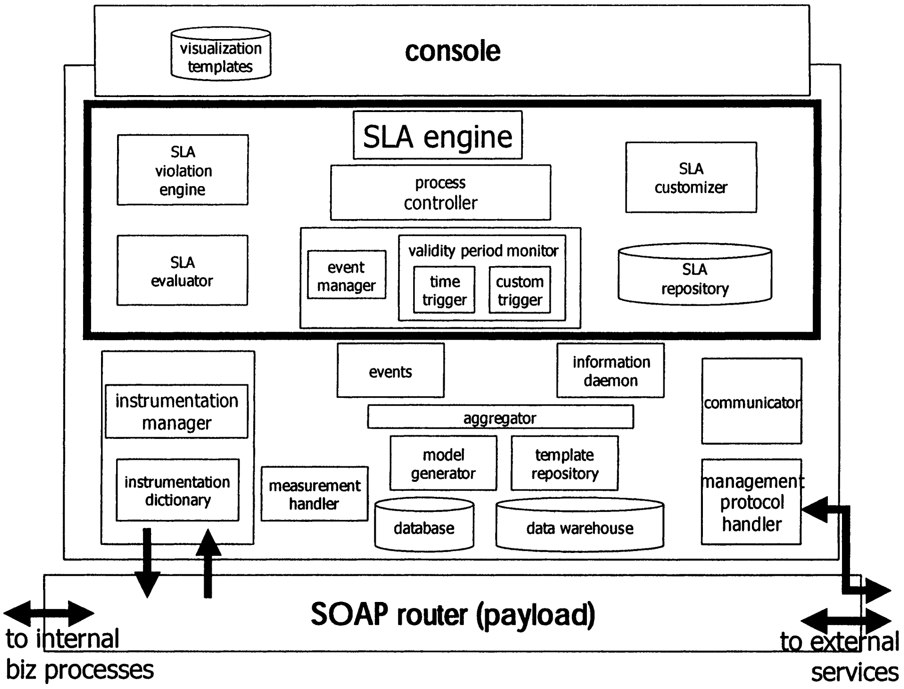


Figure 6. Components of the WSMN intermediary (detailed version)

5. CONCLUSION

WSMN provides the appropriate architectural underpinnings for future development in web service management technology and standards. WSMN is a logical overlay network constituted of communicating intermediaries, each such intermediary implemented as a proxy sitting between a service and the outside world. It assumes a service-centric model for application usage, and focuses on managing the service offering (as opposed to the internals of applications). Moreover, it realizes that service management will be more and more about managing the overall quality of interactions (service relationship management), and uses the concept of SLAs to provide a flexible and scalable management solution for such management.

'Implicit SLAs' are introduced, to allow for management of SLAs that have not explicitly been agreed upon or of SLAs that do not specify enough monitoring details. WSMN also provides a set of protocols necessary to deal with cases in which multiple parties must share management data to evaluate if SLAs are successfully met. Future research must deal with operational aspects of WSMN, including issues of security and trust (along the lines of the work in [16] for message tracking). The existing prototype implementation of WSMN allowed us to demonstrate the workings of the overlay network, and will help to further test the appropriateness of our current and future design decisions.

REFERENCES

[1] G. Kar, A. Keller, S. Calo, "Managing Application Services over Service Provider Networks: Architecture and Dependency Analysis," *Proceedings of 7th IEEE/IFIP Network Operations and Management Symposium (NOMS),* Honolulu, Hawaii, USA, April 2000.

[2] P. Bhoj, S. Singhal, S. Chutani, "SLA Management in Federated Environment," *Computer Networks*, Vol. 35, No. 1, pp. 5—24, 2001. Also *HP Labs Technical Report* HPL-1998-203, 1998.

[3] U. Black, *ISDN and SS7 Architecture for Digital Signaling Networks,* Upper Saddle River, New Jersey, USA: Prentice Hall PTR, 1997.

[4] H. Sinnreich, A. B. Johnston, *Internet Communications Using SIP*, John Wiley & Sons, 2001.

[5] T. Yoshimura, Y. Yonemoto, T. Ohya, M. Etoh, and S. Wee, "Mobile Streaming Media CDN Enabled by Dynamic SMIL," *Eleventh International World Wide Web Conference (WWW11)*, May 7-11, 2002, Honolulu, Hawaii, USA.

[6] Flamenco Networks URL: http://www.flamenconetworks.com

[7] Kenamea URL: http://www.kenamea.com

[8] Talking Blocks URL: http://www.talkingblocks.com/

[9] B. Lhereux, "Web Services Networks Secure a New Technology," *Gartner Research Note* SPA-17-0627, July 2002.

[10] A. van Moorsel, "Ten-Step Survival Guide for the Emerging Business Web," to be published in *Lecture Notes in Computer Science: Web Services, e-Business and the Semantic Web: Foundations, Models, Architectures, Engineering and Applications,* Springer-Verlag, 2002, also *HP Labs Technical Report* HPL-2002-203, July 2002.

[11] A. Sahai, S. Graupner, W. Kim, "The Unfolding of the Web Services Paradigm," to be published in *Internet Encyclopedia*, J. Wiley, also *HP Labs Technical Report* HPL-2002-130, May 2002.

[12] R. Sturm, W. Morris, M. Jander, *Foundations of Service Level Management*, Sams, 2000.

[13] A. Keller, G. Kar, H. Ludwig. A. Dan, J. Hellerstein, "Managing Dynamic Services: A Contract Based Approach to a Conceptual Architecture," *Proceedings IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pp. 513—528, Firenze, Italy, Apr. 2002.

[14] A. Sahai, A. Durante, V. Machiraju, "Towards Automated SLA Management," *HP Labs Technical Report*, HPL-2001-310, 2001.

[15] A. Sahai, V. Machiraju, "A Data Model Based on Service and Process Abstractions for Management of Systems," *HP Labs Technical Report*, HPL-2002-190, July 2002.

[16] A. Sahai, V. Machiraju, J. Ouyang, K. Wurster, "Message Tracking in SOAP-based Web Services," *Proceedings IEEE/IFIP Network Operations and Management Symposium (NOMS)*, Firenze, Italy, April 2002, also *HP Labs Technical Report*, HPL-2001-199, 2001.

[17] M. Sayal, V. Machiraju, A. Sahai, A. van Moorsel, "Correlators for Monitoring Web Services and Business Processes," to be published as *HP Labs Technical Report,* December 2002 (available from the authors).

[18] S. Frølund, M. Jain, J. Pruyne, "SoLOMon: Monitoring End-User Service Levels," *Integrated Network Management VI—Distributed Management for the Networked Millenium, Proceedings of the 6th IFIP/IEEE International Symposium on Integrated Network Management (IM),* M. Sloman, S. Mazumdar, E. Lupu (Editors), pp. 261—274, IEEE Computer Society Press, Boston, MA, USA, May 1999.

AUTO-DISCOVERY AT THE NETWORK AND SERVICE MANAGEMENT LAYER

Alexander Clemm, Anil Bansal
Cisco Systems, Inc.
170 West Tasman Drive, San Jose, CA 95134, USA
{alex, abansal}@cisco.com

Abstract: Auto-discovery capabilities of management systems typically pertain to network elements as a whole, i.e., the ability to automatically detect which network elements are connected to a network and to discover their type and physical and logical configuration. Service- and network-layer information, on the other hand, is in general not discovered but provisioned and provided by the organization operating the network and services. There are however scenarios in which the ability to auto-discover such information provides a lot of value. This paper describes the challenges that we encountered, the approach we took, and the lessons we learned in providing auto-discovery capabilities beyond the network element layer in the context of packet telephony and of metro Ethernet. We believe that these experiences will also be applicable to other contexts.

Key words: Network management, service management, auto-discovery, VoIP.

1. INTRODUCTION

A common function of element-layer management systems, as well as systems used for monitoring purposes, is auto-discovery. Auto-discovery is an overloaded term used in different contexts that can hence mean different things, but in general it refers to the ability of a management system to extract on its own certain information about what it is that needs to be managed, rather than requiring users to populate that information. When it comes to network and service layer management, however, information is rarely deduced from the network. Instead, the network and devices in it are considered service-agnostic – they support services, however information about services comes from the service provider. ("Service" refers here to the instantiation of a service for a given subscriber of that service, i.e., a service instance, not the service offering). The provider of a service uses service provisioning systems to drive the network configurations required to support the services into the network. Hence, any service information is assumed to be known a priori, not in need of being auto-discovered – the master of this information is the service provider, not the network. This service information is also used as the basis for aspects such as service level agreements (SLAs) or billing [8]. To verify that a service is provisioned correctly, it is always possible to check whether the current configuration in the network corresponds to the configuration it is supposed to have in support of the service, i.e., whether the network configuration "as built" corresponds to the network configuration "as planned".

In many cases, this is completely adequate. Nevertheless, there are situations in which it would be desirable to be able to discover network and service configurations from the network directly, rather than depending on service-related information from other sources. Reasons for this include:

- A service management and service provisioning system gets deployed at a later stage, after initial network deployment. A service provider would like to be able to see and automatically retrieve what services had earlier already been configured on the network.
- A service provider has maintained poor service records and reason to believe that its service records are not up to date. This scenario can occur specifically where service related configurations are not directly associated with specific end subscribers. An example would be a wireless service provider who needs to provide certain service capacity in a certain area, for instance a certain number of channels for GSM, TDMA, and data traffic for base station traffic.
- Operations personnel within a service provider's organization have gone around service provisioning systems and provisioned service instances "by hand", resulting in certain network-layer configuration mismatches that are hard to troubleshoot.

We have found such scenarios to be applicable for instance in the context of packet telephony management or management of metro Ethernet services, for which we hence encountered the requirement to include the ability to auto-discover network and service layer information as part of the management solution. In this paper, we will discuss the challenges that must be addressed when attempting to provide network and service layer auto-discovery, and an approach and design pattern that deals with those challenges. Our experiences are based on systems that we built for the management of packet telephony (Cisco Packet Telephony Center – PTC) and of metro Ethernets (Cisco IP Solution Center – ISC), and which incorporate the concepts described. However, no inferences should be made about Cisco product features or product direction. We expect our experiences to be not unique to those particular services but transferable to other service domains.

The remainder of this paper is structured as follows. What we mean by service layer auto-discovery and some background are discussed in Section 2. Section 3 dives into the challenges and considerations for service-layer auto-discovery. A design pattern that we have devised to tackle those challenges is subsequently introduced in Section 4. Two applications of this design pattern that have been realized in actual systems, one concerning packet telephony, the other metro Ethernet, are presented in Section 5. Finally, Section 6 offers some conclusions.

2. SERVICE LAYER AUTO-DISCOVERY

Auto-discovery is often encountered in the network element management context. It concerns the ability of a management system to automatically discover information about the network, specifically to discover what devices are in the network, what type of device they are, what their physical configuration is and how they have been logically configured. Frequently auto-discovery occurs by a management application pinging a range of IP addresses specified by the user. Upon receiving a response, the device's entity MIB is queried to identify device type and physical configuration. Subsequently, the device can be displayed on a topology map and is available for monitoring, MIB browsing and other management purposes. This way, auto-discovery obviates the need for management systems to be populated with this information by the user or obtain it through seed files or other systems. It allows the management system to obtain an accurate picture of the devices that are actually deployed in the network.

The situation is different as far as services are concerned. Management information about services is not discovered. Instead, the master of service management information is the service provider that provisions them respectively its operations support system, not the network. A service provisioning system is used to drive the required device configurations to support the services into the network, with the service provider keeping track of what services are provisioned. The provisioned services can be verified by checking whether the actual network configuration corresponds to how it was supposed to be provisioned, comparing whether the network configuration "as built" corresponds to the network configuration "as planned". (Please note that throughout this paper, we refer with service auto-discovery to the automatic discovery of management information that represents instances of services. This is not to be confused with auto-discovery of services themselves, e.g., through advertising of services by application servers in a network, e.g., [6, 9].)

The following are some examples of network and service layer concepts that would generally be provisioned but not auto-discovered by service providers:

- An H.323 zone in a VoIP (Voice over IP) network
- An instance of a residential ETTH (Ethernet To The Home) data service
- An instance of a transparent LAN service (TLS) in a metro Ethernet setting

Not having the capability to automatically discover network and service layer management information is completely adequate in many cases. However, as mentioned in the introduction, in practice scenarios can be encountered where service instances are not necessarily completely known and a capability to automatically discover them is desirable. For example, in packet telephony management, the scenario can occur where a network and service management system encounters an existing deployment when it is introduced. To be useful, this system needs to be populated with service-layer managed objects ("service MOs"), such as information on H.323 zones. Entering this information is a tedious and redundant exercise; after all, the network had already been provisioned earlier with those services. Likewise, even with a network and service management system in place, provisioning can sometimes occur working around it and configuring network elements directly (e.g., through the devices' command line interface - CLI). This leads to network-layer concepts and instances of services being introduced through the back door, without the management system knowing about them and without proper service MOs being created. This is a problem where network layer integrity might be violated without the management system knowing about it. Also, without service MOs being created, there is no way for an Operations Support System (OSS) to subsequently refer to the network and service layer concepts that have been introduced. In some cases, records of what services have actually been provisioned in the network do not exist due to poor operational practices at a service provider, resulting in a network that is overall poorly planned and resources in the network being tied up without generating revenue. In each case, it would be desirable to have a capability to auto-discover services (and network-layer concepts such as connections, which we include in our discussion without each time explicitly referring to them separately).

So what do we mean by service auto-discovery? We refer to it as the ability of a management system to automatically detect what service instances of a given type of service are present in a network, without requiring the user or an OSS to "tell" the system about those instances. In general, this involves the ability to derive service information from network element level management information ("NE MOs"), such as information on the logical configuration of network devices. To provision a service, configurations need to be applied and driven down into the network, often several devices. Hence the service instances present in the network can be "reverse engineered", respectively derived from the network configuration. Service auto-discovery assumes that information about the network elements is already known to the management system and in place, as it is a prerequisite. That information could be known as the result of (network element) auto-discovery and device configuration discovery that has taken place earlier. Figure 1 depicts the relationship between service auto-discovery and service provisioning.

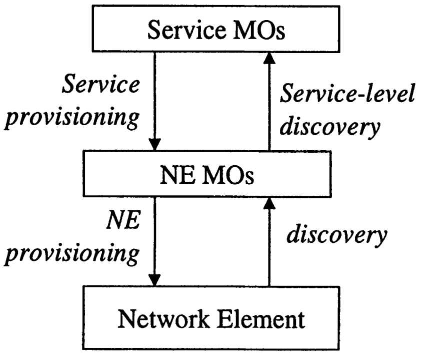


Figure 1: Relationship between provisioning and discovery

The need to auto-discover management information applies also to the network management layer. An example are connections in an ATM network. For the purposes of this paper, we treat network and service level auto-discovery jointly, and our concepts apply to both. MOs at the network management layer and at the service management layer are referred to collectively as "service MOs".

Management information constitutes the starting point for our considerations on service discovery. Service (and network) management layer information builds and depends on information at the network element layer, aggregating and abstracting information from it. Hence the mapping of service and network layer concepts onto information concerning individual network elements is reflected by the relationships between NE MOs and service MOs that express these dependencies (figure 2).

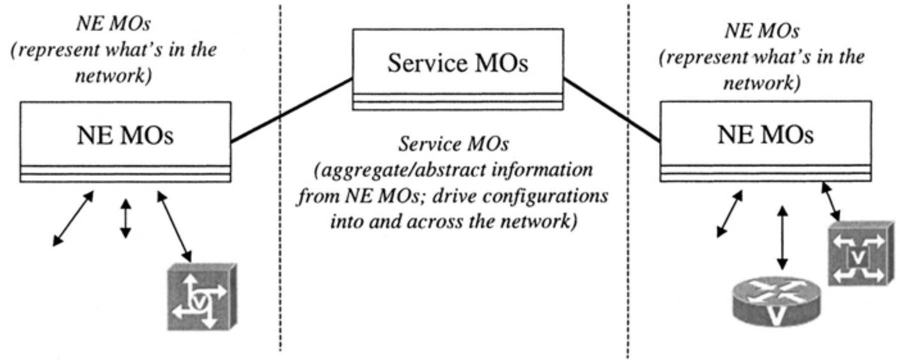

NE MOs
(represent what's in the network)
NE MOs
Service MOs
Service MOs
(aggregate/abstract information from NE MOs; drive configurations into and across the network)
NE MOs
(represent what's in the network)
NE MOs

Figure 2: Dependencies between NE and service MOs

The knowledge of these relationships is essential for service auto-discovery. One focus for service auto-discovery is hence to identify and be on the lookout for NE MOs of certain classes that are capable of maintaining relationships with service MOs, respectively that service MOs usually are dependent upon. The presence of such an NE MO is then an indication that a service MO may also be present. For example, an NE MO representing the endpoint of a connection is an indication for a corresponding MO representing the connection itself. Another trickier example would be an NE MO representing a DS0 port which may or may not be assigned to a service (and hence related to a service MO). Because service MOs usually aggregate and abstract information from several NE MOs, possibly spanning across multiple NEs, part of service auto-discovery concerns also identifying the various NE MOs that go together and collectively support a service MO. In the connection example, this would involve another NE MO that represents the other connection endpoint. In the port example, it might involve a set of other MOs representing connection endpoints, cross connects (relating the port to the connection represented by the connection endpoints), and service features on a feature service that refer to that particular port that jointly together with the port make up a residential subscriber service. Accordingly, the following are key aspects for service auto-discovery:

- Identification of NE MOs of classes that service MOs usually depend on
- Identifying which NE MOs would match up to relate to the same service instance

- Creation of service objects based on aggregate NE MO information

In addition to service information that can be derived from the network, there can be certain aspects about service objects that cannot be derived, as they concern service layer aspects that are not represented in the network elements but are maintained by the service provider. An example for this would be the customer information about who subscribes to a given service instance. This type of business information is not part of the network configuration and its automatic discovery from the network would require clairvoyant capabilities. Clearly, there may be information in a business management system that might be associated with the network information. However, this is not the emphasis of this paper and would be subject to further work; we assume that this type of information will still need to be associated by the service provider.

3. SERVICE LAYER AUTO-DISCOVERY CHALLENGES

As indicated, key to our approach to service-layer auto-discovery is the identification of NE MOs that service MOs could be related to and that hence indicate their possible presence, and derivation of service-layer information from those NE MOs. This sounds reasonably straightforward, although as so often the devil is in the details. The following are some of the aspects that need to be considered.

Multiple ways to instantiate a service. There can be different ways in which the same instance of a service can be instantiated. This means that in general, we need to know about different possible mappings, so we know what things to look for. Of interest are not so much the differences and variation in actual provisioning steps taken, but different ways in which the same service can be reflected in the resulting network configurations.

Identification of "matching" NE MOs. Generally, a service MO does not map one-to-one on an NE MO but is distributed over several MOs. This raises the question how we can find out which NE MOs "match", respectively would be related to the same service MO. How can we tell whether they potentially belong to the same service MO, or whether they would belong to different ones? In many cases, NE MOs related to a service MO contain information about other NE MOs to which they are related. An example are NE MOs that represent connection endpoints and contain information about the IP and port addresses that help identify the corresponding endpoint on the other side. (That MO in turn contains the first MO's NE's IP address information.) In some cases, only some of the NE MOs related to the same service object may have information about their relationship to the other NE MOs. An example is a DS0 port MO, cross-connect MO, and connection endpoint MO on the same network element that are all related to the same subscriber service. The cross connect MO may contain information relating it to the DS0 port and to the connection endpoint, but the connection endpoint MO may not be aware of it relating to either cross connect or DS0 port (see figure 3).

For such cases, service discovery generally needs to focus on the NE MOs that would have information allowing to identify other NE MOs related to the same service MO first. Only subsequently the discovery will expand to try to find the other

NE MOs that contain no such information but that were in part identified by the NE MOs found earlier.

Dealing with rainy day scenarios. Service instances may have been misconfigured. For example, referential integrity problems may exist. An example concerns signaling backhaul in packet telephony, where a media gateway controller might have been provisioned to backhaul signaling to a certain media gateway, but the media gateway expecting signaling backhaul to occur between it and a different media gateway controller. The way management information is represented must account for this possibility. It needs to be able to represent network and service layer information "as built", which includes a lot more possibilities compared to a representation of management information "as planned" that does not need to account for all the things that can possibly go wrong. Information models that represent services typically model services only "as planned", as indicated for instance in the cardinality of relationships. For instance, modeling of a point-to-point connection always involves two endpoints. How would the same information model represent a connection that was "broken" because the endpoints don't match up, violating referential integrity? The model needs to be capable to represent both, the network and services "as planned" with all their constraints, and the network and services "as built", with possible violations of planned constraints. Alternatively, two parallel models need to be maintained. In any event, we need to be able to deal with the world the way it is, not the way we wish it would be. A related question concerns how rainy and sunny day scenarios can even be distinguished. For example, if an NE MO is missing, it might be because truly a misconfiguration has taken place, or because the other NE MO has simply not yet been identified or discovered.

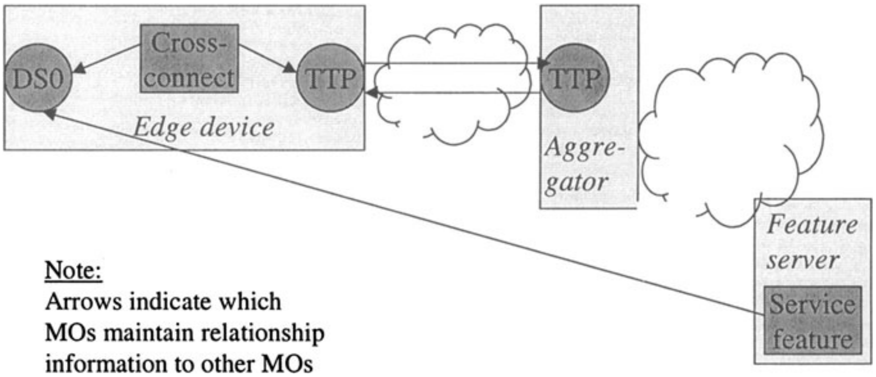


Figure 3: NE MOs involved in providing a residential subscriber service

Incremental discovery. In general, it is unacceptable having to wait for a long period of time, until all network information is precisely known (which might never fully be the case due to the constant changes occurring in real-life networks), to start discovering service MO. Service auto-discovery needs to be able to cope with incomplete information about the network, filling in missing pieces as it goes along and able to indicate to users the status of the discovery process. This is also related to the previous point.

Obtaining accurate NE information: Information has to reflect the current information in the network, requiring periodic synchronization of the NE information that service-layer auto-discovery is based on. This is associated with the usual challenges of keeping a management cache from going stale, nothing unique to service layer auto-discovery but nevertheless worth to be mentioned. Periodic upload and synchronization can be complex if the network or the EMS does not have the ability to provide only the changed information. If the complete network information is uploaded every time, the onus lies on the NMS to find out the deltas that changed since the last upload and update its information accordingly. Event based mechanism can be challenging if either the configuration change events do not carry enough configuration information, if there are too many of them, or if they are not delivered in a reliable manner.

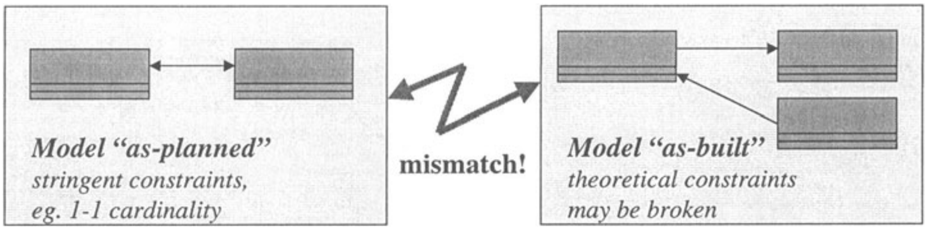


Figure 4: Rainy day scenarios – discrepancies between "as built" and "as planned"

4. AN APPROACH FOR NETWORK AND SERVICE LAYER AUTO-DISCOVERY

The following outlines our approach to discover service MOs from the network. It has been applied in the context of packet telephony networks as well as Metro Ethernet networks and expect that it can be applied also to other domains.

4.1 Initial analysis

We start with an analysis of the model of the management information that is involved. We assume that a model representing management information from the NEs, i.e., containing the NE MOs, will already be in place. (As mentioned earlier, as a prerequisite for service auto-discovery we assume that NE MOs have been discovered already, so they need to populate some model.) Certain categories of NE MOs will typically be part of (or referenced by) a service MO. We will refer to those NE MOs as "service-supporting NE MOs". For example, a termination point will be indicative of a connection.

If we do not have one already, we define a model of the network and service layer concepts that need to be discovered. This model can be arbitrarily defined; it can even include new services that were not known at the time when the NE MOs were originally defined. As far as the service MOs aggregate and abstract information originating from the network, they have dependencies on NE MOs. We need to explicitly identify these dependencies respectively relationships between service MOs and service-supporting NE MOs. Service-supporting NE MOs generally maintain relationships with other service-supporting NE MOs, jointly serving to support the higher-layer abstraction represented by the service MO. In many cases, the NE MOs will contain information that points to the other NE MOs that they are in relationship with. For example, an NE MO representing a termination point may include the IP address and port number of the remote end.

Knowledge of these relationships and dependencies is the basis on which the rules can be defined according to which discovery takes place. Some of these rules will be "triggering rules", which provide the conditions under which a service object will be created. The trigger generally identifies certain service-supporting NE MOs, which we will call "master NE MOs", that are a certain indication that a service MO is present. It is important that master NE MOs are defined such that there is only one master NE MO per service MO, so not to inadvertently introduce too many service MOs. Redundant service MOs would be difficult to match and eliminate later. In many cases, there are different possible candidates that could serve as a master NE MO. Often, information models are "symmetrical" in that for instance all the service-supporting NE MOs are of the same type. In this case, a master NE MO can be identified not by the NE MO type but by another distinction, for instance through the system it is contained in. In the packet telephony case, the call controller is generally considered key to the configuration, accordingly the service-supporting NE MOs of the media gateway controller are considered the "master". In peer-to-peer cases, a convention which of the service-supporting NE MOs should serve as the master NE MO could be the NE MO whose NE's IP address is lower than that of the other NEs to which it points.

Other rules will be "completion rules", which identify other dependencies of the service object, i.e., the other service-supporting NE MOs that must be in place for a service object to be "complete", or consistent. When a service MO is first created, it is only related to the master NE MO that triggered its creation. Since other NE MOs typically support this service MO, the information contained in it is incomplete. Hence we also introduce a state concept to indicate the status of discovery, termed the "discovery state". When initially created, a service MO will typically have a discovery state of "incomplete". It moves to a completed state once the completion rules have been satisfied, respectively all NE MOs that the service MO depends on identified. The discovery state also helps deal with cases where services were misconfigured, for example where referential integrity is violated and other NE MOs that are needed for the service to be consistent and complete cannot be found. An incomplete discovery state can be an indication for such situations.

Finally, aggregation rules will define the computation of any derived attributes. All those rules could conceivably be specified separately and processed by a discovery inference engine. However, in our case we chose to simply encode those rules in the respective discovery algorithms of the systems that we implemented.

4.2 Steps during runtime

The steps that occur during service layer auto-discovery are accordingly as follows:

The first step is really outside the scope of service layer auto-discovery itself and is a prerequisite for the auto-discovery of service MOs that follows. It involves discovery of the NE MOs. This means management information is uploaded from the underlying element management system or network element, as part of initial upload or (later) of synchronization. (The network elements will generally be auto-discovered themselves; however, it is not a prerequisite as this information could also be populated using some other mechanism.) Internally, MOs representing those NE resources are created. This includes physical aspects (cards, ports) as well as logical aspects (protocol entities, termination points, etc.).

Next, NE MOs are scanned to identify service-supporting NE MOs, for instance NE MOs of certain classes. Examples are termination points (indicative of a connection), trunk group controls in a call agent (indicative of a trunk group), a cross-connect in an edge device that relates a DS0 with a trail termination point connecting to an aggregation device (indicative of a residential subscriber service, as per the earlier depicted example). A matching NE MO will in all likelihood exist on another NE, e.g., a matching termination point in the case of the connection or a DS1 in case of the trunk group control. If a matching NE MO is found, a service MO with those NE MOs should be created.

Master NE MOs are identified. For each master NE MO, a service MO is created as a result. This can occur before a matching counter piece is found. In the MGCP-based packet telephony case, the media gateway controller is generally considered key to the configuration, accordingly the service-supporting NE MOs of the media gateway controller are considered the "master". The service MO will be marked as "incomplete", as not all NE MOs that the network/service layer concept is composed of are identified. However, the service MO will have enough information to locate the "missing" NE MO. For instance, a trunk group control will indicate the port number, slot number and shelf name of the DS1 it is supposed to be controlling, or a termination point has the address of its counterpart on the other side.

Subsequent steps attempt to identify the other NE MOs that support the service MOs that have been created. This can be done as NE MOs are discovered, or in an extra pass scanning all the NE MOs. Matching NE MOs can be identified through the specific semantics of the underlying model. A simple example concerns network connections: a network connection service object could be initially created with the information contained in an MO representing a connection endpoint. This MO contains the far end's IP address and port number. NE MOs from the far end's systems can now be searched for another connection endpoint object, that has as far end the initial NE MO's NE's IP address (and corresponding port number).

As service MOs are "completed", they will be marked with a discovery status of "complete". Also, information aggregated and abstracted from the supporting NE MOs can be computed. If conflicting information is found between the service supporting NE MOs, the service MO can be marked as "inconsistent".

Finally, the service provider has the possibility to associate the identified service instances with other service-related information from the OSS, such as customer information.

The steps can be interleaved. For example, it is possible for NE MO auto-discovery (or discovery) to take place while auto-discovering service objects, performing analysis of service-supporting NE MOs as things go along. Also, multiple passes may be applied. In the first pass, information is extracted from the network and raw service MOs, based on master NE MOs are created. In the second pass, service MOs are refined and completed.

Finally, it is possible to discover service MOs on an ongoing basis, even after the initial auto-discovery pass. Changes to NE MOs will trigger auto-discovery rules to be re-evaluated, based on whether the NE MO was associated with a service MO to ensure that service integrity is still met), whether the change should trigger creation of a new service MO, or whether the change implies that the NE MO can be newly associated with an existing service MO

To identify network-layer inconsistencies and misconfigured services, a user should check for service MOs with a discovery state of incomplete or inconsistent. Also, service-supporting NE MOs that are not related to any service MO are indicative of "orphaned" resources in the network that lie idle and should for management purposes be garbage collected.

5. APPLICATION EXAMPLES

As mentioned earlier, we have realized the presented service and network layer auto-discovery concepts in two systems addressing two very different domains. One (PTC) concerns the management of packet telephony networks, the other (ISC) management of metro Ethernet, specifically Transparent LAN Service (TLS). This is an indication to us that the concepts are indeed generic and will be applicable to other areas as well.

Packet telephony. The fundamental ideas underlying the system for packet telephony management have been described in [2]. Central to it is the notion to hide the distributed nature of a packet telephony network by projecting virtual entities onto it that provide a logical management wrapper around the physical network. This greatly simplifies its management, as management complexity that results from the distribution is largely abstracted away. Examples for virtual entities are a virtual switch that represents a media gateway controller (MGC) and the media gateways (MG) that it controls along with the various signaling and control connections between them in an MGCP network, or a virtual zone representing a gatekeeper and a set of associated gateways sharing the same dial prefix in an H.323 network. The virtual entities constitute a mix of network and service management layer objects to which auto-discovery can be applied. Provisioning of packet telephony networks can be fairly error prone, therefore the ability to detect network-layer configuration mismatches using auto-discovery of the virtual entities proved to be a very attractive side aspect of the system.

An analysis of the packet telephony information model [3] yields the service-supporting NE MOs, basically the NE MOs that the various virtual entity MOs are related to. Most of the virtual entities are based on "symmetrical" relationships between a virtual entity (service layer) MO and two NE MOs of the same type. We declare the NE MOs contained by the media gateway controller the master NE MOs. Key to discovering the various virtual entities is discovering the top-level virtual entities, i.e., the virtual entities that contain the other virtual entities, for example the virtual switch in an MGCP-based network. The virtual switch can be easily identified by identifying the MGCP connections, derived from the MGCP termination point maintained by a media gateway controller and the MGCP termination point of the media gateway that it points to. In a first pass, the virtual switches are identified this way. In a second pass, the other virtual entities are identified, i.e., the various aspects contained in the virtual switch, such as trunk groups, trunks, or backhaul connections. Matching up of the related NE MOs is fairly straightforward once the virtual switch itself is known, as it is clear where to look for the counterparts of the MGC's NE MOs.

The algorithm as described is of course somewhat simplistic. In reality, some aspects turned out to be quite tricky. For example, there can be multiple MGCP associations between an MG and several MGCs in a failover configuration in order to provide fault tolerance. At the same time, an MGC can have MGCP associations with multiple MGs, for which however the same failover configurations will need to apply. This leads to very sophisticated construction principles for the virtual entities and to fairly complex configuration constraints whose integrity must not be violated, respectively many rainy-day scenarios. Another challenge was the interdependence between service layer objects. Service objects representing PRI signaling backhaul, trunk groups, and DSx lines are dependent on creation of MGCP association and virtual switch objects. Unless MGCP association as well as virtual switch objects are discovered first, PRI backhaul and other service objects cannot be discovered because they use the association knowledge between MG and MGC. Thus the complete discovery process involves multiple phases where the initial phase discovers those MOs that have no dependency on other MOs, and subsequent phases discover those MOs that are dependent on already discovered MOs.

Metro Ethernet. Metro Ethernet management [1] deals with Transparent LAN Service. TLS could constitute a multipoint to multipoint connection or a single point to point connection and is used to transparently connect LANs at multiple customer sites as one single LAN. TLS is abstracted as a service object and is formed by grouping multiple network layer objects. Refer to Figure 5 for a TLS example. In the example, TLS connects three customer sites LANS together to form one virtual LAN. The end user is interested in only the endpoints of the TLS and does not care which intermediate nodes the circuit goes through. However, the TLS is comprised of multiple segments (6, in this case) at the network layer. As a part of auto-discovery, the information is read from multiple devices, individual segments are formed, and then these segments are grouped together to form a TLS service.

When the management system is connected to a live network where some TLS circuits are already provisioned, the management system auto-discovers all the TLS services and makes them visible to the user. This is also useful for further resource allocation in the network for the services provisioned later. The management system can keep track of resources such as ports, VLAN IDs which have already been used by the auto-discovered services and does not let operator use those resources.

Similar to the case of packet telephony management, auto-discovery can also serve to detect network layer inconsistencies. For example, a segment belonging to TLS1 can be misconfigured to belong to TLS2. When NMS uploads the network objects from individual devices and correlates these objects, the algorithm can detect the inconsistency and flag it to the user through its GUI.

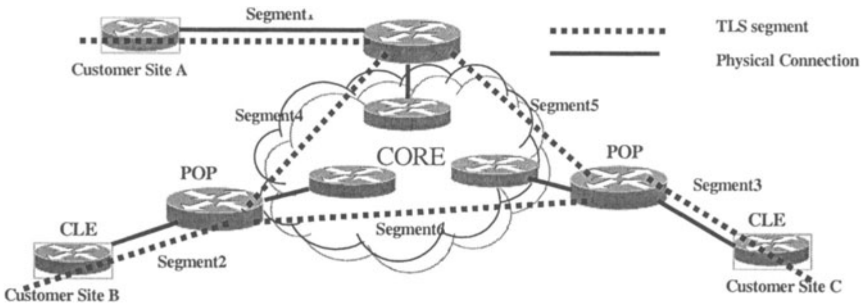


Figure 5: TLS in a Metro Ethernet

6. CONCLUSION

In general, auto-discovery focuses on the network element management layer. However, there are legitimate and important reasons to extend auto-discovery to higher management layers as well. The concepts discussed in this paper have been utilized by management systems for Open Packet Telephony and for Metro Ethernet, with convincing results. The most important aspect perhaps is a side effect of the auto-discovery itself, namely the ability to detect network-layer inconsistencies in the network where service objects are not able to reach a discovery state that indicates they are consistent and complete. Some of our practical experiences have been that service layer auto-discovery is however an expensive operation that should be used sparingly. It is applied once during initial cold start of the system and subsequently on operator request; essentially in situations where the service provider has reason to believe that the service information is no longer accurate. An important feature of our systems is the ability to restrict service-layer auto-discovery to a certain scope, such as in packet telephony an H.323 zone. The scope should of course be defined such that the service-layer concepts within it are self-contained, so that unnecessary and erroneous flagging of seeming inconsistencies – where service-layer aspects extend to information outside the scope is avoided.

Future work could aim at deriving auto-discovery rules automatically from service definitions used to provision the network, such as the ones defined in and applicable to systems and methodologies such as the ones described in [4, 5, 7, 10]. Another area for further research concerns matching auto-discovered data with information contained in the service provider OSS. This would allow for instance to automatically associate subscriber information with service instances identified in the network, so really complete service auto-discovery with service aspects that are not to be derived from the network side.

ACKNOWLEDGMENTS

The authors would like to thank Prakash Bettadapur, Poon Leung, and Petre Dini for fruitful discussions on this subject.

REFERENCES

[1] Barry, D.: Metro Ethernet Management. Packet Magazine 3/2002, softcopy at http://www.cisco.com/warp/public/784/packet/jul02/p45-cover.html, 7/2002.
[2] Clemm, A., P. Bettadapur: Building Management Solutions for Open Packet Telephony Networks. IEEE/IFIP IM 2001, Seattle, WA, 5/2001.
[3] Clemm, A., P. Leung: Model-Driven Open Packet Telephony Management. IEEE/IFIP NOMS 2002, Florence, Italy, 4/2002.
[4] Dreo Rodosek, G., L. Lewis: Dynamic Service Provisioning: A User Centric Approach. IEEE/IFIP DSOM 2001, Nancy, France, 10/2001.
[5] Garschhammer, M., R. Hauck, B. Kempter, I. Radisic, H. Rölle, H. Schmidt: The MNM Service Model – Refined Views on Generic Service Management. Journal of Communications and Networks (JCN) Vol. 3 Nr. 4, 12/2001.
[6] Jacob, B.: Service Discovery: Access to Local Resources in a Nomadic Environment. OOPSLA'96 Workshop on Object Replication and Mobile Computing, San Jose, CA, 10/1996.
[7] Kong, Q., I. Rose, D. Cameron: Towards Technology Independent and Automated Service Activation and Provisioning. IEEE/IFIP NOMS 2002, Florence, Italy, 4/2002.
[8] Lewis, L: Managing Business and Service Networks. Kluwer Academic / Plenum Publishers, New York, NY, 2001.
[9] Preuss, S.: JESA Service Discovery Protocol (SDP). Proceedings Networkers 2002 (Springer), Pisa, Italy, 5/2002.
[10] Shen, F., Clemm, A.: Profile-Based Subscriber Service Provisioning. IEEE/IFIP NOMS 2002, Florence, Italy, 4/2002.

MANAGING HETEROGENEOUS SERVICES AND DEVICES WITH THE DEVICE UNIFYING SERVICE

Implemented with Parlay APIs

Erik Vanem, Dao Van Tran, Tore E. Jønvik, Pål Løkstad and Do Van Thanh
Telenor R&D, Snarøyveien 30, 1331 Fornebu, Norway

Abstract: Due to the ubiquitous explosion of the Internet and the fast proliferation of heterogeneous networked devices and services, the end user are faced with a formidable task in managing it all. The Device Unifying Service presented in this paper is a novel powerful concept that allows anyone confronted with multiple devices and services to administer and manage them by themselves. With this service, different devices connected to different networks offering different types of data and communication services can be integrated and handled in a straightforward way. It allows the usage of multiple devices simultaneously and the possibility of adding, removing or changing devices in a service session. The Device Unifying Service addresses personal service management for both communication services and data services. Communication services like a normal phone call can for example easily be transferred from a fixed phone to a mobile phone or an IP phone with multimedia capabilities with the Device Unifying Service. Additionally, a data service like a WEB browsing session can be transferred from a PC to a laptop or PDA including cached information like history and bookmarks. Providing an easy way of moving services between devices, the Device Unifying Service thus fulfils the vision of service portability in an elegant way. This paper presents the concept of the Device Unifying Service, a novel service that will take care of the management, coordination and configuration of all the devices that the user has at his disposal and provide service portability and adaptation of services to devices with different characteristics. Furthermore, this paper will reveal how the first version of the Device Unifying Service has been implemented using Parlay APIs and H.323 technologies.

Key words: Service Portability, Management of Networks and Services, Personal Communications, Mobility Evolution, VHE, IP/mobile technologies, Open APIs, Interoperability and Cooperative Control, Personal Network Management.

1. INTRODUCTION

The main concepts of the Device Unifying Service (DUS), developed within the Eurescom project P1101, has already been described in previous papers [1,2], and will therefore only be given in brief in this paper.

The Virtual Home Environment (VHE) [3] is an important portability concept of the 3G mobile systems that enables end users to bring with them their personal service environment whilst roaming between networks and also being independent of terminal used. Although considering multiple terminals, the VHE concept addresses only the usage of one terminal at the time and the swapping between terminals. It is thus envisaged the needs of using multiple device simultaneously and the needs of coordination such that devices collaborate and provide a coherent user interface.

Nowadays the user is confronted with several different devices such as a plain-old telephone, a mobile phone, a cordless phone, and a PC or a workstation as well as more simple devices like microphones, loudspeakers, TV screens, digital cameras etc. These devices might very well be autonomous in that they are able to function individually and independently of each other, but the proposed service will allow the user to manage them all and hence consider them all as one big terminal - the Virtual Terminal.

Another important issue is the combination of mobile and stationary devices. The user might visit places away from his home domain where there are stationary devices offering different types of services, e.g. printers, screen displays etc, and the DUS will allow the user to include these in his Virtual Terminal, i.e. it will offer a dynamic configuration of the Virtual Terminal depending on what devices and services are available on any given time.

The DUS will manage data services as well as communication services, and it enables moving of data sessions. For example, an e-mail that is halfway written can be transferred to another device before continuing with the writing, and a web browsing session could be transferred with history, bookmarks and all between devices.

This paper describes the different aspects of the Device Unifying Service and the Virtual Terminal concept. The functional requirements are identified and the system architecture is discussed. Finally, the implementation of the first version of DUS, which utilizes Parlay [4] call control and a Parlay-H.323 gateway, is described.

2. THE DEVICE UNIFYING CONCEPTS

2.1 Device unification and coordination

Since many communication devices, e.g. mobile and fixed phones, may have a limited user interface, the combination of these devices with various other devices with different capabilities into a Virtual Terminal as illustrated in Figure 1, may result in much better user interfaces and hence enhance the offered services.

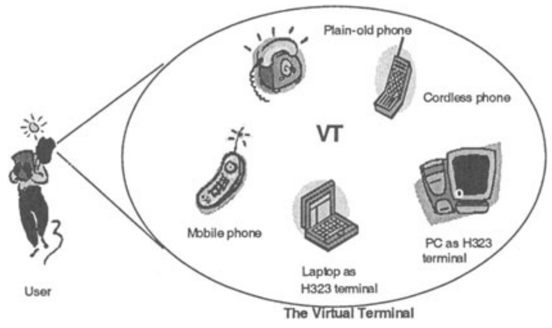


Figure 1. The Device Unifying Service

During a service session, a user may want to use one or more elementary devices with just input and output functionality in addition to his communication device. Examples of such elementary devices can be a big screen display, loudspeakers, a microphone etc. In other cases the user may want to reroute the voice or data streams coming in to his mobile phone to other, more convenient devices such as a fixed phone, a PC etc. This can be done via the mobile device using technologies such as Bluetooth [5], but due to limitations of the mobile device, it might prove a better alternative to introduce the Device Unifying Service and let it handle this.

With this concept, all the different devices will be considered as one big terminal with multiple input and output capabilities. The user terminal will no longer be an integrated and recognizable device but a set of distributed devices that allow access to certain services, e.g. multimedia communication services, various computing services etc.

2.2 Virtual Terminal management

The Device Unifying Service will allow the user to define, add and remove the devices that are included in the Virtual Terminal at any time. The user's many terminal profiles and user profiles on different devices will be replaced by a unique user profile defined once for all in the Device Unifying Service. Whenever this profile is set up or updated, it will automatically be accessible from all other devices contained in the Virtual Terminal.

The profiles can also be set up to specify different personal preferences when it comes to subscription of services, user interfaces on particular device types etc so that the user will be offered the same services and experience the same look and feel regardless of where he is and what particular device he is using. The only requirements are that the device in question supports such services and such user interfaces.

2.3 Service portability and adaptation

Service portability and adaptation to devices with different characteristics are another important feature of the DUS. Moving a telephony call from an ISDN phone to an IP phone or GSM phone will be made possible and the signaling and media are adapted to comply with the end-point. Likewise, a data session can be transferred and adapted from one PC to another, from a PC to a laptop or to a PDA. This is illustrated in Figure 2.

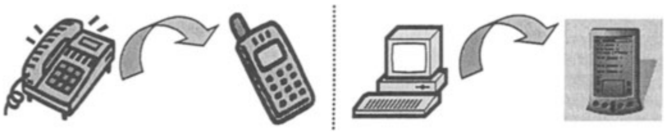


Figure 2. Service portability for communication services and data services

2.4 A personal secretary

The proposed service will be user centric instead of device centric in that it will be possible to address a person directly instead of addressing one of his devices, as illustrated in Figure 3.

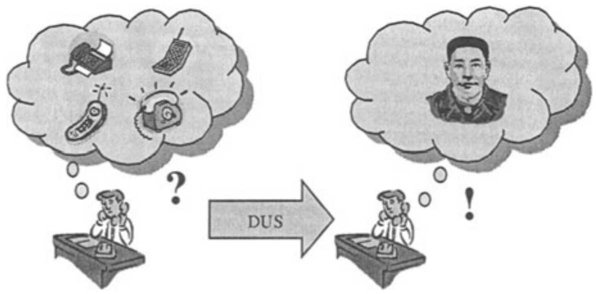


Figure 3. DUS as a personal secretary

The Device Unifying Service will be responsible for handling available devices, which may include a variety of personal or public devices. Anyone wishing to communicate with him will dial up his Device Unifying Service, which in turn will contact the user. In other words - the Device Unifying Service will function as the user's personal secretary.

The Device Unifying Service will offer personal mobility to the end user in that there are no fixed relations between the user and any specific device. In this way, the different identities of the different devices will become transparent, but the identifier belonging to the Virtual Terminal will stay unchanged.

3. FUNCTIONAL REQUIREMENTS

The functional requirements of the Device Unifying Service describe what functions it should be able to perform. The following are identified as the main functional requirements of the DUS:

- Initiating services – The user should be allowed to initiate any services through his Device Unifying Service.
- Receiving services – The user should be able to receive any incoming services via his DUS.
- Using multiple devices – The Device Unifying Service should allow the user to combine and use multiple devices as if they where one in the same session, both simultaneously and successively.
- Configuring the devices – The Device Unifying Service should help the user to automatically configure his devices whenever this is necessary.
- Managing the Virtual Terminal – It should be possible for the user to dynamically manage his Virtual Terminal, i.e. specify which devices are components of the Virtual Terminal, handle different kinds of services, define his different personal service environments, set up his preferences, access and edit his user profiles, read his mailbox, edit his address book, modify his schedule etc

These requirements apply for both the communication services part and the data services part of the DUS, and from the user's point of view, no great distinction between data and communication services should be noticed.

4. LOGICAL SYSTEM ARCHITECTURE

In order to realize the described service, a terminal management and coordination function is needed – the Device Unifying Service. In short, such a function should have the following capabilities: It should continuously maintain, monitor and update the configuration of the Virtual Terminal, i.e. it should always know exactly what devices are present and active in the Virtual Terminal. It should also be able to multiply and deliver streams from applications to respective devices and to unify and deliver streams from devices to respective applications, when multiple devices are active in one session. This function should be located on a server connected to a standard IP-network, which is again connected to other networks, e.g. Internet, GSM/ISDN network through gateways. This conclusion regarding the location of the management logic is in agreement with other research projects' conclusions [6].

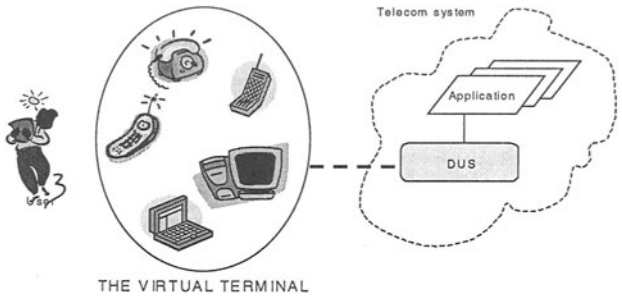


Figure 4. The logical system architecture

Figure 4 shows the proposed logical system architecture of the Device Unifying Service. Other applications are only aware of the DUS and not of the user's many devices. They will deliver services to the DUS, which again has the responsibility to ensure that the delivery to the user is done through the most appropriate device – a device within his Virtual Terminal. The DUS thus manages the users' various devices and services, while the user manages the DUS itself.

5. DUS COMPONENTS

In the design of DUS different management components are drawn up. Figure 5 shows the different components and how they are linked together.

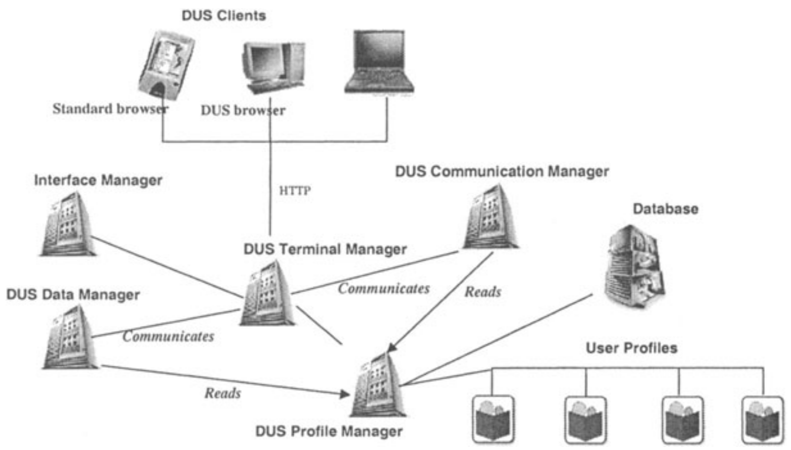


Figure 5. The DUS components

The different DUS components provide different types of functionality and are implemented as java classes.

- The DUS Clients are clients used to access the DUS management service. A standard web browser can be used to manage communication services (making calls, adding devices to an existing voice session, transferring calls etc.), but for the data services (e.g. transferring a WEB session) a special DUS browser, developed within the P-1101 project is needed.
- The DUS Terminal Manager is the way into the DUS Communication Manager or the DUS Data Manager. It is reached by the DUS clients by HTTP and communicates with the DUS Data Manager and DUS Communication Manager using CORBA.
- The Interface Manager provides different functions for generating the user interfaces, which are returned to the screens of the user's devices. These can be in the form of HTML or WML to suit different terminals with different capabilities.
- The DUS Profile Manager. This component stores the different profiles within the system. It also provides a range of functions to manage each profile. It can be accessed by the DUS Data Manager and DUS Communication Manager and provides information needed for the management of the data and communication sessions.
- The DUS Communication Manager handles the DUS communication sessions, i.e. voice sessions, multimedia sessions etc. This is the component that handles call control and sets up and tears down connections in the network between the actual devices. It reads from the DUS Profile Manager to get necessary information from the Database.

- The DUS Data Manager handles the management of the DUS data sessions and allow data services to be transferred between devices.
- The Database contains the User Profiles and communicates with the DUS Profile Manager using JDBC.
- The User Profiles contains the information about the DUS users, their different devices, services, preferences, environments etc.

Together, these components make up the Device Unifying Service.

6. IMPLEMENTATION OF DUS VERSION 1.0

The first version of DUS has been implemented and deployed on Telenor's research labs at Fornebu, Norway. It has been implemented on an IP-based network with the components pictured in Figure 6.

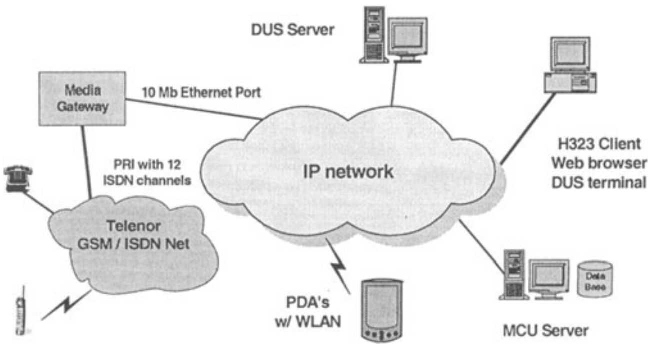


Figure 6. The physical parts of a DUS implementation

The DUS implementation contains the DUS server that contains the actual DUS application and a MCU server that is capable of mixing multiple media streams. Figure 7 illustrates how the DUS application is connected to the user terminals via the MCU server that mixes the media streams in a communication session.

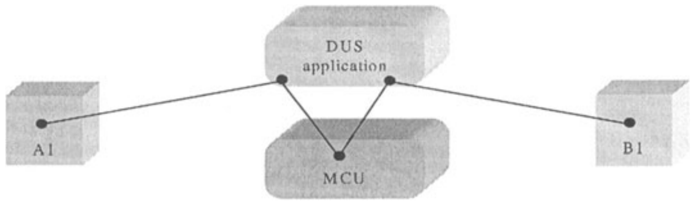


Figure 7. Connection between DUS application, MCU and user terminals

The DUS server also has access to a database where the user profiles, the device profiles, the address books and the schedules are stored. The DUS user can access and administer the service from a standard PC with a Web browser or from a PDA as indicated in the figure.

The DUS application itself runs on a standard PC with an industry standard operating system such as Windows, Unix or Linux. It is implemented in Java with development tools from Borland [7] and a Parlay[1] compatible development platform from Appium Technologies [8, 9]. The server machine running the DUS application has a Java Virtual Machine (JVM) installed and it contains a WEB application server (Apache/Tomcat), and a user profile server. JDBC is installed to ensure communication with the user profile server. In Figure 8, the components of the DUS application server are shown.

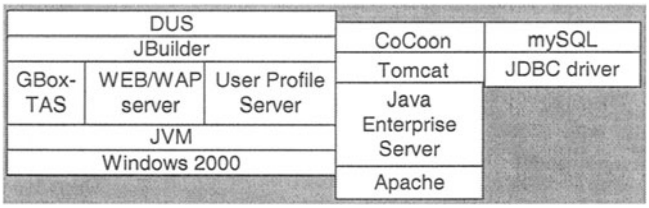


Figure 8. The DUS application server

The DUS application uses a Parlay-H.323 gateway from Appium technologies to handle call control for IP telephony. This contains an implementation of the Parlay APIs that are used for the actual call control and the signaling to and from the DUS server is performed by H.323 [10]. Such a call set-up example is shown in Figure 9. The GSM/ISDN network is connected to the DUS network through a signaling and media gateway from Cisco systems [11] so that both IP telephony and GSM/ISDN telephony can be managed by the DUS. The DUS network is also connected to the Internet.

Some of the most important Parlay APIs used in the DUS application are createCall, routeReq, routeRes, release, handleCall, getCallSessionID, getCalledAddress and getCallingAddress.

[1] Parlay APIs version 2.1 is used.

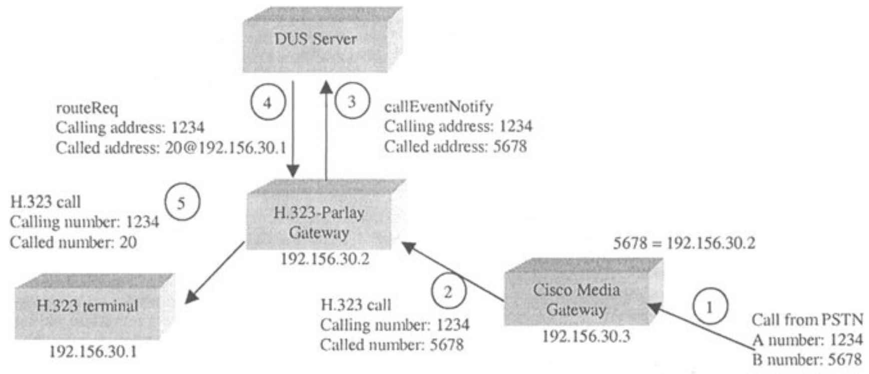


Figure 9. Incoming call via the H.323 - Parlay gateway

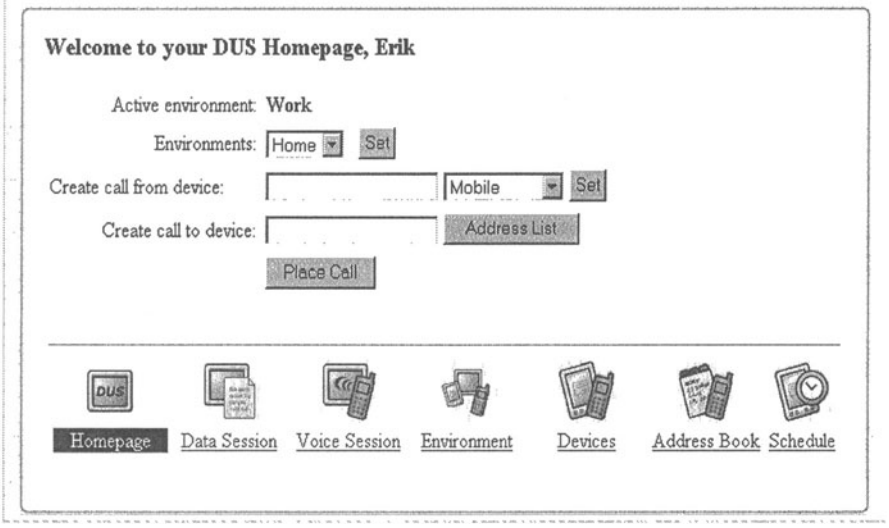
Version 1 of DUS that has been implemented so far consists of two parts: A communication services part and a data service part. The communication service part includes the following functionality: session creation, session teardown, adding devices to an existing session, removing devices from a session, session transfer between two or more devices, e.g. transferring a session from a fixed phone to a mobile phone and receiving services on most appropriate devices. The DUS have a globally unique identifier, so that the user can receive services through the DUS. This identifier consists of an E-164 number and a H.323 address so that it can be addressed by both traditional telephony via the gateway and IP telephony directly. The data services part provides transfer of data sessions like web browsing (including history, bookmarks etc), e-mail etc. In addition, the user may manage his profile and environments and modify a list of his own devices, an address book containing his contacts and a schedule that defines different environments according to the time of day. All this is accessible to the user from an ordinary web browser on a PC, laptop or PDA. These two servers are combined and integrated into one DUS server offering both communication and data DUS services.

With the DUS version 1.0 four of the five identified functional requirements are already realized. Only the configuration of devices requirement is left for further releases.

7. THE DUS 1.0 MANAGEMENT INTERFACE

The DUS version 1.0 offers its functionality to the user by an easy-to-use management interface. When subscribing to the service, the user will be given a username and a password, and should then be able to log on to the service from any PC, laptop or PDA connected to the Internet and with a web browser. A DUS homepage will appear, and from there it is easy to navigate between the different functions. The display in a traditional browser will naturally differ slightly from the

PDA display, but the services they are offering are essentially the same. In this section, some selected screenshots from the standard browser version of the DUS interfaces are exhibited.


Welcome to your DUS Homepage, Erik

Active environment: Work

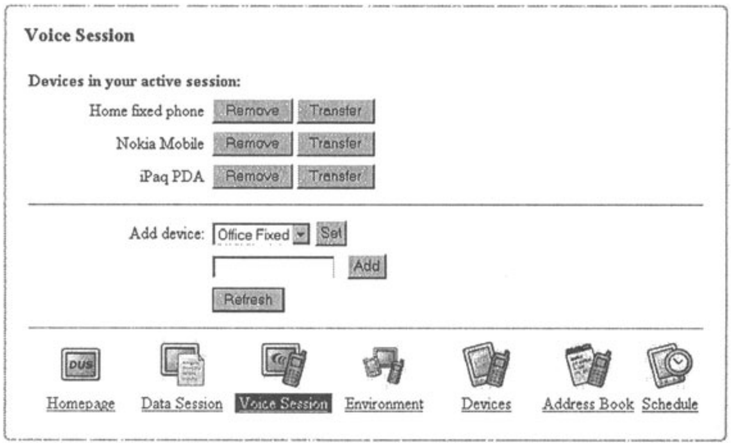
Environments: Home

Create call from device: Mobile

Create call to device:

Figure 10. The DUS Homepage

In Figure 10, the DUS homepage that appears as soon as the DUS user has logged in is shown. From this menu, the user can create a new call from any of his devices to any user. He can use the address book to call one of his contacts or type in the number directly. At the bottom of the screen, different icons allow for easy navigation between the different DUS management functions.


Voice Session

Devices in your active session:

Home fixed phone

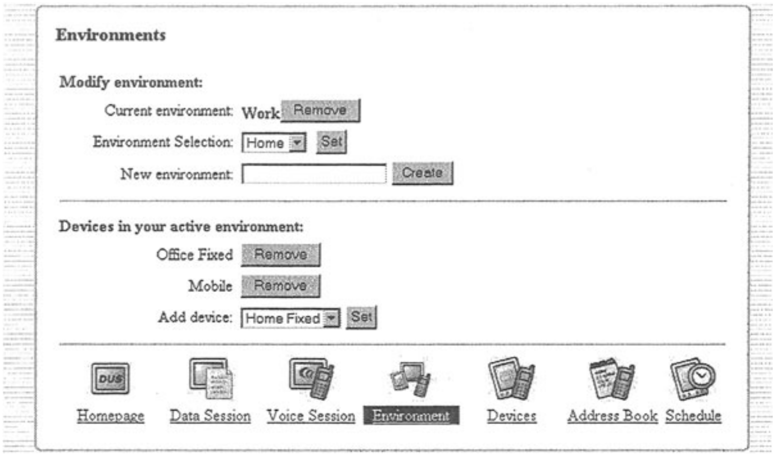
Nokia Mobile

iPaq PDA

Add device: Office Fixed

Figure 11. DUS voice session management

Figure 11 shows the DUS voice session management interface. From here, new devices can easily be included in an ongoing communication session, devices can be removed from the session and the whole session can be transferred to new devices. Different types of devices like ISDN phones, GSM phones or IP phones can be included in the same session.


Environments

Modify environment:

Current environment: Work

Environment Selection: Home

New environment:

Devices in your active environment:

Office Fixed

Mobile

Add device: Home Fixed

Figure 12. DUS environments management

In Figure 12, the DUS environment management is shown. From here, the user can set one of his environments, e.g. work, home, travel etc, to active, meaning that incoming calls will be routed to a device present in that environment. One can create new environments and define what devices that belong to the different environments. A device can belong to as many environments as desired, and for example the user's mobile device will typically be defined in all environments. The active environment can also be set by a default value or according to a predefined schedule.

In addition to the displays shown in this article, similar user interfaces allow management of data sessions, as well as management of devices, address book and schedule.

The DUS user interfaces are offered as Java Server Pages (JSPs) that the user can request from standard client browsers on either a PC or a PDA.

8. CONCLUSION

In contradiction to the commonly accepted assumption that the communication devices of the future will be integrated devices integrating several functions into one mobile device, this paper expect another trend towards several personal and public devices, both mobile and stationary, each offering different functionality to the end user, and each with a different network connection. These devices might be autonomous in that they are able to function individually and independently of each other, but they might also be coordinated so they can act together as one big terminal – the Virtual Terminal. The outlined service is an approach to allow the users manage its own Virtual Terminal and its own services through a management interface over a network.

This paper has presented the concept of the Virtual Terminal and described a novel service - the Device Unifying Service that will realize the Virtual Terminal. The requirements and functionality this service should offer is outlined and the design and architecture are discussed. A first version of the service has been implemented and the concept is thus proven. However, before the service is ready to be deployed as a large-scale commercial service in the real world, additional work should be done. Some of the technologies that would be desired to enhance the value of this service, i.e. service discovery and announcements via Bluetooth, location awareness, SIP etc. are still immature. When these technologies are more mature, they can be incorporated in DUS and add considerable value to the DUS user. Other aspects such as security and billing issues are not considered in much detail. Nevertheless, the first version is ready for deployment and will be introduced to real users in field trials shortly. The feedback from these surveys will unquestionably indicate how to improve version 1.0 to subsequent version.

The DUS can be of interest to a number of different service providers, as it does not require any heavy infrastructure to be deployed. Access to an IP network and some computers is all it takes. Operators and service providers offering this service can differentiate themselves from their competitors. It can also be offered as an enterprise service that incorporates office devices, mobile devices and home devices. Regardless of how the service is offered to the user, it will provide a valuable service that will be more and more relevant to the user as the number of different devices he has to relate to increases. The Device Unifying Service thus has a promising future deployed as a commercial service for heterogeneous device and service management.

ACKNOWLEDGEMENT

Work with the Device Unifying Service has been carried out within Eurescom [12], in project P1101 [13], with partners from Telenor ASA (Norway), British Telecom (UK), Hellenic Telecommunications Organization S.A. (Greece), Deutche Telekom AG (Germany), Iceland Telecom Ltd. (Iceland) and Czech Telecom (Czech Republic).

REFERENCES

[1] E. Vanem, D.V. Tran, T.E. Jønvik, D.V. Thanh, Extending VHE with the Device Unifying Service, Proceedings of 2002 International Conference on Communications, ICC 2002, New York City, USA, April 28. – May 2. 2002.

[2] E. Vanem, D.V. Tran, D.V. Thanh, Multimedia Communications with Multiple Devices Using the Personal Virtual Network Service, Proceedings of 2002 IEEE Wireless Communications and Networking Conference, WCNC 2002, Orlando, Florida, USA March 17-21 2002.

[3] 3GPP, Technical Specifications Group Services and Systems Aspects, Service aspects; The Virtual Home Environment, 3G TS 22.121 version 1.2.0, 1999-04.

[4] The Parlay group, (2002, July 23) [Online]. – URL: http://www.parlay.org/

[5] Bluetooth SIG, (2002, July 23) [Online]. - URL: http://www.bluetooth.com

[6] The ICEBERG project, (2002, July 23) [online]. URL: http://iceberg.cs.berkeley.edu/

[7] Borland, (2002 July 23) [Online]. URL: http://www.borland.com/

[8]The Parlay group – specifications (2002, December 11) [Online]. – URL: http://www.parlay.org/specs/index.asp

[9] Appium technologies (2002, June 25) [online]. – URL: http://www.appium.com/

[10]IEC: H.323 (2002, December 11) [Online]. URL: http://www.iec.org/online/tutorials/h323/index.html

[11]Cisco Connection Online by Cisco Systems, Inc. (2002, July 23) [Online]. – URL: http://www.cisco.com/

[12]EURESCOM, European Institute for Research and Strategic Studies in Telecommunications, (2002 July 23) [Online]. – URL: http://www.eurescom.de/

[13] P1101, Always on – heterogeneous services – everywhere and on any kind of terminal. (2002 July 23) [Online]. – URL: http://www.eurescom.de/public/projects/P1100-series/p1101/

SESSION 8

Distributed Management

Chair: José Nogueira
Federal University of Minas Gerais, Brazil

DELEGATION OF EXPRESSIONS FOR DISTRIBUTED SNMP INFORMATION PROCESSING

Rui Pedro Lopes and José Luís Oliveira
Polytechnic Institute of Bragança, ESTiG; 5300-302 Bragança; Portugal (rlopes@ipb.pt)
University of Aveiro, DET; 3810-193 Aveiro; Portugal (jlo@det.ua.pt)

Abstract: Due to the scalability problems of SNMP, management distribution has been an important topic during the last years. The DISMAN workgroup propose a set of MIB modules to address this matter. One of the DISMAN modules has the capability of using expressions to perform decentralized processing of management information – the Expression MIB.

Although it is essential to network management, the Expression MIB is not as well known as other DISMAN modules, such as the Script MIB, and not as available in terms of implementations. This paper focuses on the Expression MIB features, its implementation details and it also discusses, from a critical point of view, its functionality. It also proposes minor changes which can boost its application range and importance.

Key words: DISMAN, Expression MIB, Network Management, SNMP.

1. INTRODUCTION

During the last years the SNMP management framework has strongly guided the development of network systems and management applications. This architecture, regardless of some well-known shortcomings, has managed not only to survive but also to evolve to a rather complete set of features. This fact, combined with its inherent simplicity and APIs wide availability, has pushed it into a dominant position in today's network management market.

On the other hand, one of the problems associated with SNMP is its centralized architecture, not well suited to offline operation and not scalable on large networks.

According to many authors, the solution to this problem is management distribution, a research topic since the early 90's[1]. More recently, mobile agents have also been a hot research topic for management applications[2-5]. Considering the standardization of a common model for wide adoption in IP-networks, the IETF have also endorsed a working group to study this matter - the IETF Distributed Management charter, or DISMAN[6].

This group has set up a solid framework composed of several MIB modules, namely the Script, Schedule, Expression, Event, Remote Operations, Notification Log, Alarm and Condition MIB[6]. This set of MIB modules provides a rather complete framework for distributing management operations over a hierarchy of several midlevel managers – distributed managers (DMs).

The Script MIB is one of the best known, probably because of the early availability of implementations. Schoenwaelder, following an excellent study of distribution models and solutions, presents the distribution of management tasks in the context of the IETF Script MIB[7]. However, some other DISMAN modules seem to have captured less interest from the research community and have been left outside development plans. Our aim to evaluate management distribution in SNMP has also led us to the development and assessment of the Expression, Schedule and Event MIB.

In this paper, we present our work on the Expression MIB. We describe some implementation details, which are considered relevant in this paper context, and we evaluate the advantages and disadvantages of the current specification[8]. Based on the development experience and on its trial in the network, we also propose adaptations to the MIB that enhance significantly its functionality with minor modifications in its structure.

The paper is structured as follows: Section 2 presents a general description of management distribution in SNMP. Section 3 describes the Expression MIB model and our open source architecture. It point out the assessment of this management agent, and suggests some minor changes which can boost its applicability and potentiality. The paper ends with some conclusions.

2. MANAGEMENT DISTRIBUTION

The history of management distribution, well discussed by Martin-Flatin in[9], started with early work by Yemini *et al.* in 1991 when features such as scalability, flexibility and robustness were identified as necessary for future developments on network management[10]. Goldszmidt and Yemini early supported a management distribution methodology by delegating management operations near management information[1]. According to this concept, management processing functions are dynamically delegated to the network elements and executed locally. This introduces a shift in the original concept where the information is transported to a central location to be processed. This approach is known as Management by Delegation (MbD) and although the research prototypes did not have the expected community recognition they unquestionably proved the concept.

Other approaches for management distributions suggested using mobile agents to implement and distribute management functions. Many authors supported several usage scenarios, platforms and applications and enforced the concept of a cooperative management effort on the network[2-5].

The industry also adopted management distribution by releasing tools, APIs or agents, such as Sun's JMX[11] or SNMP Research's CIAgent[12].

Some of these products, technology and concepts do not easily survive the community resistance because they are neither compatible nor adapted to the management technology of choice – the SNMP. The SNMP community has also proposed, under the DISMAN workgroup of the IETF[6], several tools for management distribution.

2.1 Distribution under DISMAN

The typical usage scenario of the DISMAN architecture is based on the distribution of management tasks through a set of midlevel managers known as Distributed Managers (DMs). The main purpose of this approach is to reduce the command exchange with the management station, to alleviate the processing load usually residing at a single central point and to increase the system robustness by introducing redundancy and by allowing offline operation.

Several MIB modules were proposed to address different but complementary issues of management operations distribution. The Event MIB allows monitoring the real-time evolution of specific MIB objects either locally or remotely, and takes an action when a trigger condition occurs (a value outside a range limit for instance)[13]. The action can result in a set operation or in a notification.

The Notification Log MIB is intended mainly for notification providers but consumers may also use it. It defines a mechanism to cope with lost notifications by recording each notification data[14].

The Remote Operations MIB modules (`ping`, `traceroute`, `lookup`) enable the corresponding network-checking operation to be performed at a remote location. It provides a standard way to perform remote tests, to issue periodical sets of operations, and to generate notifications with test results[15].

The Schedule MIB provides the definitions to perform the scheduling of actions periodically or at specific times and dates. The actions are modelled by SNMP set operations on local MIB variables (restricted to `INTEGER` type). More complex actions can be performed by triggering a management script, which is responsible for carrying out complex state transitions[16].

The Script MIB module allows the delegation of management functions over distributed managers. Management functions are defined as management scripts written in a language supported by the managers. It may be a scripting language (such as TCL) or native code, if the implementation is able to execute it under its control. The module does not make any further assumptions on the language. The distributed manager may be decomposed into two blocks: the SNMP entity, which implements this MIB, and the runtime system, capable of executing the scripts. The Script MIB sees the runtime system as the managed resource, which is controlled by the MIB[17].

The Expression MIB was planned to move to the agent side part of the management information processing typically performed by managers. In other words, it supports externally defined computation expressions over existing MIB objects. The Expression MIB allows providing the Event MIB with custom-defined objects. The result of an expression can trigger an event, resulting in an SNMP notification. Without the Expression MIB such monitoring is limited to the objects in predefined MIBs[8].

The most recent modules are related to alarm reporting. In fact, SNMP is mainly based on polling instead of alarm reporting (a node sending notifications to the manager when status changes to and from fault conditions). These modules provide the necessary mechanisms to build up management procedures based upon exception handling. It starts by defining a model-neutral method to specify and store alarms[18] thus providing the DM access to SNMP alarm information in a consistent manner across systems.

The DISMAN architecture may be classified somewhere between the weak distribution model and the strong distribution model, according to the user defined distribution policies[19]. In this case, the degree of distribution depends on the set of operations defined in the DM, the complexity of the operations that the DM supports, the communication between them, and the total number of DMs.

It is possible to foresee several examples of scenarios suitable for DISMAN application. For example, the local processing of RMON probes statistics[20] according to specific expressions, the autonomous resource monitoring in workstations through the HOST-RESOURCES-MIB[21], the periodic analysis of data integrity in persistent storage and many more. These tasks can be built around the Expression MIB, Event MIB, Schedule MIB and Script MIB, respectively.

2.2 Delegation of SNMP Operations

SNMP operations may be executed in three different locations according to its nature, complexity and network requirements (Figure 1): on the central manager, on the distributed manager, or on the agent. Each situation is non-exclusive, meaning that any combination of these scenarios is possible and sometimes it is even recommended.

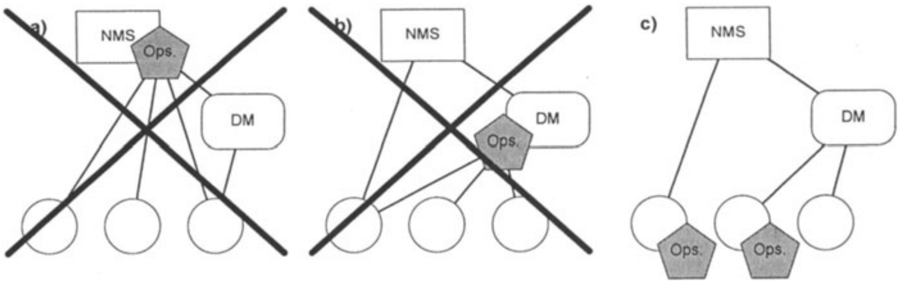


Figure 1. SNMP task execution (Ops.)
a) on the management station (NMS), b) on the distributed manager (DM), c) on the agent

Under DISMAN, each MIB encapsulates a set of operations around a specific concept. This fact allows using almost any combination of modules to achieve the above distribution models. However, some modules are more tightly related than others due to their type, i. e., some modules work better if associated with others which complement their capabilities. For example, the Script MIB cannot autonomously start scripts. This feature is obtained by associating it to the Schedule

MIB. On the other hand some associations are useless, such as the pair <Schedule MIB, Expression MIB>.

Considering local or remote information handling, some modules are able to interact with remote agents (ex. Script, Event) and others are restricted to the local agent (Expression, Schedule). This fact reduces the number of useful associations on real management scenarios:

- a single Schedule MIB cannot start scripts on different locations: it must be multiplied;
- an expression in the Expression MIB cannot obtain values beyond the local agent: a script could help although it is impossible to prevent an increase in complexity.

We have developed and evaluated the Schedule MIB and the Expression MIB, and we are currently performing some work around the Event MIB. The following section will focus mainly on the Expression MIB. It describes the implementation strategy and the evaluation of several aspects such as overhead, functionality and, generically, it points out the general advantages and disadvantages collected from the experience of placing this functionality inside the agent.

3. EXPRESSION MIB

We started working on the Expression MIB implementation when the documentation was still in the Internet draft condition[22]. Meanwhile, some minor details have changed both in the IETF documentation and in the implementation code, particularly the expression parser and the sampling mechanism. We included a more robust expression parser and changed some expression functions according to the clarifications made as the document evolved. We also improved the sampling mechanism to cope with the Event MIB requirements so that it could be used in both modules.

The MIB is divided into three main groups:

- *expResource* – this group is related to resource control, with particular emphasis on sampling parameters since this operation can have some impact on system resources.
- *expDefine* – is organized in three tables which collect information about the expression definition and about the errors occurred while evaluating it: a) *expExpressionTable*, defines the expression string, the result type as well as the sampling period. b) *expErrorTable* maintains a table of error registers gathering information such as: the last time an error occurred on evaluating the expression, the operation in which it occurred, the error type. c) *expObjectTable* controls each element characteristics inside the expression. The expression string may contain variables and each variable may have different sampling types and it may either be wildcarded or not.
- *expValue* – this group has a single table which instantiates the evaluation objects. It is by querying this table that the result of the expression is known.

The values used in the expressions may be absolute (the values of the MIB objects at the sampling time), delta (the difference from one sample value to the next) or changed (a boolean indicating whether or not the object changed its value since the last sample). In addition to sampling, the MIB also defines wildcarding, allowing the use of a single expression over multiple instances of the same MIB object. While regular objects are resolved by an SNMP get operation, wildcard objects are retrieved through the get-next operation. Users are familiar with wildcarding for referencing multiple files (such as the UNIX command "`cp foo.* /tmp`"). If there is more than one wildcard parameter in an expression they all must have the same OID termination (semantics) to maintain coherence in the result.

An expression result is retrieved by querying a row in the *expValueTable*. Each row has a single column, formatted according to the result type of the expression. The value is accessed by an OID containing the OID for the data type, the expression name and a fragment (Figure 2).

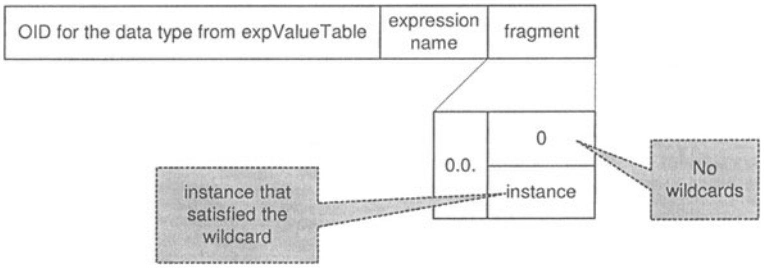

OID for the data type from expValueTable
expression name
fragment
0.0.
0
instance
No wildcards
instance that satisfied the wildcard

Figure 2. Value identification OID

The *expression name* has the form x."owner".y."name" converted to dot separated integers. The integer x is the length of the owner and y is the length of the string which identifies this expression to the particular owner. Each word character is also converted to a dot separated integers format according to the `SnmpAdminString` textual convention[23].

The *fragment* starts with "0.0." and it ends with a zero, when no wildcard is defined, or with the instance that satisfied the wildcard.

3.1 Implementation issues

The development of management agents is a complex and tedious task. To cope with these difficulties we have developed an open source, extensible API gathering all the agent common procedures – the Agent API[24]. The modular approach of this system allows (already developed) direct access to the agent through SNMP, RMI, CORBA, HTTP or WAP.

The Expression MIB[*] uses the Agent API services to provide the common SNMP mechanisms and to simplify and accelerate the agent development (Figure 3).

According to the expression properties defined in the tables *expObjectTable* and *expExpressionTable*, the Timer module wakes up at every delta interval and the Sampling module calculates the value according to the sampling type (`absolute`, `delta` or `changed`). This value is then forward to the Expression Parser.

[*] The implementation of the Expression MIB as well as implementations of the Schedule MIB and preliminary efforts on the Event MIB are available at http://nms.estig.ipb.pt/.

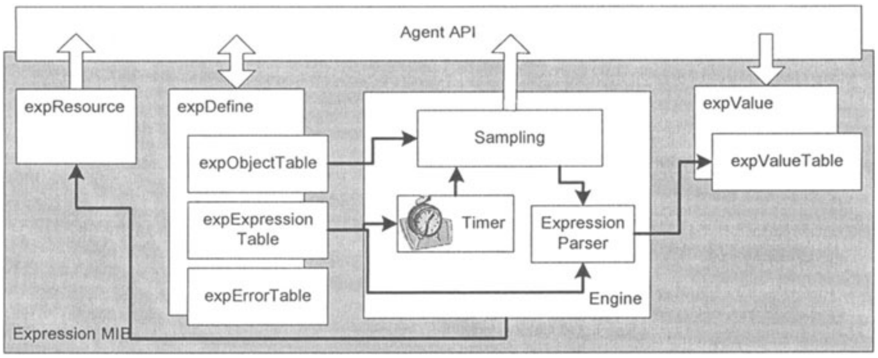


Figure 3. Architecture of the Expression MIB implementation

The Expression Parser module is responsible for evaluating the expression. To achieve this goal it must recognize the expression components (operators, functions, constants and variables), i.e. the lexicon and the grammar (the expression organization). There are, available as public domain software, lexical and grammar analysis tools, which generate code such as C[25] or Java[26, 27]. As this implementation is Java based, we choose Javacc[27]. This tool generates source code based on specification files which are then compiled (into Java .class files) and included in the Expression MIB agent.

The lexical analyser starts by reading the stream of characters and tries to match the sequences by identifying tokens. The tokens' information is forwarded to the grammar, which groups them into meaningful sequences and invokes action routines to act upon them. In this particular case, it must recognize a complete expression and evaluate the result.

3.2 Comments to the Expression MIB

An expression is composed of operators, functions and values. The values may be constants or variables, the latter being associated with OIDs that refer to the corresponding value. A string defines each expression.

The variables are defined in a separate table and indexed by a number of the '$v' form where 'v' is the variable number. For example, the expression '$1+$2+$3' represents the sum of the variables '$1', '$2' and '$3'. Variable indexes (the number prefixed by '$') correspond to entries in a table (*expObjectTable*) that contains an OID and the additional sampling parameters. Any expression can thus be defined according to the following generic format:

$$x = \text{Expression}(\text{oid}_1, \text{oid}_2, \ldots \text{oid}_n) \tag{1}$$

The Expression MIB retrieves the variable values from the local agent and evaluates the result (Figure 4).

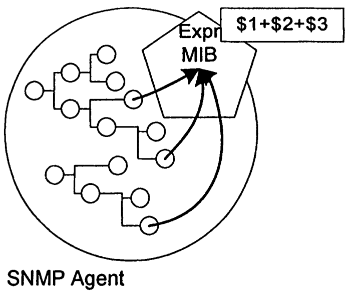


Figure 4. Expression MIB operation on local SNMP agents

The possibility of using variables in expressions is, simultaneously, the strength and the weakness of the Expression MIB. As currently proposed, the MIB does not allow retrieving values from remote agents restricting the expression evaluation to local objects. This limits the possibility of creating some expressions, for example, when they require values from different sources.

This scenario is modelled as weak distribution because the connectivity between DMs is non-existent, the delegation is occasional (restricted to expressions valid only in the local agent) and the number of DMs with the Expression MIB is usually low (it is more meaningful near raw management information, typical on the agent side). These facts limit the usage of the Expression MIB to a single situation – the agent side (Figure 5).

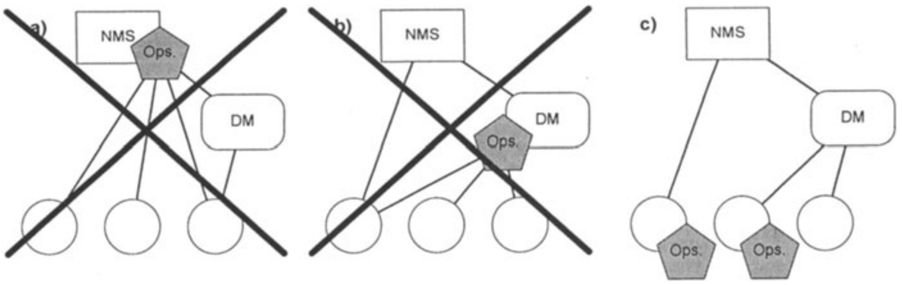


Figure 5. Expression MIB usage situations

Moreover, the integration of the Expression MIB module in existing SNMP agents is not possible unless the agent uses some extensibility feature, such as AgentX[28]. In this situation, the Expression MIB module could be the subagent and the existing SNMP agent would play the master role. Even in this case, some obstacles may persist because "Subagent access (via the master agent) to MIB variables" is a non-goal in AgentX.

The Event MIB, for instance, depends on the SNMP-TARGET-MIB to describe the remote hosts under monitoring. However, the same approach was not adopted for the Expression MIB, which eliminates the possibility to construct expressions over remote attributes. The use of "target-based" approach in the Event MIB caused some debate in the DISMAN workgroup because of the difficulty in storing credentials needed to contact remote hosts. However, it solves the problem (at least theoretically) by storing the security name and model in the SNMP-TARGET-MIB and by looking for the keys in the *usmUserTable* of the SNMP-USER-BASED-SM-MIB[29].

The same approach could be used in the Expression MIB to provide this kind of feature thus allowing it to gather parameters from remote MIBs. This change would allow defining expressions where variables are mapped to target tags and then resolved with the help of the SNMP-TARGET-MIB:

$$x = \text{Expression}(\text{target}_1, \text{oid}_2, \ldots \text{target}_n) \qquad (2)$$

In this case, the expression variables are defined as target tag references. The security credentials and profiles required for contacting remote agents are defined in the appropriate tables on the SNMP-TARGET-MIB.

To update the MIB with this functionality it would be necessary to modify the table responsible for designating the objects where parameters are obtained – *expObjectTable*. Like in the Event MIB, it would be necessary to introduce the following objects:

- *expObjectTargetTag* – specifies the remote system. Works together with the SNMP-TARGET-MIB.
- *expObjectContextName* – specifies the context used to get the parameter.
- *expObjectContextNameWildcard* – points out if the context name should be truncated for wildcards.

The main advantage of this approach is the compatibility with the current SNMPv3 framework. However, it requires an additional MIB module, the SNMP-TARGET-MIB.

A second approach to this problem is based on the use of SNMP URLs[30] in variable mapping instead of OIDs. We have previously proposed this concept as a way to globally identify network management information. The main idea is to use a single string to locate and configure network and systems parameters. Examples of this URL can be:

```
snmp://rlopes@sw1.estig.ipb.pt/sysContact/0?op=set&value=Rui?v3
snmp://guest@nms.estig.ipb.pt:161/sysUpTime?op=getNext?v3?router
```

Within the Expression MIB these specifications can be used to obtain values from remote agents thus increasing its flexibility and capability. It is then possible to use expressions like:

$$x = \text{Expression}(\text{url}_1, \text{oid}_2, \ldots \text{url}_n) \qquad (3)$$

This second solution implies replacing the OID column by a URL column, which allows changing only the column data type. It uses SNMP URLs to designate the target host, the object, context and other communication parameters in a single line of text. Contrarily to the first solution, this one will not require implementing a new MIB module. The only change to the Expression MIB occurs with the *expObjectID* column. Despite this modification is conceptually simple, considering the IETF standardization track, it implies the redesign of the MIB or even the proposal of a new one. The processing mechanism for each expression has to be further elaborated in order to cope with the URLs, but the increase in complexity is not too significant.

Regardless of the adopted solution, if the Expression MIB were allowed to obtain remote parameters it would be possible to obtain values from remote agents as well as remote DMs. Moreover, it could be associated to the network management station as well as other DMs and agents (Figure 6).

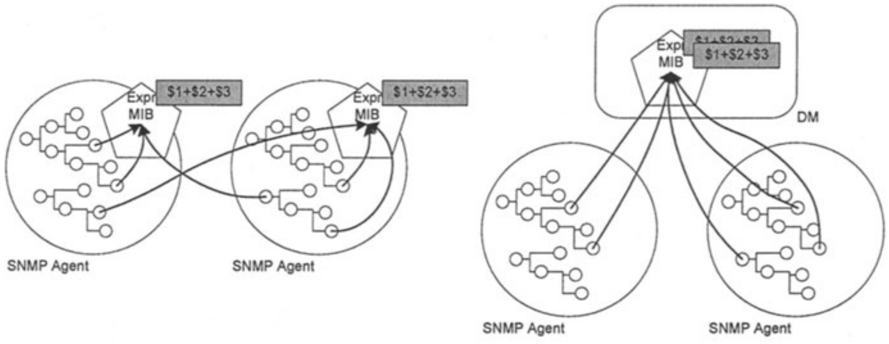


Figure 6. Modified Expression MIB operation

A bigger picture is shown in Figure 7, revealing the increase in connectivity among DMs and agents equipped with the Expression MIB. This scenario makes it possible to use variables from different sources within the same expression thus making the information correlation possible. Moreover, according to the set of expressions defined by the manager, it is now possible to build a strong distribution scenario.

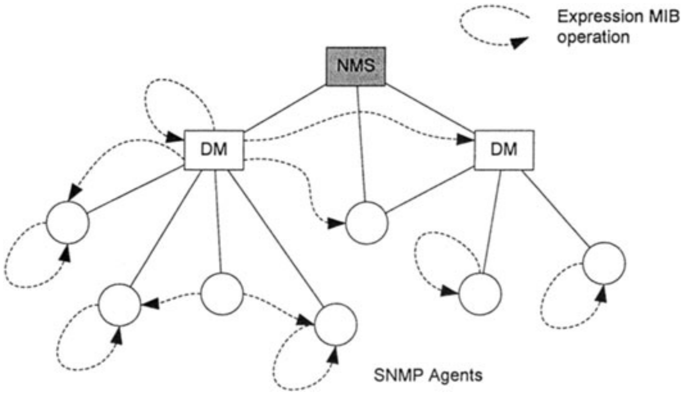


Figure 7. Strong distribution with the Expression MIB

These considerations may also be extended to other modules, such as the Schedule MIB. The current specification does not allow it to perform operations on remote modules, limiting its functionality to the local entity. This limitation may be eliminated by associating it with the Script MIB although some situations do not require such an elaborate tool. The extension of the Schedule MIB to allow remote operations does not seem to significantly increase the overall complexity and it would enlarge the usage possibilities because of the added flexibility.

Generally speaking, the DISMAN architecture can successfully distribute SNMP management operations because it is completely compatible and has a set of meaningful autonomous management operations. It should, however, allow the possibility for evaluation of expressions with remotely obtained parameters and to start remote periodic or calendar actions, which would increase the overall flexibility.

3.3 Changes to the Implementation

To cope with the previous recommendations it is necessary to perform some changes to some objects in the Expression MIB implementation. The modification of standard MIB structures is not a straightforward process. However and for research purposes, this can be done in a controlled and private environment without disrupting the standard and without affecting other management systems.

Besides the obvious type changes on the managed objects (which could imply the creation of a new object and the deprecation of the current one), it is necessary to change the sampling mechanism to allow retrieving values from remote locations. A pleasant side effect of this design choice is that the new sampling mechanism also works unchanged in the Event MIB.

We have chosen the SNMP URL approach to reduce the complexity and to minimize the table structure modification: only a single column type is modified.

There is however another important detail which cannot be left unnoticed: the expression functions. According to the specification, expressions can use a set of functions which work with a broad choice of value types, such as constants, variables, OIDs and others.

Will the SNMP URL approach dramatically change the implementation of functions?

For example, the function `sum(integerObject*)` may receive an OID or a variable referring to an OID, causing it to sum all the integer values of the wildcarded object. Will this definition suffer some modifications to cope with the above recommendations?

The answer is no. The resolution of the parameters happens before calling the function. So, the function `sum(OID)` will receive an array of integer values. The same happens to `sum($1)` or `sum(snmpURL)`, all resulting in an array of integer values to be summed. The result of the function remains unchanged regardless of the parameter type because the latter is resolved the function is called.

The main changes to the implementation happen at the sampling level. The suggested sampling mechanism is modular and relies on inheritance to provide different access and sampling methods. We should not forget that the sampling can be regular or wildcarded and absolute or delta or changed. Moreover, it can target a local or a remote peer. By chaining together the appropriate classes we will be able to build sampling mechanisms for any of these combinations as well as targeting different location hosts (Figure 8).

The Sampler uses Peer objects to provide the location dependent classes, namely the access to local or to remote agents. The value transformation (absolute, delta, changed) and access method (regular, wildcard) is provided by chaining Sampler classes. For example, the following code defines the access object to a remote wildcard MIB object with delta sampling:

```
Sampler sampler = new DeltaSampler(new WildcardSampler(remotePeer));
Sampler.sample();
Sample sample = sampler.getSample();
```

Further sampling examples may be built by using different classes and different constructor parameters.

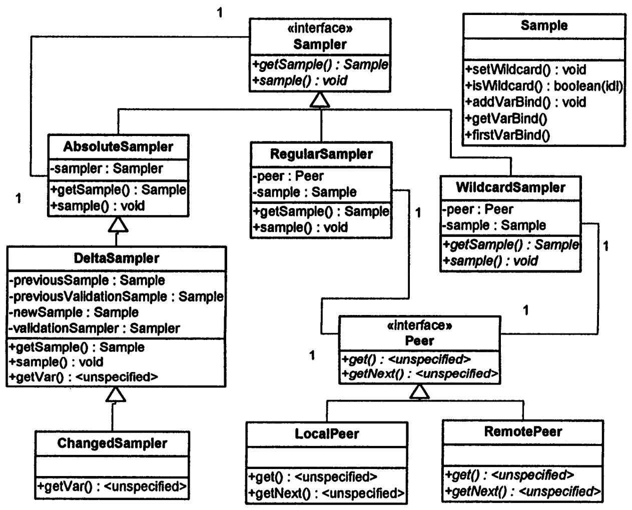


Figure 8. Sampling mechanism class diagram

4. CONCLUSIONS

Management distribution is a requirement to modern networks. As features appear and technology evolves, better tools are needed to maintain the network in excellent working condition.

The DISMAN workgroup have defined a rather complete set of MIB modules to ease the distribution of management tasks under the context of SNMP, which become compatible with the vast majority of installed systems. Among them, the Script MIB is the most studied and deployed module thus gathering a reasonable degree of knowledge around its features and applicability.

Mathematical expressions are fundamental to process and somehow filter the knowledge behind the evolution of network working parameters. The Expression

MIB is responsible for these tasks but, unfortunately, it is not allowed to use values from remote agents in the expressions.

In this paper we presented a possible solution to remote data retrieving by the Expression MIB. By performing some simple changes to the specification, this MIB can be extended to retrieve data from remote locations such as the Event MIB. Due to its importance the Expression MIB should be less restricted.

REFERENCES

[1] G. Goldszmidt, Y. Yemini, "Delegated Agents for Network Management", IEEE Communications Magazine, Vol. 36 No. 3, March 1998, pp. 66-71.

[2] A. Bieszczad, B. Pagurek, T. White, Mobile Agents for Network Management, Carleton University, Canada, 1997.

[3] V. Pham, A. Karmouch, "Mobile Software Agents: An Overview", IEEE Communications, Vol. 36, No. 7, July 1998, pp. 26-37.

[4] S. Krause, T. Magedanz, "Mobile Service Agents enabling Intelligence on Demand in Telecommunications", Proc. IEEE GLOBCOM'96, 1996.

[5] R. Lopes, J. Oliveira, "Software Agents in Network Management", proc. of the 1st International Conference on Enterprise Information Systems – ICEIS'99, March 1999, Setúbal, Portugal.

[6] DISMAN Charter (http://www.ietf.org/html.charters/disman-charter.html).

[7] J. Schoenwaelder, "Network Management by Delegation: from Research Prototypes towards Standards", Proc. 8th Joint European Networking Conference - JENC8, Edinburgh, Scotland, UK, May 1997.

[8] R. Kavasseri, B. Stewart, "Distributed Management Expression MIB", Internet Request for Comments 2982, October 2000.

[9] J. Martin-Flatin, S. Znaty, J. Hubaux, "A Survey of Distributed Network and Systems Management Paradigms", Technical Report SSC/1998/024, Swiss Federal Institute of Technology Lausanne, August 1998.

[10] Y. Yemini, G. Goldszmidt, S. Yemini, "Network Management by Delegation", I. Krishnan and W. Zimmer (Eds.), Proc. IFIP 2nd Int. Symposium on Integrated Network Management - ISINM'91, Washington, DC, USA, April 1991.

[11] Sun Microsystems, "Java™ Management Extensions Instrumentation and Agent Specification, v1.0", (http://www.javasoft.com/).

[12] SNMP Research (http://www.snmp.com).

[13] R. Kavasseri, B. Stewart, "Event MIB", Internet Request for Comments 2981, October 2000.

[14] R. Kavasseri, B. Stewart, "Notification Log MIB", Internet Request for Comments 3014, November 2000.

[15] K. White, "Definitions of Managed Objects for Remote Ping, Traceroute, and Lookup Operations", Internet Request for Comments 2925, September 2000.

[16] D. Levi, J. Schoenwaelder, "Definitions of Managed Objects for Scheduling Management Operations", Internet Request for Comments 3231, January 2002.

[17] D. Levi, J. Schoenwaelder, "Definitions of Managed Objects for the Delegation of Management Scripts", Internet Request for Comments 3165, August 2001.

[18] S. Chisholm, D. Romascanu, "Alarm MIB", draft-ietf-disman-alarm-mib-07.txt, June 2002.

[19] J. Schoenwalder, J. Quittek, C. Kappler, "Building Distributed Management Applications Using the IETF Script MIB", IEEE Journal on Selected Areas in Communications, Vol. 18, N° 5, pp. 702-714, IEEE Communications Society, May 2000.
[20] S. Waldbusser, "Remote Network Monitoring Management Information Base", Internet Request for Comments 1757, February 1995.
[21] S. Waldbusser, "Host Resources MIB", Internet Request for Comments 2790, March 2000.
[22] R. Lopes, J. Oliveira, "Distributed Management: Implementation issues", Proc. of the International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet - SSGRR 2000, August 2000, L'Aquila, Italy.
[23] D. Harrington, R. Presuhn, B. Wijnen, "An Architecture for Describing SNMP Management Frameworks", Internet Request for Comments 2571, April 1999.
[24] Agent API (http://nms.estig.ipb.pt/).
[25] T. Manson, D. Brown, lex & yacc, O'Reilly & Associates, 1990, ISBN 0-837175-49-8.
[26] A. Appel, A Modern Compiler Implementation in Java, Cambridge University Press, 1998, ISBN 0-521-58388-8.
[27] Javacc (http://www.webgain.com/products/java_cc/).
[28] M. Daniele, B. Wijnen, M. Ellison, Ed., D. Francisco. Ed., "Agent Extensibility (AgentX) Protocol Version 1", Internet Request for Comments 2741, January 2000.
[29] U. Blumenthal, B. Wijnen, "User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)", Internet Request for Comments 2574, April 1999.
[30] R. Lopes, J. Oliveira, "A Uniform Resource Identifier Scheme for SNMP", proc. of the 2002 IEEE Workshop on IP Operations & Management - IPOM 2002, Dallas, Texas, USA 2002.

WEAVER: REALIZING A SCALABLE MANAGEMENT PARADIGM ON COMMODITY ROUTERS

Koon-Seng Lim & Rolf Stadler
KTH Royal Institute of Technology
Stockholm

Abstract: While there is agreement on the drawbacks of centralized management, many approaches that address those do not scale well to large networks. We believe that effective management of future large-scale networks requires decentralized but coordinated control. In our recent work, we introduced the paradigm of pattern-based management, an approach that formalizes the use of graph traversal algorithms for controlling and coordinating lightweight agents that perform computations and data aggregation inside the network. We have shown analytically and through simulations that such a management system potentially scales to tens of millions of nodes, without significant performance problems regarding execution time and traffic overhead. In this paper, we report on a first implementation designed to realize the paradigm. Our system, *Weaver*, consists of active nodes constructed from small, low-cost Linux computers that are deployed onto a network of commodity routers. Management programs are written in C++ and can be validated and tested for performance on a simulator before being deployed. From the design of Weaver, we derive a simple performance model that allows us to predict the execution times of management operations on this platform. We evaluate the model through measurements on a laboratory testbed and demonstrate the efficiency of the platform. Finally, we use the model to predict the performance of a management operation running on a Weaver system for a large-scale network and thus show that our system is likely to meet the scaling potential of the paradigm.

Key words: Network management, scalability, management platform, active and programmable networks

1. INTRODUCTION

Over the last decade, the drawbacks of centralized management schemes have been recognized [2][3][22], and several approaches to distributing management tasks have been developed [9][32]. Interestingly, most of this research aimed at distributing the computations associated with a management task while keeping the overall control of a task centralized. In our recent work, we reached the conclusion that effective management of future large-scale networks requires decentralized management operations.

A significant step towards decentralized control has been made with the introduction of mobile agents for management tasks. Mobile agents can be characterized as self-contained programs that move in the network and act on behalf of a user (i.e., a human operator) or another entity. Mobile agents are generally complex, since they often need a degree of intelligent behavior for autonomous decision-making. Our approach can be understood as a variation of the mobile agent paradigm, where a management operation is realized through the coordinated actions of a swarm of lightweight mobile agents. However, in contrast to most mobile agent schemes to date, the agents in our approach are very simple; they exploit the parallel processing capability of the network, and, although they always carry state, they carry program code only when necessary

We call our approach to distributed management, which we have developed over the last two years, *pattern-based management* [16][18][19]. It centers around the concept of the navigation pattern, used for controlling and coordinating the actions of light-weight agents. Navigation patterns realize graph traversal algorithms that determine the dissemination of local management operations and the aggregation of the results of these local operations.

As our previous work shows, the approach of pattern-based management systems has interesting implications. First, management programs can be formally analyzed with respect to performance and scalability. The analysis of a management program is based on the analysis of the graph traversal algorithm of its underlying pattern. Second, navigation patterns allow the separation of the semantics of a management operation from the distributed programming aspects of the operation. From a software engineering perspective, this separation allows us to design generic patterns that can be combined with specific semantics to implement a particular management operation. A pattern, once designed, can be reused in the implementation of many management tasks. Conversely, a specific management task can potentially be built from a choice of patterns, which enables us to build management operations with different performance profiles. Ultimately, this approach frees an application programmer from developing distributed algorithms, allowing him/her to focus on the management task at hand, by selecting a navigation pattern from a catalogue that captures the requirements for that task.

Third, as our work on robust patterns indicates, the reaction to network faults, which can be complex to understand and handle, can be programmed into a pattern, thereby eliminating the need for the management application programmer to deal with faults. Finally, the degree of code mobility can be controlled in a fine-grained manner, since the execution of a management operation in a network involves distributing only those parts of the program that are not already resident in the network nodes. In other words, for a management program that is frequently executed, only the states of the distributed computation need to be exchanged between network nodes, not the code, which is locally available.

From the perspective of scalability, a pattern-based management system can eliminate bottlenecks associated with centralized processing of management data by distributing the load to network nodes via an appropriate navigation pattern. For example, the *Echo pattern* is particularly efficient at distributing and aggregating data over large networks [18]. By using an Echo pattern as a means for distributing computation to network nodes, highly scalable management programs can be implemented in compact form.

In order to support the development and study of pattern-based management programs, we have developed a PC-based discrete-event simulator, called SIMPSON [20]. SIMPSON is a C++ application that runs under Microsoft Windows (Win98, NT, 2000, XP) and is capable of simulating a large pattern-based management system of up to 60,000 nodes. Management programs on SIMPSON are written in C++ and compiled into dynamic libraries that are loaded on the fly for simulation. SIMPSON's interactive features allow the dynamics of a pattern to be visualized and recorded when its associated management program is executed. Performance data, such as completion time and volume of management traffic, can also be collected and analyzed.

In this paper, we describe the design and implementation of a pattern-based management system, called *Weaver*, on a network of commodity routers. Our design requires *neither modifications to the routers nor any special features on them.* The paper articulates and supports our belief that it is possible to build flexible and scalable network management systems at moderate cost, by using our paradigm.

This paper relates to previous publications on pattern-based management as follows. In [16], we introduced the idea of a management pattern and outlined an architecture supporting a pattern-based management system. We further reported on an explorative prototype, built to validate the very concept of management patterns. That prototype was based on Voyager, a commercial mobile agent platform, and was written in Java. The system presented in this paper is completely new and has a very different design focus, which is to demonstrate the feasibility of an efficient and scalable platform. In [18], we presented the Echo pattern and analyzed its performance with focus on large networks. In [19], we discussed a possible software design for management patterns and introduced SIMPSON, the simulator we had developed to test the functionality and estimate the performance of pattern programs.

In section 2 of this paper, we present the architecture of Weaver and discuss its operation as well as pertinent design issues. In section 3 we benchmark the performance of Weaver and develop a performance model for analyzing its behavior. In section 4 we use the performance model and the results from the previous section to evaluate its scalability through simulation. Finally, we conclude with a discussion of lessons learnt and future work. An appendix summarizes the aspects on management patterns needed to understand this paper, relieving the reader from accessing [16][18] and [19].

2. THE WEAVER PATTERN-BASED MANAGEMENT SYSTEM

Figure 1 shows the architecture of a pattern-based management system, first described in [16]. The gray nodes represent physical routers, while the white nodes, attached to them, represent the execution environments in which management programs run. The combination of a physical router and its associated logical execution environment constitute a logical network node. The execution of a pattern-based management program begins, when it is launched by a network management station onto the execution environment of the start node. When the program has completed executing on the node, its pattern determines the subsequent node (or nodes), on which the program must execute next. If that node already contains a copy of the program, only the program's state is transferred to it. Otherwise, both code and state are transferred. Alternatively, a node can download the program code directly from a secure repository called a code server.

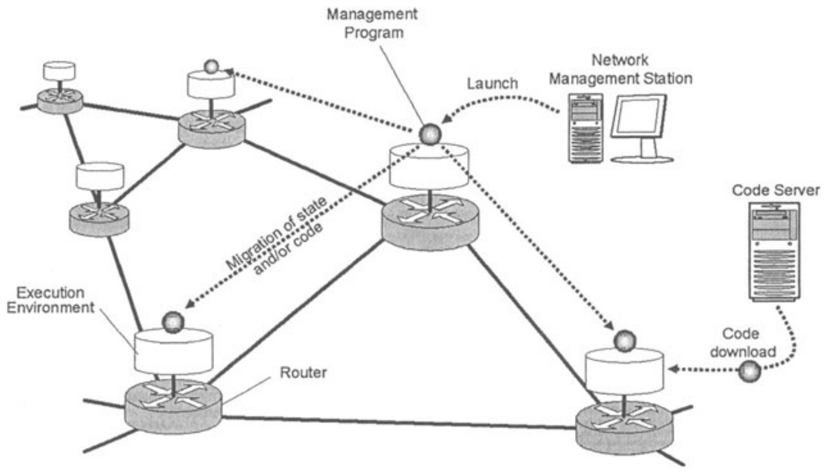


Figure 1. Architecture of a pattern-based management system

A pattern-based management system can be realized in a number of ways, for example, using a general purpose framework, such as Java, or an active networking toolkit, such as ANTS [31]. It can run internally on the processor of a router or externally on a device attached to the router, etc. However, realizing such a system at low cost on commodity routers restricts the design space and thus poses a significant challenge.

2.1 Weaver design aspects

Our most important design goals for Weaver were, first, to realize an *efficient* implementation of a pattern-based management system. By efficient, we mean that management programs complete execution quickly, even in large networks. Second, we wanted to realize a system that works with virtually all *commodity routers*.

In our system design, each router is managed by a dedicated active node that hosts the execution environment needed for running pattern programs. Such an active node, called a Weaver Active Node (WAN), is an internet-enabled, single-board computer, equipped with an Intel StrongARM 1110 microprocessor, 32MB of SDRAM, and a 10Mbps Ethernet interface. The hardware of the WAN is commercially available at a cost of an average PDA, in the form of an aluminum cube of 3 inches per side. To provide sufficient local storage, we attached a 1GB IBM Microdrive to the onboard compact flash slot. Each WAN runs a modified distribution of the Linux kernel (version 2.4.9), as well as an Apache web server, which is used to implement the WAN's management interface.

Code mobility was a further design issue in the development of our platform. Specifically, we had (1) to choose between transferring program code in either binary or source form, and (2) to decide, if code is to be transported by patterns or downloaded from code servers. The decisions made in addressing these questions have implications on the system's performance and its vulnerability to attacks.

For instance, if a program is to be transferred in source form, then a time-consuming compilation process must be invoked, before it can be executed. In addition, every node must be equipped with sufficient disk space to store the compiler, the linker, and the header files. The advantage of transferring source code is that source programs are significantly smaller than compiled programs.

On the second question, code servers can become bottlenecks, if the network becomes very large. Increasing the number of such servers introduces other problems, because keeping the code on all repositories up-to-date and consistent can be expensive.

In the current design of Weaver, programs are transferred in binary form by the pattern itself. We made this choice for performance reasons, as our tests indicate that simple management programs (for instance, programs based on type 1 to type 4 patterns, see appendix) take 10,000 times longer to compile than to run. Furthermore, compiled management programs in Weaver are generally less than 10 times larger than their source.

Figure 2 shows the software architecture of a WAN with its primary components. The first, the Active Node Manager (ANM), comprises an Apache SSL-enabled web server and a set of server-side PHP scripts [1]. The main functionality of the ANM is to offer a web interface to the management station for configuring and operating the node. The second component, the Active Node Daemon (AND), is a C++ application running as a background process. It implements the execution environment, which runs the pattern-based management programs on the node. In addition, there are several repositories (drawn as cylinders) with state information. For example, the node state repository holds the operational state of the node (such as the numbers and parameters of executing management programs), while the binaries repository serves as a cache of ready-to-run patterns and aggregators. When a pattern migrates to another node, it can leave local state variables in the local program state repository. Finally, the result repository provides for persistent storage of the results returned from a management operation.

2.2 Executing a pattern program on Weaver

In order to start a management program on Weaver, the management station downloads the source code of the pattern and aggregator, as well as the run-time parameters, via http onto a WAN, which will be the start node of this operation. The WAN's ANM then saves the code into its source repository and relays the source file names and the parameters to the preprocessor module through a local socket (Figure 2).

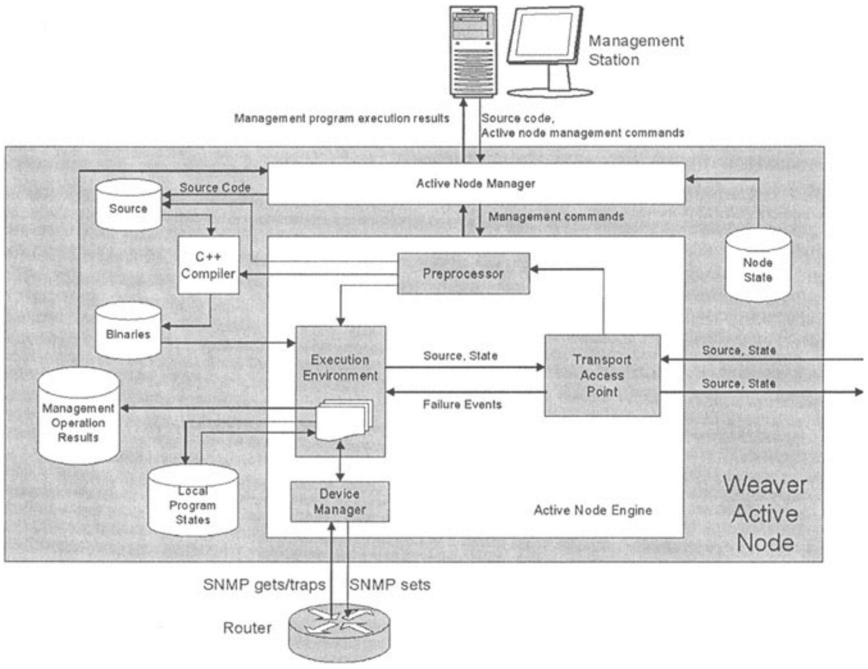


Figure 2. Software architecture of a Weaver Active Node (WAN)

The preprocessor module invokes the compiler to process the program source, computes the MD5 checksum of the resulting binary, and invokes the execution environment to run the program. If the compiler encounters an error, the execution is aborted, and an error code is returned to the management station via the ANM. If the program binaries are already in the WAN's repository, the preprocessor invokes the execution environment directly, passing to it the filenames and paths of the compiled binaries.

The execution environment dynamically loads the program and instantiates the pattern and the aggregator objects. It also generates a system-wide unique cookie, which associates the distributed state of the program with its current execution. Finally, it relinquishes control to the program, passing to it the arguments as specified by the management station.

A program accesses the management interface of the attached router through the WAN's device manager. In addition to the specific access protocol, the device manager also implements low-level monitoring procedures, such as heartbeats, to detect failures in the attached device. In principle, a WAN may include multiple device managers, one for each access protocol supported by a router. Our current prototype, though, has only a single device manager for SNMP.

When the management program has completed its execution on a node, it returns control back to the execution environment, along with a list of node addresses, to which to migrate next. After that, the execution environment stores the local program variables in the local state repository, serializes the mobile state variables, and passes them to the transport access point, along with the list of node addresses and the cookie.

The main function of the transport access point is to securely transfer program code and states between adjacent nodes. Whenever a WAN is initialized, it connects to its neighboring WANs by establishing secure channels.

When a transport access point receives a request to send program code and (mobile) state to a neighboring WAN, it first checks, whether the destination already has a copy of the program. If so, only the state of the program and the cookie are sent. Otherwise, the program code is sent, as well.

Every transport access point keeps a record of the nodes, to which it has sent programs, by saving their MD5 checksums. If no record exists for a particular node and a program checksum, then the program code is sent to the neighboring node, despite the chance that the neighbor might actually have a copy of the program. While this scheme incurs a (usually small) overhead, it is simple and requires no handshake.

When program code is sent, the receiving transport access point saves it into the local binary store, if necessary, and invokes the execution environment. Using the cookie, the execution environment determines, if the node has participated in the current execution of the program. If so, it checks for the program's local state variables, before passing control to the pattern and aggregator objects. Otherwise, new instances of the pattern and aggregator objects are created prior to program execution.

3. BENCHMARKING WEAVER

In this section we give a performance model for pattern-based management programs that are executed on the WAN architecture described in the previous section. The metric of interest in our model is the *execution time* of a management operation. It is measured as the time period from when a program is launched on a start node to when the results are returned to the management station.

We have conducted two series of experiments to obtain a delay profile for Weaver management programs. Table 1 and 2 show results from these experiments. The first series focuses on measuring the delay incurred by a management program based on the type 1 pattern, which models a simple polling operation, where control passes from a node to one of its neighbors before returning (see appendix). Other types of patterns (e.g. the type 2 and type 4 pattern, see appendix) can be expressed as serial compositions of type 1 patterns. In a similar manner, the type 3 pattern (see appendix) constitutes the basic building block for patterns, in which control is passed to neighboring nodes in parallel. The Echo pattern (see appendix), for instance, can be built from a type 3 pattern.

For accurate measurements, the experiments have been carried out on an isolated testbed of four Cisco 2621 routers, which are interconnected via a Cisco Catalyst 2900 fast Ethernet switch. Each router is equipped with two fast Ethernet ports, one of which is connected directly to a WAN. A 1.13 GHz DELL Inspiron 8100 notebook serves as the management station. Static routes have been set up from each router to the fast Ethernet switch, so that all nodes are able to communicate with each other.

In order to understand the delay profile of a management program that is based on a type 1 pattern, we decompose the execution of the program into a series of phases, listed in Table 1.

As we are only interested in measuring the delay from the point of view of the management program, the first phase begins when control is passed to the management program (T1). We call this the execution phase. When the program completes its execution, the serialization phase (T2) is invoked, in which the program's mobile state is serialized and then, during the dispatch phase (T3), sent to the remote WAN. The receiving phase (T4) begins, when the mobile state has been received on the remote node. Depending on whether the management program has been executed on this node before, the next phase can be either the loading phase (T5) or the instantiation phase (T6).

The loading phase occurs the first time a management program is executed on the node. Typical tasks performed include invoking the dynamic linker to load the program code as a shared library and instantiating the pattern and aggregator objects. Also, the program code, if received during T4, is saved in this phase. If the management program has already been loaded because of a previous execution, only the instantiation of the objects are performed. This is referred to as the instantiation phase.

If the program is still active on a node (i.e., the pattern will traverse the node again during its current execution), the pattern and aggregator objects are not deleted when the program migrates to another node. In this case, only a lookup will be needed to return their object references. We refer to this phase as the resolving phase (T8). Finally, the de-serialization phase (T7) creates (or recreates) the mobile state variables in the program's address space.

Table 1. Overhead incurred by each phase of execution of the type 1 pattern

	Duration in ms	Performed by Module
Execution (T1)	1.57 ($\sigma = 0.48$)	Execution Environment
Serialization (T2)	3.46 ($\sigma = 0.71$)	Execution Environment
Dispatch (T3)	1.67 ($\sigma = 0.49$)	Transport Access Point
Receiving (T4)	0.62 ($\sigma = 0.30$)	Transport Access Point
Loading (T5)	23.42 ($\sigma = 0.70$)	Execution Environment
Instantiation (T6)	0.77 ($\sigma = 0.015$)	Execution Environment
De-serialization (T7)	2.04 ($\sigma = 0.49$)	Execution Environment
Resolving (T8)	0.15 ($\sigma = 0.001$)	Execution Environment
Communications Delay (T_C)	4.04 ($\sigma = 0.10$)	---

Table 1 gives the mean and standard deviation of the delay for each of the phases (T1 through T8 and T_C), as measured over 40 runs. The communication delay on the last row of the table includes transmission delay, propagation delay and operating system overhead. The size of the mobile state is 207 bytes. The pattern program contains the minimal code necessary to implement the type 1 pattern and does not perform any other computations. The aggregator program contains only empty functions.

Based on the above discussion, we can derive the (average) completion time of a type 1 pattern as:

$$T_{type1} = 3T1 + 2(T2 + T3 + T4 + T7) + T6 + T8 + T_C$$

For a more detailed explanation of the above formula, see [21]. When the pattern is executed for the first time, an additional delay of T5-T6 incurs, because the execution environment needs to invoke the dynamic linker. Also, the estimate given by the above equation does not take into account the situation when the node daemon is swapped out by the operating system. Such instances appear rarely during our measurements, because of the small testbed and the light system load.

Following the above approach, we can derive similar expressions for the average completion times of management programs based on the type 2, 3, and 4 patterns. (See [21] for more details).

Table 2 compares the estimated completion time (based on the above formula and table 1) with the actual measurements on the testbed for all 4 basic pattern types. As can be seen, the estimations lie below the measured delays in all cases, with a margin of error between 8.3% and 10%.

Table 2. Comparison of estimated vs. actual measurements for the four basic patterns

	Average completion time (estimated)	Average completion time (measured on testbed)
Type 1	25.2 ms	27.6 ms
Type 2	72.6 ms	78.4 ms
Type 3	44.3 ms	47.6 ms
Type 4	49.5 ms	55.0 ms

4. EVALUATING THE SCALABILITY OF WEAVER

In this section, we investigate the scalability of the Weaver architecture when the network to be managed becomes large. Specifically, we estimate the completion times of pattern-based management programs on large networks using SIMPSON and the delay profiles shown in the previous sections. For comparison purposes, we also estimate the time of the same operation executed on a centralized SNMP-based management system, polling nodes serially or in parallel.

For simplicity, we assume the network topology to be a full *b*-ary tree with height *h*. Each node in the network is a router with *b*+1 ports, one of which is connected to a WAN. This way, each WAN manages exactly one router. We assume that the latency between two adjacent routers is identical to that experienced in our testbed (i.e. $T_{C1}=T_{C2}=0.5T_C=2.022$ms). We also assume that routes taken by packets are symmetrical; that is, PDUs of an SNMP request take the same path, from the manager to the managed device and vice versa. We assume that the management station is attached to the root of the tree and that all management program executions use the root as the start node. Finally, we choose the management task to be an operation that computes the average value of a specific MIB variable across all nodes in the network.

Given the above topology, the total number of nodes in the network, *N*, is therefore given by

$$N = \frac{n^{h+1}-1}{n-1}$$

The most scalable manner to implement the task in a centralized management scheme is to compute the network-wide average value of the desired variable incrementally from the values obtained from each node that is polled (via SNMP GET). If the polling is performed serially on N nodes (i.e., each GET operation must complete before the next GET is initiated), and if we neglect the time needed for the simple averaging computation, the total completion time is given by:

$$T_{centralized\_S} = (T_C + T_S) + (2T_C + T_S)n + \ldots + ((h+1)T_C + T_S)b^h$$

which evaluates to:

$$T_{centralized\_S} = \frac{(T_C(b+1)+T_S)b^{h+2} - (T_C(h+2)+T_S)b^{h+1} + T_C + T_S(1-b)}{(b-1)^2}$$

where T_S is the time required by the SNMP agent on a node to process a request. Our measurements on the Cisco 2621 routers puts this to be approximately 1.9 ms. On the other hand, if the polling is performed in parallel (i.e., the system does not wait for the completion of a GET before polling the next node) and nodes farther away are polled first before nearer nodes, the total completion time is given by:

$$T_{centralized\_P} = 2hT_C + (b^h - 1)T_P$$

where T_p is the polling interval between nodes. Our measurements on the WANs indicate that this is approximately 1.5ms.

In the case of a pattern-based management solution, we employ the Echo pattern described in section 2 to accomplish the same task.

We compare the performance of Weaver against centralized management using the serial polling scheme as a common yardstick. Specifically, we define the scalability measure S to be the ratio between the average completion time of a management task using the serial polling scheme and the scheme underlying the specific management task (such as parallel polling or Weaver/Echo). Figure 3 plots S against h, the number of levels in the tree network, (a) for parallel polling (dashed lines) and (b) Weaver/Echo (solid lines), for the two cases where b, the number of children of each node, is 2 and 6, respectively.

From the plot it is evident that parallel polling always outperforms serial polling, since its scalability measure S never falls below 1. Furthermore, for networks of small to moderate size (i.e., b=2, h<7 and also b=6, h<3), it also outperforms Weaver/Echo because of its lower overhead. However, for large networks, i.e., (b=6, h>4) Weaver/Echo yields completion times that are several orders of magnitude lower than the other schemes.

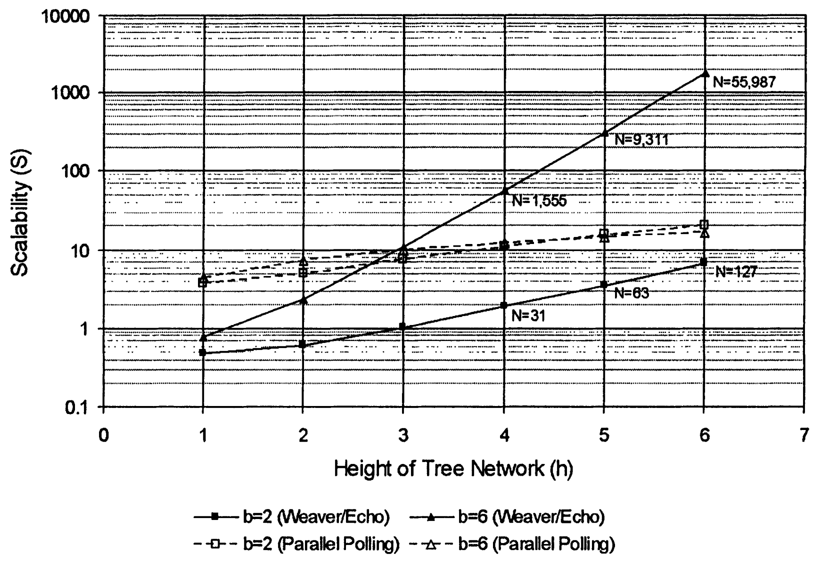


Figure 3. Scalability (*S*) versus Height (*h*) for parallel polling and Weaver/Echo. *N* indicates the number of network nodes.

5. DISCUSSION AND CONCLUSIONS

In this paper, we described a possible realization of the pattern-based management paradigm, using a network of low-cost, single-board computers that are attached to commercial routers. Our goal was to engineer a system that (a) would be lightweight, fast, and scalable, and (b) would be deployable on existing networks. The first goal dissuaded us from using a general-purpose mobile agent platform, which, while probably cutting down the development effort, would have provided unneeded flexibility at the cost of performance. The second goal necessitated the design of an execution environment external to a router, since the current generation of commodity routers does not permit the execution of user-supplied code internally. We chose a low-cost solution in terms of both hardware costs and space needed for the management devices inside the network. The choice of SNMP for router access also arose from our desire for a widely deployable solution.

Since every testbed limits the performance predictions of a system for different configurations, sizes, etc., we developed a performance model for a pattern-based operation on Weaver and used the testbed to instantiate and validate the model. Based on the measurements we took so far, we believe the model given in Section 3 to be basically sound. However, the accuracy of +/-10% for the predicted execution times of the type 1 to 4 patterns on our testbed tells us that our model must be refined. We are currently investigating, why the current model seems to systematically underestimate execution times, and why a large variance in certain sub operations occurs.

In Section 4, we gave evidence that pattern-based management operations on a large-scale Weaver system are likely to be very scalable. To prevent misunderstanding, we add two comments here. First, we measure scalability by comparing an Echo-based operation to a centralized operation. This means that we use the centralized management solution as a point of reference. Specifically, we are not implying that traditional or current solutions to managing very large networks are centralized in this manner. Second, one could probably come up with a traditional management system that is built on a hierarchy of SNMP agents and would exhibit similar scaling properties as shown in Figure 3. Note though that, while scalability is a key design goal of Weaver, its underlying paradigm of pattern-based management potentially has many advantages over traditional management systems. For instance, the communication and computation structure of a management operation is dynamically constructed by the pattern during its execution, which makes patterns independent of network topology and network size. Also, the discussion in Section 4 is based on the Echo pattern, which exhibits a specific hierarchical communication structure. Other patterns establish different communication structures during execution.

As with any system that includes mobile code, there is a danger of unsafe or malicious code being introduced. In its current implementation, Weaver addresses this issue as follows. First, all communication between the management station and a WAN occurs through an SSL-enabled web interface. This reduces the risk of unauthorized access and protects against masquerade attacks. Second, the compiled code of a management program is only executed within the context of a separate process, with restricted rights and resource quotas. This prevents management programs from interfering with one another or from crashing the daemon, should a fatal error occur. Finally, the communication channel between Transport Access Points of peer nodes is implemented using a simple TLS-like protocol [12], thereby preventing a third party from altering the program code or state while the data is in transit. In addition, all Linux services that are not needed to run Weaver have been disabled in our platform. While the above measures introduce security elements into the design of Weaver, a thorough study of this system's vulnerabilities and how to reduce them effectively still needs to be carried out.

There have been many efforts into building efficient management platforms by recent research into active and programmable networks. First, high-performance active network platforms have been used to implement management applications where management traffic is processed (close to) wire speed. Systems implemented with this philosophy typically have low-level, assembly-style instruction sets [14][23][29], optimized for space and speed. Unfortunately, developing management programs on such systems is difficult, due to their low-level nature, and thus limited to applications, such as programmable traffic probes. Furthermore, these platforms are built on customized or special network nodes that require features not available in commodity routers [10][11][17]. In order to apply some of these techniques to the management of traditional IP networks, (PC-based) software routers have to be used [5][14][17][23][25][29]. In these environments, the (operating system) kernel intercepts packets for processing in a management execution environment, usually through an IP option called Router Alert [15].

The second approach suggests that scalability can be achieved through mobile agent platforms that support intelligent preprocessing and data aggregation inside the network [3][6][22][24]. Many of the systems developed along this line emphasize flexibility over performance. They generally require the support of a heavyweight infrastructure, which is often Java-based [4][6][24].

Our current and planned work in pattern-based management follows several tracks. On the one hand, we are further exploring the potential of pattern-based management, by designing patterns for dynamic construction of network hierarchies and routing schemes, as well as investigating how basic patterns can be combined into complex ones with desirable properties. At the same time, we are accelerating our work on Weaver. We are currently extending our testbed from 4 to 16 nodes, which will help us in better evaluating and refining our performance model. Since Weaver is a decentralized system, initializing it is a non-trivial task, even in a medium-size network. We began working on a (pattern-based) scheme that would automatically configure Weaver on any network topology and dynamically integrate new WANs into the system. Finally, we have begun studying the use of patterns in policy-based management systems for disseminating policies in large networks and for dynamically re-computing policies, when triggered by state changes or network faults.

REFERENCES

1. S. Bakken, A. Aulbach, E. Schmid, J. Winstead, L. Wilson, R. Lerdorf, A. Zmievski and J. Ahto, PHP Manual, at http://www.php.net/manual/en.
2. M. Baldi, S. Gai and G. Picco, "Exploiting Code Mobility in Decentralized and Flexible Network Management," First International Workshop on Mobile Agents (MA'97), Berlin, Germany, April 1997, pp. 13-26.
3. M. Baldi, G. Picco, "Evaluating the Tradeoffs of Mobile Code Design Paradigms in Network Management Applications," First International Working Conference on Active Networks (IWAN'99), June/July 1999, Berlin, Germany.
4. C. Baumer and T. Magedanz, "The Grasshopper Mobile Agent Platform: Enabling Short-term Active Broadband Intelligent Network Implementation," First International Working Conference on Active Networks (IWAN'99), June/July 1999, Berlin, Germany.
5. S. Berson, B. Braden and L. Riciulli, "Introduction to ABone", June 15, 2000, available at http://www.isi.edu/abone/DOCUMENTS/ABarch/
6. A. Bieszczad, T. White and B. Pagurek, "Mobile Agents for Network Management," IEEE Communications Surveys, Vol. 1, No. 1, September 1998, pp. 2-9.
7. E. J. H. Chang, "Echo Algorithms: Depth Parallel Operations on General Graphs," IEEE Transactions on Software Engineering, Vol. 8, No. 4, pp. 391-401, July 1982.
8. J. Case, M. Fedor, M. Schoffstall and J. Davin, "A Simple Network Management Protocol (SNMP)," RFC 1157, IETF, May 1990.
9. R. Preshun (chair), Activities and Results of the IETF Working Group on Distributed Management (disman), http://www.ietf.org/html.charters/disman-charter.html.
10. D. Decasper and B. Plattner, "Dan: Distributed Code Caching for Active Networks," IEEE INFOCOM'98, San Francisco, California, March/April 1998, pp. 609-616.
11. D. Decasper, G. Parulkar, S. Choi, J. DeHart, T. Wolf and B. Plattner, "A Scalable High Performance Active Network Node," IEEE Network, Vol. 13, No. 1, January 1999, pp. 8-19.
12. T. Dierks and C. Allen, "The TLS protocol version 1.0", RFC2246, January 1999.

13. J. Gosling, B. Joy and G. Steele, The Java Language Specification, Addison-Wesley, 1996.
14. M. Hicks, J. Moore, D. Alexander, C. Gunter and S. Nettles, "PLANet: An Active Internetwork," INFOCOM'99, New York, New York, March 1999, pp.1124-1133.
15. D. Katz, "IP Router Alert Option," RFC 2113, IETF, February 1997.
16. R. Kawamura and R. Stadler: "A Middleware Architecture for Active Distributed Management of IP networks," IEEE/IFIP NOMS 2000, Honolulu, Hawaii, April 2000, pp. 291-304.
17. D. Larrabeiti, M. Calderon, A. Azcorra and M. Uruena, "A Practical Approach to Network-Based Processing," 4th International Workshop on Active Middleware Services (AMS'02), Edinburgh, U.K., July 2002.
18. K.-S. Lim and R. Stadler, "A Navigation Pattern for Scalable Internet Management," 7th IFIP/IEEE IM'01, Seattle, USA, May 2001, pp. 405-420.
19. K.-S. Lim and R. Stadler: "Developing pattern-based management programs," 4th IFIP/IEEE International Conference on Management of Multimedia and Network Services (MMNS'01), Chicago, Illinois, October/November 2001, pp. 345-358.
20. K.S. Lim, "SIMPSON—A simple pattern simulator for large networks," source code and documentation, http://www.comet.columbia.edu/adm/software.htm.
21. K.S. Lim, R. Stadler, "Weaver: Realizing a Scalable Management Paradigm on Commodity Routers," KTH/IMIT/LCN Technical Report Nr. 02-5021, August 2002.
22. A. Liotta, G. Knight, G. Pavlou, "On the Performance and Scalability of Decentralized Monitoring Using Mobile Agents," DSOM '99, Zurich, Switzerland, October 1999.
23. J. Moore, M. Hicks and S. Nettles, "Practical Programmable Packets," IEEE INFOCOM'01, Anchorage, Alaska, April 2001.
24. A. Puliafito and O. Tomarchio, "Using Mobile Agents to Implement Flexible Network Management Strategies," Computer Communications Journal, Vol. 23, No. 8, April 2000.
25. D. Raz and Y. Shavitt, "An Active Network Approach for Efficient Network Management," First International Working Conference on Active Networks (IWAN'99), June/July 1999, Berlin, Germany, pp. 220-231.
26. M. Rose, The Simple Book. New Jersey: Prentice Hall, 1994.
27. E. Rosen, A. Viswanathan and R. Callon, "Multiprotocol Label Switching Architecture," RFC 3031, IETF, March 1998.
28. A. Segall, "Distributed Network Protocols", IEEE Transactions on Information Theory, IT-29, pp. 23-35, 1983.
29. B. Schwartz, A. Jackson, W. Strayer, W. Zhou, R. Rockwell and C. Partridge, "Smart Packets: Applying Active Networks to Network Management," ACM Transactions on Computer Systems, Vol. 18, No. 1, February 2000, pp. 67-88.
30. G. Tel, Introduction to Distributed Algorithms, Cambridge University Press, 2nd Edition, 2000.
31. D. Wetherall, J. Guttag and D. Tennehouse, "ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols," First Workshop on Open Architectures and Network Programming (OPENARCH'98), San Francisco, California, March 1998.
32. Y. Yemini, G. Goldszmidt and S. Yemini, "Network Management by Delegation," IM'91, Washington, DC, April 1991, pp. 95-107.
33. E. Gamma, R. Helm, Ralph Johnson, and John Vlissides: Design Patterns—Elements of Reusable Object-Oriented Software, Addison Wesley, 1995.

APPENDIX: PATTERN-BASED MANAGEMENT

The pattern-based management paradigm is a distributed management approach based on the use of graph traversal algorithms to control and coordinate the processing and aggregation of management information inside the network. From the perspective of a network manager, the algorithms provide the means to 'diffuse' or spread the computational process over a large set of nodes. A key feature of the approach is its ability to separate this mechanism of diffusion and aggregation from the semantics of the management operation. The paradigm achieves this through the development of two important concepts; the navigation pattern and the aggregator. The former represents the generic graph traversal algorithms that implement distributed control while the latter implements the computations required to realize the task. A pattern-based management program includes both components.

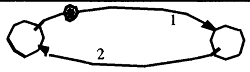
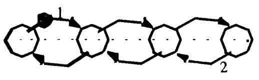
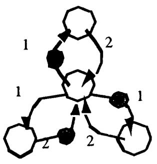
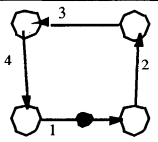
Pattern	Typical Application	Visualization
type 1: node-to-node	1 node control/monitor (e.g. get/set of variables)	
type 2: visit all nodes along a path/flow	1 flow/path control (e.g. traceroute, bottleneck detection, signalling, VPN operation)	
type 3: distribute agents to all nodes in subnet (parallel control)	subnet control, message broadcast (e.g. congestion location detection)	
type 4: visit all nodes in subnet (sequential control)	subnet control (e.g. topology detection)	

Figure 4. Examples of simple navigation patterns

Figure 4 presents the simplest examples of navigation patterns. The most basic pattern is the type 1 pattern, where control moves from one node to another and returns after triggering an operation. The manager-agent interaction is an example of this pattern. A type 2 pattern represents the scenario where control moves along a path in the network, triggers operations on the network nodes of this path, and returns to the originator node along the same path. A possible application of this pattern is resource reservation for a virtual path or an MPLS tunnel [27]. In a type 3 pattern, control migrates in parallel to neighboring nodes, triggers operations on these nodes, and returns with result variables. This pattern can be understood as a parallel version of the type 1 pattern. Finally, in a type 4 pattern, control moves along a circular path in the network. As these examples illustrate, navigation patterns can be defined independently of the management tasks performed in an operation.

For further clarification, we briefly discuss the *Echo pattern*, first introduced in [18]. It is an extension of the basic type 3 pattern and is based on a class of distributed graph traversal algorithms known as wave algorithms [7][28][30]. The behavior of the Echo pattern can be described as follows. The pattern starts out from a single start node, migrating to all its neighbors for further execution. This forward migration of the pattern from a node to each neighbor is called an explorer. An explorer, arriving on a node for the first time, marks the node as 'visited' and generates an explorer for each neighbor, except for the one from which it arrived (which is called its parent). Explorers arriving on a node that has been marked as 'visited', terminate at that node (i.e., they do create more explorers). If the node has no neighbors other than its parent, the program returns to its parent node. This return of the pattern from a node to its parent is called an echo. When a node has received an echo from each of its neighbors, it returns an echo to its parent. The Echo pattern terminates, when the start node has received an echo from each of its neighbors. Figure 5 shows the Echo pattern in pseudo code. It is a refined version of the code given in [19].

```
var $visited_i$ : boolean              init : false;
    $G_i$       : set of integers      init : neighbors();
    $parent_i$  : integer              init : -1;

    Echo(inmsg: bytes, from: integer) {

       $G_i$ := $G_i$ - from;
       If $visited_i$ = false {
          $parent_i$ := from;
          $visited_i$ := true;
          OnInitiate(inmsg, outmsg);
          if $G_i$ != empty {
             dispatch($G_i$, outmsg, i);
          }
       } else {
          OnAggregate(inmsg);
       }
       if $G_i$ = empty {
         OnComplete(outmsg);
         if $parent_i$ >= 0
           dispatch($parent_i$, outmsg, i);
         else OnTerminate(inmsg);
       }
    }
```

Figure 5. Echo pattern in pseudo code

Note that the concept of a navigation pattern is very different from that of a design pattern as used in software engineering [33]. While a navigation pattern captures the flow of control of executing a distributed operation, a design pattern describes communicating objects and classes for the purpose of solving a general design problem in a particular context.

ADAPTIVE RESOURCE MANAGEMENT OF A VIRTUAL CALL CENTER USING A PEER-TO-PEER APPROACH

Munir Cochinwala, Namon Jackson, Hyong Shim, and Eric Sigman
Applied Research, Telcordia Technologies Inc
445 South Street
Morristown, New Jersey 07960, USA
{munir, namon, hyongsop, erics}@research.telcordia.com

Abstract: As the number and diversity of end user environments increase, services should be able to dynamically adapt to available resources in a given environment. In this paper, we present the concepts of migratory services and peer-to-peer connections as the means of facilitating adaptive service and resource management in distributed and heterogeneous environments. Our approach has been realized using object-oriented principles in Adaptive Communicating Applications Platform (ACAP). The architectural design and implementation of a real-life high-level service, Virtual Call Center (VCC), are used to illustrate issues in adaptive service and management issues and discuss in detail our approach in ACAP.

Key words: Adaptive and Distributed Service and Resource Management, Peer-to-Peer Application, Service Platform, and Virtual Call Center

1. INTRODUCTION

With the increasing availability of network connectivity and network-enabled devices at work, at home, and on the road, users will require that services be adaptive to their environments and devices. Already, some popular services, such as email and instant messaging, are available on PCs, PDAs, and cell phones. However, the existing practice of creating multiple versions of the same service for execution on different devices is inefficient, unproductive, and wasteful of development and management resources. Ideally, services should be built once and be able to dynamically adapt to available resources and capabilities in diverse environments. Thus a mechanism is needed that allows service providers to effectively deliver and manage services in different environments

In this paper, we describe a peer-to-peer approach for managing end user environments and resources in a scalable and flexible manner. An environment mainly refers to a computing/communication platform from which the user accesses their services, and a resource refers to a specific hardware device or application tool used in a service. In our approach, each environment manages its own resources and is aware of their capabilities, and services negotiate with a given environment for required resources at runtime. An environment could be a home network, an ISP or a provider network. The approach is peer-to-peer in that once a service is activated in an environment, service endpoints communicate with the environment for resource requests. Subsequently, service endpoints share data and resources directly among themselves. Any accounting and state update information at the end of a service session is relayed to the service provider on an as-needed basis.

Since each environment is self-managed, the load increase on service providers needed to support new environments is incremental compared to a centralized approach, in which service providers manage all the environments and resources themselves. This helps increase the scalability of services and allows for rapid introduction and management of new services Our approach is also adaptive in that resource availability can be locally maintained and environment management can be tailored to the specific characteristics and requirements of individual environments.

Our approach has been incorporated in a prototype platform, called Adaptive Communicating Applications Platform (ACAP) [1]. ACAP takes an object-oriented approach to managing services and resources. Specifically, in ACAP, services specify resource requirements, and environments provide resources that match the requirements--multiple resources may match specific requirements. Services dynamically execute in diverse environments by adapting to available resources by migrating and requesting available resources from environments at runtime. Migratory services are facilitated by use of migratory objects.

ACAP is a working prototype and has been used to develop a number of applications to show the viability of our approach to service and resource management. In this paper, we present one such application, Virtual Call Center (VCC), to illustrate service and resource management issues in distributed and heterogeneous environments. We also describe in detail how ACAP is used for the development of the VCC application. Note that ACAP is a general-purpose platform that can be used for any application that requires integrated service and resource management in distributed environments.

The rest of the paper is organized as follows. Section 2 describes VCC and its service and resource management issues. Section 3 provides an overview of ACAP. Section 4 describe and discuss in detail the use of ACAP for VCC. Section 5 discusses related work of ACAP with emphasis on industry efforts on service management. Section 6 describes future work and concludes the paper.

2. VIRTUAL CALL CENTER: A MOTIVATING EXAMPLE

A virtual call center (VCC) consists of operators who are geographically distributed, and on duty at different times. When a customer calls, VCC's call processing/scheduling engine determines an operator and routes the call to the operator. In this paper, we show the call processing/scheduling engine as a centralized dispatcher. Dispatching functionality can be distributed across environments or implemented in distributed databases such as in 800 call routing [10]. Either of these mechanisms could easily be incorporated into our approach.

One critical issue in increasing the effectiveness of the VCC (or a regular call center) is state transfer. A typical user experience with a call center is that the customer often has to repeat relevant information while being transferred from one operator to another. This problem is even more prevalent with the advent of the Web, where users often perform a lot of "work" before calling the VCC, e.g., putting potential purchases into online shopping carts, filling out electronic order forms, and registering with merchant sites.

VCC can allow operators to effectively determine what callers are trying to do without requiring further, duplicate information by seamlessly transferring caller registration, history, state, and other information. This dramatically enhances the quality of user experience and increases the likelihood of turning users into customers. Furthermore, a centralized approach of having server components always manage state transfer and update transactions would not scale to a large number of callers and any server failure may disrupt the services of all callers, which would significantly degrade the effectiveness of the overall system. Thus, a distributed approach is needed that efficiently allows for scalable, robust, and fast state transfer. In this paper, we describe a peer-to-peer approach, in which server components "get out of the way" once an operator is connected to a caller, and the operator is responsible for updating the state information of the caller at the VCC once the call is complete or on an as-needed basis.

Another issue involves the means by which VCC operators and callers communicate and exchange information. Different callers may have different preferences for how to communicate with the operator. Some may prefer text-based instant messages, while others may prefer more interactive, real-time communications. In the latter case, the caller preference may range from a telephone/cell phone to a VoIP call to a multimedia call that includes voice, video, and documents. In order to accommodate diverse caller preferences, the communication mechanism used by the VCC should be adaptive to end user environments and available resources. At the same time, it is unreasonable to expect that the VCC would/could pre-provision all the computing/network environments of distributed operators to meet all caller requirements; doing so would be prohibitively expensive. Therefore, an efficient mechanism is needed to dynamically discover and adapt to available resources and their capabilities in diverse operator environments. We use the concept of migratory services to address this issue. Specifically, calls are represented as services that can dynamically move to operator and caller environments and execute using locally available or preferred resources.

3. ACAP

Adaptive Communicating Applications Platform (ACAP) is an object-oriented service platform that provides support for developing and managing high-level services in a distributed and peer-to-peer manner. In this section, we provide a high-level overview of its basic constructs: services and environments. In Section 4, we describe in detail how the VCC system is built using these constructs.

The main objective of ACAP is to provide support for adaptive services. An adaptive service can change the way it operates, depending on available resources in environments. To illustrate, a two-party voice call service can be made adaptive by facilitating the parties involved to use the communication devices of their choice, e.g., a regular phone for the caller and a PC-based phone for the called party. In ACAP, the concept of a migratory service is the main means by which adaptive services are realized. The basic mode of operation in ACAP is that a service endpoint first moves to an environment, negotiates for the available resources in the environment, and then executes using the negotiated resources. This way, services adapt to environments in a distributed manner, in which much processing occurs at or near endpoints. This reduces processing overhead on servers, which, in turn, helps increase the scalability of services. To facilitate migratory services, ACAP uses migratory objects. Specifically, a service is modeled as a collection of service objects that can move based on user request and system resource availability. A service object implements service logic, maintains service state, and specifies a resource requirement for the service. A service is a special service object that also functions as a container of other service objects. A service may be contained in other service containers, thus forming a hierarchy of services. Henceforth, the terms, service and service object, are used interchangeably, unless a distinction needs to be made.

Services and their service objects always maintain their containment relationships, even when they are at different locations. When a service moves, only those objects that are co-located with the service also move. In ACAP, we use remote references to represent inter-object relationships. A remote reference is similar to an ordinary object reference, except that (1) when a remote reference is serialized, it does not serialize the referee, and (2) when a serialized remote reference is recreated, it points to the original referee, even if the recreation is on a different machine. Remote references are used for passing objects in (possibly remote) method invocations and also allow migratory service objects to maintain their links to other service objects.

In ACAP, environments are modeled in terms of available resources and their capabilities. For example, a PSTN phone is modeled as a resource capable of audio communication with a certain bandwidth requirement. An environment is modeled as a resource, which in turn, is a collection of other resource objects. In this way, the entire environment, including the network, can be modeled hierarchically. It also allows us to efficiently compute and manage the state of an environment and by induction, the state of any service or system.

Each environment can independently administer its own resources. Policies may be used for security and/or enforcing user preferences. This allows us to efficiently and independently represent and introduce new resources into the system without central administration and provisioning. Rapid introduction of new resources requires that new devices and services are deployable by multiple third party providers.

When a service enters an environment, ACAP binds each service object with a resource object that can meet its requirements. For example, a call service object may require resources for inputting and outputting audio, for which ACAP may bind a resource object that represents a regular phone, an IP phone or a PC audio system to the call service object. The decision as to which resource objects are bound to service objects depends on resource availability in a given environment and user preferences.

In our current implementation, resource requirements are hard requirements. If no resource is available, then the service will not be able to execute. We already have the notion of service adaptation to equivalent resources such as different types of phones. We are designing mechanisms for services to adapt to degraded resources such as insufficient bandwidth for video streaming. We can easily adapt to degraded resources using service specific logic such as the designer of the service replacing streaming with periodic still images. We are currently exploring generalized mechanisms for resource degradation (non-availability is the worst kind of degradation).

4. ACAP AND VCC

VCC makes use of ACAP support for adaptive services to provide scalable and adaptive call center services. Specifically, VCC models a call as an adaptive service that can dynamically move to the environment of the VCC operator who gets assigned to the call. By executing the received call, the operator can retrieve registration, history, current state, and any other relevant information of the caller and establish a peer-to-peer communication channel with the caller. In fact, a call is a container object that has other objects with caller information and VCC call service logic. In this section, we describe and discuss in detail use of ACAP in the development of the VCC application.

4.1 Architecture

Figure 1 graphically shows an architectural overview of the VCC in relation to ACAP. VCC DISPATCHER is the central administrative hub for the system. It maintains a database of customer registration information, operator information (e.g., their schedules, locations, and state), and other system resources. It also handles all the incoming calls and assigns them to appropriate and available operators, the process of which is described shortly.

In Figure 1, VCC OPERATOR is an ACAP-based tool that implements the VCC application logic for VCC operators. It provides a notification service that alerts the operator. Furthermore, it can interact with the tools and resources that are available

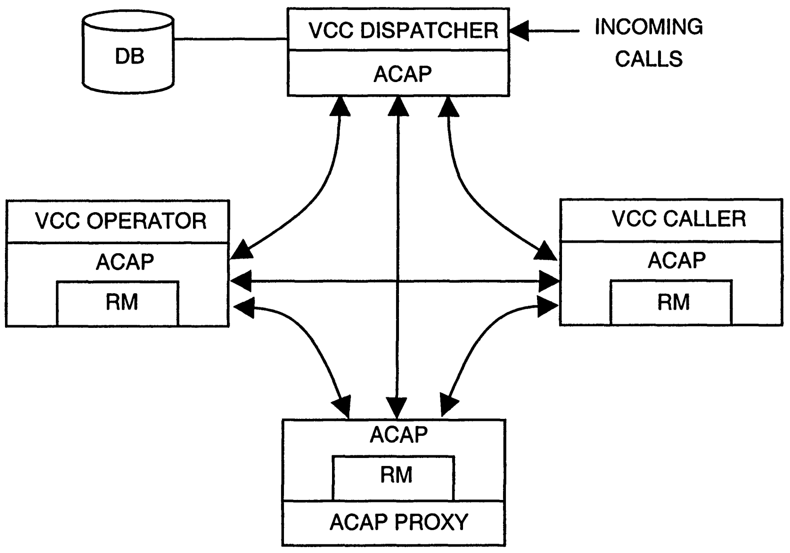


Figure 1. Architectural Overview of the VCC Using ACAP

in the operator's environment via the ACAP Resource Manager (RM), which is described shortly.

VCC CALLER is an ACAP-based tool that enables the caller to interface with the assigned operator. It establishes a peer-to-peer communication channel with the VCC OPERATOR of the operator and creates collaboration/communication sessions between the caller and operator on an as-needed basis. Currently, it is assumed that the VCC CALLER is pre-installed and configured in callers' environments, say, as part of creating a subscription with a service provider. However, it could also be provided as an applet that can dynamically be downloaded via a Web browser and installed and configured for the caller's environment when a call is made.

Callers may use regular or cell phones to contact the VCC. In such a case, the VCC OPERATOR connects to the ACAP PROXY to establish a voice channel between the caller's phone and the operator's device for voice communication. ACAP PROXY is a logical entity that may include PSTN/VoIP gateways to allow use of SIP phones or VoIP application tools. ACAP PROXY keeps track of the capacity and current usage state of its PSTN/VoIP gateways. VCC may have multiple instances of ACAP PROXY, in which case the VCC DISPATCHER may dynamically determine a particular instance to be used based on the caller phone number and the location of the assigned operator.

In Figure 1, the flow is from a caller to the dispatcher, which migrates the call to the appropriate operator so that the operator can communicate directly with the caller. Resources are allocated to the call once it migrates to the environment of the assigned operator. In the next sub-sections, details of interactions among various components are described.

4.2 ACAP Resource Manager (RM)

In ACAP, Resource Manager (RM) manages devices, application tools, and other resources that are available in a given environment. Typically, RM runs in the same environment as the one whose resources it manages (as is the case for the VCC system). However, if infeasible to do so, RM can also run at a remote location.

Resources are associated with a hierarchical, ACAP-defined type system. Figure 2 shows a partial resource type hierarchy and example resource instances for a typical VCC operator environment based on a networked desktop computer. Each resource type has a set of attributes that describe capabilities and other properties. For example, the resource.comm.audio.ip type represents resources for VoIP communication and has properties, in and out, which represent audio input and output resources/devices respectively. The same resource type hierarchy is used by ACAP services to specify required resources.

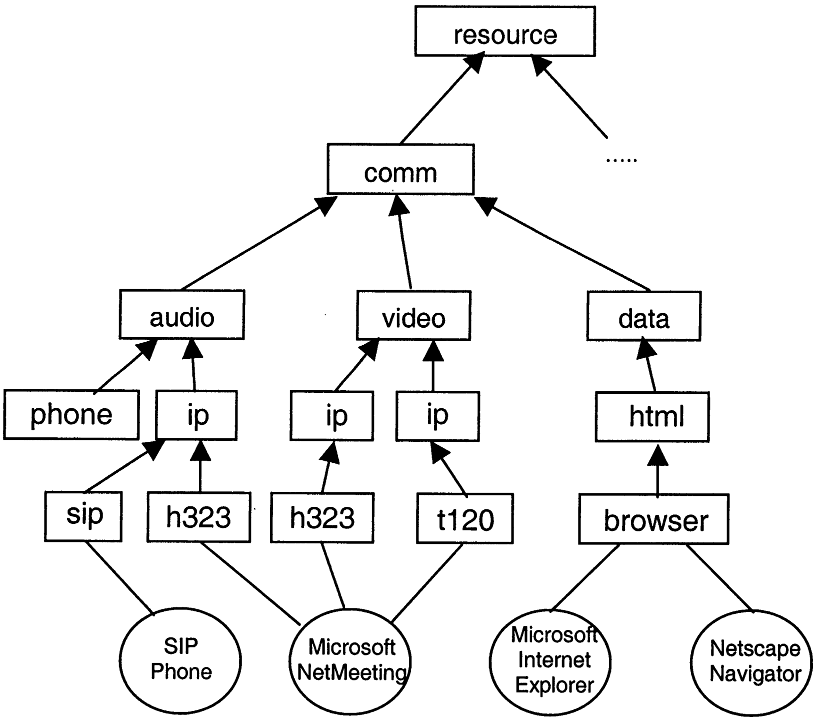


Figure 2. A Partial Resource Type Hierarchy and Example Resource Instances for a VCC operator environment. A resource type is a rectangular box, and a resource instance is a circular box.

Part of the VCC OPERATOR/CALLER installation and configuration process is to construct a resource hierarchy for an environment, most of which can automatically be deduced from the local (file type ←→ tool) association settings. Multiple instances may exist for a given resource type, e.g., resource.comm.audio.ip and resource.comm.data.html.browser in Figure 2, in which case the operator/caller is prompted to specify preferences.

When a service enters an environment, each of its contained service objects makes a request to the RM for a required resource. If the requested resource is available, the RM returns the corresponding resource object. The resource object is a proxy for the real device or application tool and provides an interface through which the service object can control, communicate, and manage the device or application tool. Communication between a resource object and the corresponding device or application tool is device or application-dependent. For example, the resource object for Microsoft NetMeeting may use a COM interface of this tool that allows other applications to control it programmatically. This way, existing tools and devices can easily be integrated with an ACAP environment, which in turn helps reduce the "learning curve" both for VCC operators and callers.

4.3 Handling Incoming Calls

Upon receiving a call, VCC DISPATCHER performs the following tasks:

- Retrieve the caller history (based on caller id), current state (e.g., URL history), address, and preferred method of communication of the caller.
- Create a CALL service object, which is a topmost container that represents the current call and contains USER, CONTEXT, and COMM service objects. The USER object stores any registration information of the caller, the CONTEXT object any received state information, and the COMM object the caller's current location and preferred methods of communication in terms of resource types. Each of these service objects has an application-specific interface.
- Create and store a copy of the CALL object for record keeping and persistence. This is the primary copy and holds the true state of the call if the information is lost in transit.
- Select an operator and put the CALL object in the QUEUE of the selected operator. QUEUE represents a communication channel between the VCC DISPATCHER and VCC OPERATOR of the selected operator. The exact operator scheduling and selection criteria may be policy-dependent and are beyond the scope of this paper.

When the CALL object arrives at the operator site, ACAP notifies the operator-end of the QUEUE, which, in turn, notifies the VCC OPERATOR. VCC OPERATOR starts or activates the new CALL and alerts the operator of its arrival. When the operator wishes to communicate with the caller, the VCC OPERATOR makes a request to the COMM service object, which first asks the ACAP RM for a communication resource of the same type as that of the caller's top preference. When the RM returns a resource object (see Section 4.2), the COMM creates two

ACAP Endpoint objects. Each Endpoint is a service object that represents an end point of a "call" of a specific type and is to be bound to a resource object for a device or application tool used for the call. Binding an Endpoint to a resource object may involve starting an application tool and retrieving its address information, e.g., the IP address and port of the host computer, on which the application tool is running. In ACAP, this address information is called the resource address of the Endpoint. Subsequently, the COMM asks the RM to bind the operator Endpoint to the returned resource object and then sends the other Endpoint to the caller.

On the caller's side, the VCC CALLER receives and starts the caller Endpoint object, at which point the Endpoint asks the RM in the caller's environment for a communication resource of its type. When the RM returns a resource object, the Endpoint object binds to it and alerts its counterpart at the operator's site of its resource address information. Subsequently, the operator's Endpoint makes a request to its resource object to initiate a communication session, passing it the resource address of the caller Endpoint. In turn, the resource object instructs its application tool or device to "call" the specified destination.

Figure 3 graphically illustrates the architecture of an example communication session between the VCC operator and caller, the set-up process of which is just described. This architecture applies to all types of communication in ACAP. The operator and caller Endpoints may continue to exchange information even after a communication session has been established. For example, in a collaborative Web Browsing session, the operator Endpoint may send to the caller Endpoint the URLs of the Web pages that the caller's Web browser should display and vice versa. For "multimedia" sessions, ACAP allows multiple pairs of operator ←→ caller Endpoints to exist and operate at the same time.

Note in Figure 3 that the resource objects do not have to run on the same host as their application tools or devices. This design is critical in increasing ACAP's ability to adapt to the preferred or available resources. To illustrate, consider a case where the operator and caller wish to have a voice communication, and one or both of the parties wish to use a regular phone. Here, the resource object cannot control the phone directly. Instead, it interfaces with a gateway to the phone network, which most likely is located at a remote site, and control the phone via this gateway. Furthermore, the operator/caller may be on a device with limited resources or capability, e.g., a phone. ACAP can still accommodate such a case by running the resource and Endpoint objects on a proxy host. Also note in Figure 3 that the COMM object on the operator site still contains the caller Endpoint object, even after it has moved to the caller's site. This way, the COMM object can still maintain context across contained objects in presence of the mobility.

When the call is complete, the VCC OPERATOR requests that the CALL release all the allocated resources. Furthermore, it notifies the VCC DISPATCHER of the call completion and sends it any state updates by sending the CALL and its contained objects back. Subsequently, the VCC DISPATCHER updates its database with any updated information from the received CALL. It may be possible that the connection between the VCC OPERATOR and DISPATCHER may fail while the operator and caller communicate. In such a case, the VCC OPERATOR locally stores the CALL and its contained objects until the connection is restored. The state of the call is contained within the object and can be updated in the future, even in batch mode, if desired.

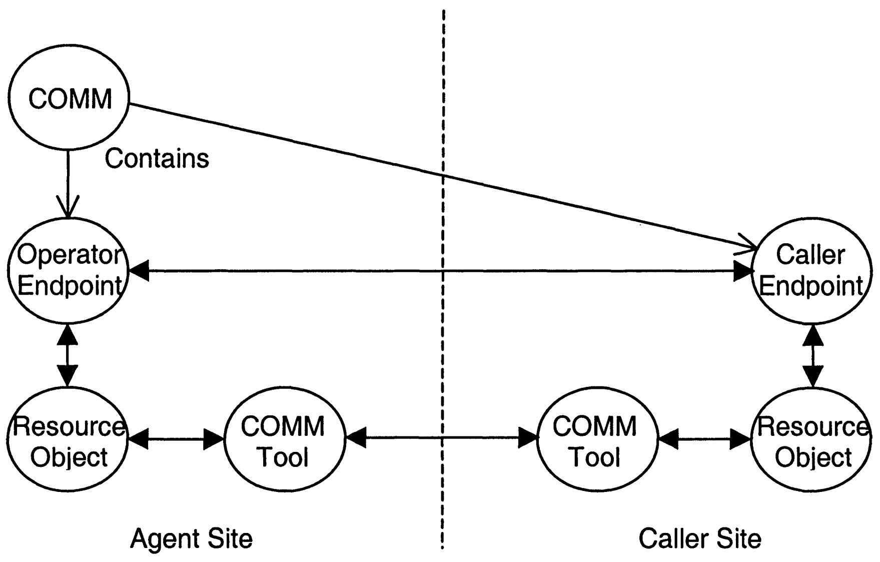


Figure 3. Example Architecture for Operator and Caller Communication

5. RELATED WORK

To our knowledge, ACAP is the first adaptive services platform that uses the concept of migratory services and peer-to-peer connections as the main means of developing and managing high-level services. However, the idea of adapting services to available resources in end user environments has been receiving an increasing amount of attention. For example, Open Services Gateway Initiative (OSGi) [2] is an industry consortium that specifies an object-oriented framework for remotely delivering and managing services. The OSGi framework provides a common life-cycle management service and allows services from different vendors not only to co-exist in the same environment but also dynamically discover and make use of each other. In addition, it allows a variety of end user devices to be represented and integrated with other services.

Note that the OSGi framework does not specify the mechanics of service operation, whereas ACAP specifies a model of service operation using migratory objects. This makes ACAP and OSGi complimentary to each other. For example, ACAP can use the service/device management facilities of the OSGi framework in implementing its Virtual Call Center (VCC) application as follows. ACAP Resource Manager can be implemented as an OSGi service bundle. This bundle can be downloaded and installed on the OSGi framework of a call center operator, at which the Resource Manager, once activated, can discover communication and other resources of the operator's environment. The ACAP call service can also be implemented as an OSGi service bundle, which can dynamically be downloaded and activated and discover the local Resource Manager. Integration with the OSGi framework is part of the future work.

Telecommunications Information Networking Architecture (TINA) [3] is an example of high-level service deployment and management platform that takes a client-server approach. Mainly developed from the viewpoint of service providers, TINA specifies a set of architectural principles and object-oriented information models for next generation multimedia telephony services. TINA takes a client-server view of how services should be provided in that every aspect of managing a service involves server components "running in the network." For example, in TINA, creating end-to-end communication channels involves a Communication Session Manager (CSM), which is mainly responsible for collecting and distributing the address information of end user devices to be used.

In contrast, ACAP is inherently peer-to-peer. In ACAP, the basic model of communication is to create and send peer service objects to communicating endpoints. Once in place, these objects connect to each other directly. This way, much processing of the application logic of a service can take place directly in end user environments, and server components mostly perform administrative tasks such as billing and subscriber management. Thus we argue that ACAP provides better support for scalable service environments than TINA.

Object mobility has received a lot of attention in the literature. One of the main applications of object mobility has been in localizing access to distributed objects in order to increase system performance. The main idea is to move an object to where it is needed, which makes accessing the object not much more expensive than local method calls. In order to automate the decision as to when and where to move objects, a variety of system- or language-level support is provided. For example, Kan [4] extends the Java language to include Kan-specific keywords and constructs for asynchronous method calls and transactions and keeps track of read/write access patterns on Kan objects. When a Kan object is write-accessed one thread at a time, it migrates. Otherwise, the object is replicated to where read requests are issued, and write requests are sent to its home site. StratOSphere [5] allows migratory objects to adapt to the available resources of new sites by enabling them to adopt the local implementations of their methods. The Aleph toolkit system [6] facilitates efficient location of migratory objects via its Arrow distributed directory protocol. Migratory objects, in the form of mobile operators, have also been used in the area of active messaging, e.g. [7].

In ACAP, migratory objects do not contain any "intelligence." Rather, they are used in designing and managing migratory and adaptive services in a distributed environment. Specifically, they are mainly used as the means of transporting application code and data to end user environments and establishing peer-to-peer connections between communicating endpoints.

6. FUTURE WORK AND CONCLUSION

ACAP has effectively been used for rapid development of VCC and other advanced services, and its capabilities have successfully been used in both internal and external demonstrations. However, a number of important issues still remain. One such issue is security. Specifically, ACAP dynamically moves among communicating endpoints services objects that may contain executable code. This raises the issue of how to trust the authenticity and integrity of received service objects. On the other hand, service object owners should have control over who has access to their objects and where they move. In the former case, a PKI-based approach may be used to sign and verify the integrity of migratory service objects. In the latter case, we have a capability-based [8] approach, which allows owners to keep track of and revoke capabilities even when their objects are at remote locations. Fully addressing these and other security issues in ACAP is part of future work.

Another area of interest is to apply ACAP approach to management of network resources. Specifically, our model of individual environments managing their own resources can be extended to include network-level resources. That is, when a service migrates and asks for resources, the RM can grant or deny requests based on both current network conditions and locally available resources. This way, network operators/service providers can have fine-grained control over access and utilization of their resources. Providers can also implement resource usage-based billing.

Integrating network resource management with service creation and management is an important area for providers. This will allow providers to manage and prioritize limited manpower and network resources to revenue generating services. We are actively working in this area developing a service model that allows service definition across network and service layers.

In summary, ACAP applies object-oriented principles to managing services and environments. In ACAP, services are specified in terms of required resources, and environments in terms of available resources. ACAP facilitates services to adapt to diverse environments by allowing them to dynamically migrate and discover available resources in a given environment. Migratory services are facilitated by use of migratory objects in ACAP. In this paper, we have described in detail a Virtual Call Center (VCC) system to illustrate adaptive resource management requirements for high-level services. Furthermore, we have shown our approach to addressing these issues by describing how support for migratory services and peer-to-peer connections in ACAP is used in its architecture and implementation.

REFERENCES

[1] Alberi, J., Cochinwala, M., Cohen, E., Jackson, N., Pucci, M., and Sigman, E., "An Object-Based Framework for Communication Services," IEEE GlobeCom 2000, Workshop on Service Portability and Virtual Customer Environments (SerP-2000), December 2000.

[2] "OSGi's Service Platform Release 2" available at http://www.osgi.org.

[3] Abarca, C., et al., "Service Architecture," TINA-C Deliverable available at http://www.tinac.com/specifications/documents/sa50-main.pdf.

[4] James, J. and Singh, A.K., "Design of the Kan distributed object system," Concurrency: Practice and Experience 12(8): 755-797, 2000.

[5] Wu, D., Agrawal, D., and Abbadi, A.E., "Mobility and Extensibility in the StratOSphere Framework," Distributed and Parallel Databases 7(3): 289-317, 1999.

[6] Demmer, M., and Herlihy, M.P., "The Arrow Directory Protocol," Proc. of 12th International Symposium on Distributed Computing, September, 1998.

[7] Okino, C. and Cybenko, G., "Modeling and Analysis of Active Messages in Volatile Networks," Proc. of the 37th Allerton Conference on Communications, Control and Computing, Monticello, IL, 1999.

[8] Landwehr, C.E., "Formal Models for Computer Security," ACM Computing Surveys, Vol 13, No. 3, September 1981.

[9] http://www.recursionsw.com/products/voyager/voyager.asp.

[10] Cochinwala, M., "Database Performance for Next Generation Telecommunications," Proceedings of International Conference on Data Engineering 2001: 295-298.

ANALYSIS OF MOBILE RADIO ACCESS NETWORK USING THE SELF-ORGANIZING MAP

Kimmo Raivio,[1] Olli Simula,[1] Jaana Laiho[2] and Pasi Lehtimäki[1]

[1]*Helsinki University of Technology*
Laboratory of Computer and Information Science
P.O. Box 5400, FIN-02015 HUT, Finland
{Kimmo.Raivio,Olli.Simula,Pasi.Lehtimaki}@hut.fi

[2]*Nokia Networks*
P.O. Box 301, FIN-00045 Nokia Group, Finland
Jaana.Laiho@nokia.com

Abstract: Mobile networks produce a huge amount of spatio-temporal data. The data consists of parameters of base stations and quality information of calls. The Self-Organizing Map (SOM) is an efficient tool for visualization and clustering of multidimensional data. It transforms the input vectors on two-dimensional grid of prototype vectors and orders them. The ordered prototype vectors are easier to visualize and explore than the original data. There are two possible ways to start the analysis. We can build either a model of the network using state vectors with parameters from all mobile cells or a general one cell model trained using one cell state vectors from all cells. In both methods further analysis is needed. In the first method the distributions of parameters of one cell can be compared with the others and in the second it can be compared how well the general model represents each cell.

Keywords: Neural networks, self-organizing map, cellular network, performance optimisation.

1. Introduction

As the launch of third generation technology approaches, operators are forming strategies for the deployment of their networks. These strategies must be supported by realistic business plans both in terms of future service demand estimates and the requirement for investment in network infrastructure.

When provisioning 3G services the control for the access part can be divided into three levels. Two lowest layers are radio resource management (RRM) functionalities and the highest hierarchy level is control performed by the network management system (NMS). More about this control hierarchy can be found in [10]. The scope of this paper is the NMS level. The role of NMS is essential owing to the fact that major enhancements or new service roll-outs are planned by utilizing the measured long term performance data from existing network.

The multidimensional performance space in future cellular networks force the traditional operator processes to go through some major changes. Additional challenges arise from the fact that in the case of 3G there will be multiple services, customer differentiation (customers with different priorities) and multiple radio access technologies to be managed simultaneously, optimally, as one resource pool. Furthermore, the high competitive situation forces operators to fast changes in service provisioning. All this will move the focus of operators daily tasks from offline planning to rapid network performance evaluation, trend analysis and optimisation based on network measurements. Therefore new analysis schemes for 3G networks are presented in this paper. The strength of the proposed method is its ability to combine multiple measurements and thus provide the result in a simple format despite the fact that the input space is very complex. The method also aids the operator in visualizing the service performance and in classifying the cells. The cell classification (clustering) will aid the operator in setting the configuration parameters controlling the service provisioning. Furthermore, similarly behaving cells can be identified and thus problem solving in the network is more effective.

In this paper, the use of the Self-Organizing Map (SOM) in optimization process is proposed. The SOM is a widely used neural network algorithm [7]. It has several beneficial features that make it a useful tool in data mining and exploration. The SOM follows the probability density function of the underlying data and functions, thus, as an efficient clustering and data reduction algorithm. The SOM is readily explainable, simple and - perhaps most importantly - highly visual. SOM based methods have been applied in the analysis of processes data, e.g., in steel and forest industry [8]. In addition, the SOM has been used in analysis and monitoring of telecommunications systems. Applications include novel equalizer structures for discrete-signal detection and adaptive resource allocation in telecommunications networks. In this paper, wideband code division multiple access (WCDMA) mobile network has been analyzed using the SOM. The goal is to develop efficient adaptive methods for monitoring the network behavior and performance. Special interest is on finding clusters of mobile cells, which can be configured using similar parameters.

In [3], [5] and [13] examples of 3G optimization cases are represented. In general the availability of references related to 3G analysis and optimization is limited. This is owing to the fact that there are very few commercial networks deployed at the time of writing. In abovementioned references the approach has been parameter centric: how to measure and tune configuration parameters to obtain wanted performance. In the case of this paper the network status visualization is the main focus. This information can be further used in order to obtain optimization of correct parameter/parameter set of selected cells.

In the next section, the application domain which is mobile radio access network is described. Then the SOM algorithm is presented in Section 3 and two methods to classify mobile cells are described in Sections 4 and 5.

2. Mobile network and the data

The scope of this section is to describe the used network scenario and the parameters used in the simulations. The data used in this work has been generated using WCDMA radio network simulator [6]. The WCDMA radio network depicted in Fig. 1 has been planned to provide 64-kbps service with 95% outdoor coverage probability. The average site distance is around 910 m.

The network configuration used to produce the data consisted of 32 base stations in Helsinki city area. The users of the network were circuit-switched with 64-kbps and

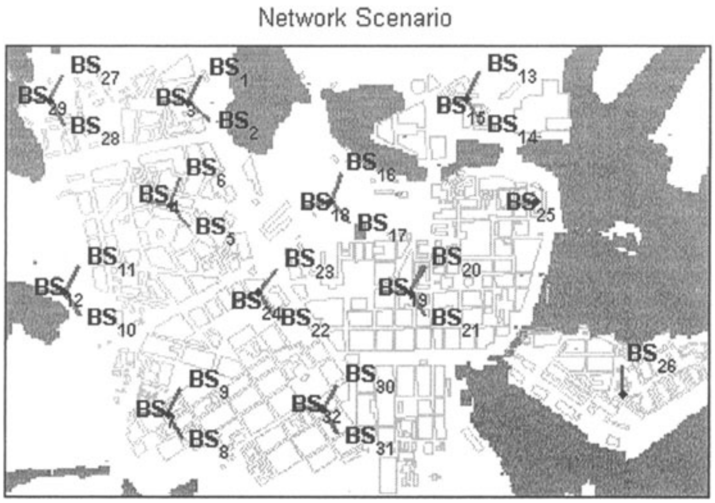


Figure 1. Helsinki city area with base stations.

the admission control was parameterized so that uplink interference had no impact on the admission process. The most important radio network simulation parameters are listed in Table 1.

Table 1. Important radio network parameters

Terminal maximum power	126 mW
Base station maximum power	20 W
Base station maximum power per link	450 mW
Target of UL/DL FER	5 %
Uplink system noise	-102.9 dBm
Downlink system noise	-99.9 dBm
Terminal speed	3 km/h

Used propagation model was Okumura-Hata with average area correction factor of -1.5 dB (excluding water areas). The multipath channel model was Vehicular A: five-taps with gains of -2.9, -5.2, -9.5, -13 and -15 dB respectively.

Slow fading deviation was 8 dB and the correlation distance was 50 m. Minimum coupling loss was 50 dB. Pilot power was 1 W. Softhandover was limited by saving maximum 3 links per terminal.

Power control is done once in a frame only to speed up the simulation. The power control step size is 0 to 15 dB depending on the difference between the average E_b/I_o over 10 previous frames and the target E_b/I_o [9]. Number of subscribers was 2112, which generate five 120 second calls on the average in an hour. Total simulation time was 1800 seconds.

The state of the network is characterized by 17 parameters of each base station which are saved every 100ms. The parameters include uplink noise raise in dBs, downlink average total transmission power in watts, number of users and average frame error rate (FER) of both uplink and downlink.

In this study, only uplink noise raise and uplink FER of each cell is used. A logarithmic scale with 10^{-2} as minimum FER is used.

3. Self-Organizing Map

The Self-Organizing Map forms a nonlinear topology preserving mapping from the input space to the output space. This means that patterns near each other in the input space are mapped to neurons which are close to each other in the neural net. In the original algorithm, the SOM is trained by the following unsupervised algorithm.

Each input vector $x(t)$ is compared with node vectors m_i to find the best-matching unit (BMU) c.

$$||x - m_c|| = min_i\{||x - m_i||\} \tag{1}$$

The best-matching node and the neighboring nodes are modified in the direction of the input data.

$$m_i(t+1) = m_i(t) + \alpha(t)h_{ci}(t)[x(t) - m_i(t)] \tag{2}$$

The neighborhood function h_{ci} is usually a Gaussian function, which is centered around node c and multiplied by decreasing learning rate $\alpha(t)$.

One step of the training algorithm of the SOM is illustrated in Fig. 2. The size of the SOM is 16 units, which have been arranged into a two-dimensional grid of 4 by 4 units. A data sample is marked with a cross; the black circles are the values of the prototype vectors before, and the gray circles after updating them towards the data sample. This kind of an update step is repeated iteratively during the training process.

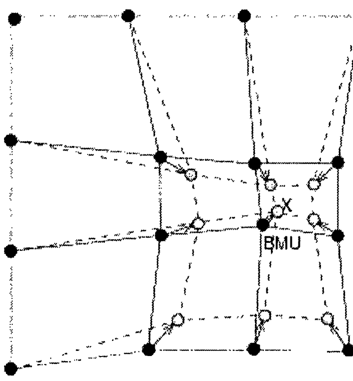


Figure 2. An illustration of the SOM training.

In this work, a batch version of the original algorithm is used, because it is computationally more effective. The samples collected from a fixed time interval are first averaged over the topological neighborhoods of the respective winner cells in the map. After that the node vectors are updated in one step using these averaged values, as in the classical K-means algorithm [11].

The SOM algorithm is able to perform both data clustering and visualization. The benefit of using SOM is in visualization of interesting parts of data. The algorithm moves the nodes of the map towards the areas of higher density of mapped input vectors. As a result, the SOM efficiently visualizes the clusters.

4. Classification of mobile cells using correlations of SOM component planes

Here a method for clustering mobile cells on the basis of covariance matrixes of SOM component planes is presented. The method utilizes the SOM algorithm twice. At first SOMs of one variable are built (see Sec. 4.1). Then the covariance matrixes of the SOM component planes are computed. Covariances of one or more variables are used as data to a second SOM. The outputs of the second SOM are the clusters of mobile cells (see Sec. 4.2). In Sec. 4.3 the classification of using several variables is demonstrated.

4.1 SOM of one variable

Data of each cell is masked so that one variable of each mobile cell is analyzed with the corresponding ones of the other cells. The data to be analyzed has been normalized to zero mean and unit variance as one data vector over all the cells. Here uplink noise raise and logarithm of uplink FER have been analyzed using the SOM. Hexagonal 2D neighborhood grid of 10 x 15 nodes is used. Fig. 3 shows the SOM component planes when the FER is studied. There is one component plane per each mobile cell. The parameter values of the mobile network state at one moment can be read from similar locations on component planes. For example, upper left corner gives one possible combination of network error rates.

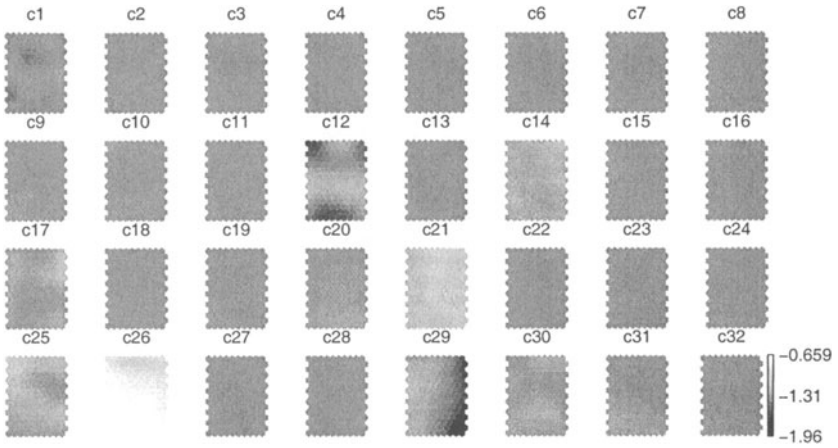


Figure 3. SOM component planes of the FERs. Minimum FER is fixed to 10^{-2}.

The component planes are visualized using a common color axis. This makes it possible to see the real error rates, but it also hides the smaller variations inside the cells. In the figure only some of the cells seem to differ from the common behavior. Cell 26 has a lot higher FER than all the others.

4.2 Reorganized SOM component planes

If we are interested in, for example, to find out which mobile cells have similar FER distribution, the task of human analyzer can be made easier by further processing the component planes of SOM. This kind of postprocessing is more important if the number of component planes is higher.

The component planes are considered as separate figures. Covariance matrix of the figures is computed by first converting the figure dot or node values c_{ij}^n to vectors a^n, where i and j are the coordinates on the map and n is the mobile cell number. The length of each vector a^n is the product of component plane dimensions.

The covariance matrix C of the planes a^n is the new data, which will be used in Sec. 4.3. This data has one row for each mobile cell. A new second level SOM is trained using the covariance matrix. The topology of the new SOM is 2D rectangular grid. Because 32 component planes are analyzed grid of size 8 x 8 nodes is used. The covariance matrix row of each cell is mapped on the second level SOM and the best-matching unit (BMU) for each mobile cell is found. The map nodes are labeled using the results of BMU search.

The second level SOM can be visualized using the labels or the corresponding first level SOM component planes. In the latter case the SOM component planes have been reorganized so that the similar ones locate near each other. This makes it easier to find correlations between SOM components.

The SOM planes reorganization method has been discussed earlier in [14] and [15]. In the latter paper several modifications of the algorithm have been represented. In Fig. 4 the SOM component planes of Fig. 3 have been reordered using the method above.

From the Fig. 4 we can see that cell 26 has higher error rate than the others and that also the FER distributions of cells 12, 14, 17, 21, 25, 29, 30 differ quite much from the others. The rest of the cells have similar FER distribution. The different behavior of cells can be partly explained with the help of the radio network plan: cells 26 and 21 suffer from bad interference situation (due to the fact that the water areas allow easy propagation for interfering signals), in case of cells 12 and 29 the difference can be explained with the position of the cell. These cells are located at the edge of the network, and thus only little data is available. Number of neighboring interfering cells is also lower compared to the other cells.

4.3 Classification using several variables

Several SOMs for different variables can be built and reorganized using the methods of previous sections. The covariance matrixes C_k of all first level SOMs can be combined so that we get a new data matrix $C = [C_k C_l \ldots]$, $k \neq l$. Matrix C has a row C^n for each cell n. The row is a concatenated vector of cell correlations of used variables.

When the SOM is trained using this new data, we are able to get a new ordering of the cells. The result (Fig. 5) is about the same as in Fig. 4. Only cells 14, 21, 25 and 26 differ from the others. It is obvious that in this case correlations of uplink noise raises

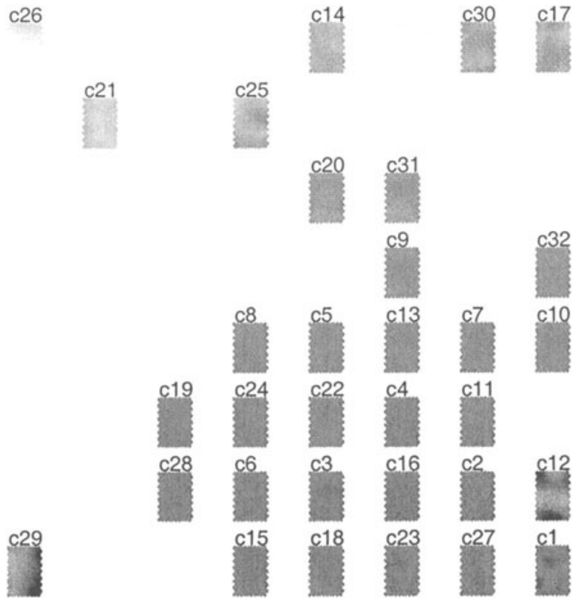


Figure 4. SOM planes representing error rates of the mobile cells are reorganized. Planes are also labeled using cell numbers. Color scaling is as in Fig. 3.

do not have a meaningful effect on clustering. The same cells differ from common behavior as before.

Clusters of mobile cells can be found using U-matrix presentation [12] or hierarchical clustering of SOM node vectors [16]. Hierarchical clustering can be either divisive or agglomerative [2]. In divisive hierarchical clustering data vectors are separated in finer groupings. Agglomerative hierarchical clustering methods add similar groups together starting from some initial base clusters. The base clusters can be either all SOM node vectors or some set of them like local minimas. Here, group-average hierarchical clustering is used with SOM node vectors as base clusters.

The number of clusters can be fixed manually on basis of the U-matrix or more sophisticated methods like Davies-Bouldin index can be used [1]. Here, the number of clusters is fixed manually to four. The clusters of the original data are shown in Fig. 6. The classification result combined with locations of the cells has been shown in Fig. 7. As it can be seen, cells can be characterized and divided into different clusters. In a radio network optimization process it is reasonable to assume that the configuration parameters for cells within a cluster are at least partly the same. The BMUs of the original data have been printed (in Fig. 6) using subscript 1 and the BMUs of the new data set with subscript 2 ($c1_1$ means cell 1 with original data). In the new data set, the pilot power of cells 21 and 26 have been decreased from the original 1W to 0.5W. The reason for this change was to reduce the physical size of these two cells to improve the overall quality of service. It can be seen in the following results that the change was not yet adequate.

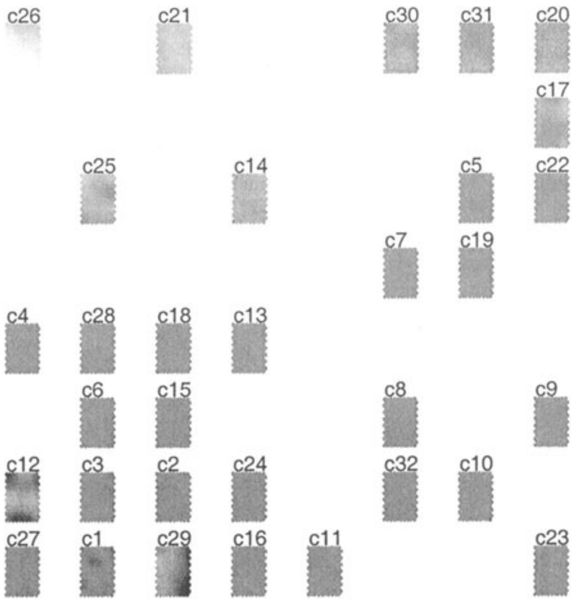


Figure 5. SOM planes representing error rates are ordered using correlations of both uplink noise raise and FER component planes.

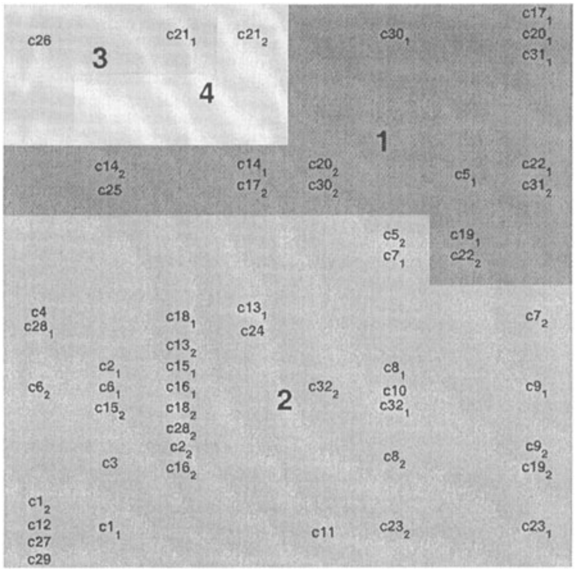


Figure 6. Four clusters of mobile cells. Cell clusters are found using correlations of both uplink noise raise and FER component planes.

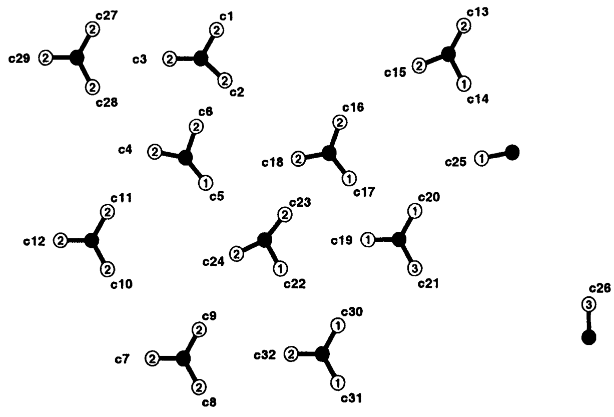


Figure 7. Locations and classes of mobile cells.

When new data is analyzed, SOM component plane representation of the data has to be constructed. The easiest way to do this is training the SOM again. When the SOM is trained, the new data can be used in the BMU search or it can be masked out. If the new data is masked out in the BMU search, but used when the neurons are updated we can obtain similar SOM as before, but in addition we get the component planes for the new data. From the component planes new covariance matrices can be computed, new clusters can be found and the BMUs of the new and the old data can be found.

The method described above classifies mobile cells on basis of correlations of selected variables. A model of mobile network which describes the relations between mobile cells has been built. This method analyses the correlations between cells i.e. does a bad performing cell have degrading influence also on the neighboring cells.

5. Classification of mobile cells using cluster histograms

In Section 4 method to form data clusters was presented. The input data was used to build a model of the network. In this section another method for classification of mobile cells is presented. Also this method uses two levels of SOMs. In order to analyze sequence of data samples instead of a single data point a histogram map is computed. Histogram consists of proportions of data samples falling in each of the data clusters. These histograms describe the long-term behavior of data sequences and they are used in the cell classification. A new SOM is generated using the histogram information as the training set. By using a clustering algorithm exact behavioral clusters can be generated. These behavioral clusters are found by hierarchical clustering method, here the Ward clustering [2] with local minimas of SOM node vectors as base clusters. Histograms for each mobile cell are computed using the clusters as bins. The histograms are the data, which are used to train the second SOM and to find the BMUs for each cell.

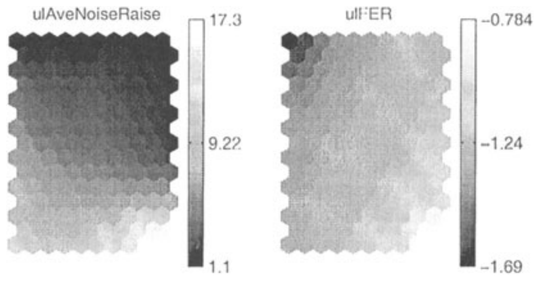


Figure 8. SOM trained by uplink noise raises and error rates of all the cells of the network.

5.1 General mobile cell model

In Fig. 8 the component planes of the general mobile cell model are shown. The optimal number of clusters minimizes Davies-Bouldin index [1]

$$\frac{1}{C}\sum_{k=1}^{C}\max_{l\neq k}\{\frac{S_c(Q_k)+S_c(Q_l)}{d_{ce}(Q_k,Q_l)}\} \tag{3}$$

where C is the number of clusters, S_c within-cluster distance and d_{ce} between clusters distance, Q_k and Q_l are the clusters. When Ward clustering is used, four clusters or states for mobile cells minimize the Davies-Bouldin index. In Fig. 9 state 4 represents the higher load state and the others normal state.

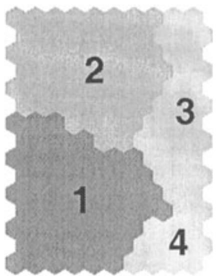


Figure 9. Four clusters of SOM node vectors given by Ward clustering and Davies-Bouldin index.

The BMUs of data vectors give the state or the class of the cell. From a sequence of states we can compute the class frequencies of mobile cells. Using these histograms as data to a second level SOM we get a SOM of histograms. The topology of the new SOM is 2D rectangular grid. Grid of size 8 x 8 nodes has been used as in 4.2. The BMU search of the map is based on Kullback-Leibler distance [4]. The Kullback-Leibler distance or relative entropy between two probability distributions $p_X(x)$ and $q_X(x)$ is defined by

$$D_{p||q} = \sum_{x \in \mathcal{X}} p_X(x) \log(\frac{p_X(x)}{q_X(x)}) \tag{4}$$

where the sum is over all states of the system (i.e., the alphabet $\mathcal{X}$ of the discrete random variable X).

The group-average hierarchical clustering with local minimas of SOM node vectors as base clusters gives new clusters of mobile cells. These clusters with mobile cell BMUs are shown in Fig. 10. The result is about the same as in the previous section. The shifts of states of cells when the pilot power of cells 21 and 26 is decreased are visualized using old clusters to label the new data and compute new histograms. As before the BMUs of the original data have been printed using subscript 1 and the BMUs of the new data set with subscript 2. The clustering information with spatial data is also shown in Fig. 11.

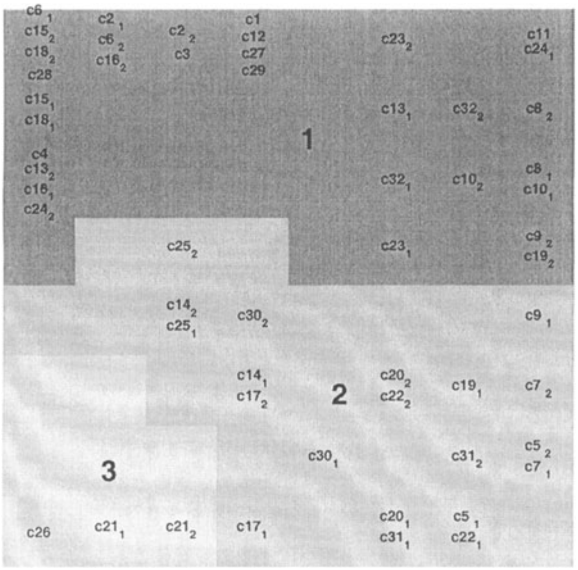


Figure 10. Three clusters of mobile cells. Cell clusters are found using cluster histograms of SOM trained by uplink noise raise and FER of each cell.

The method described above classifies mobile cells using class frequencies as models of mobile cell behavior. The distributions describe how much a particular mobile cell differs from a general cell model, which has been built using as much data as possible. General cell model is an absolute reference for cell performance. The position of the cell on the reference map reflects its actual performance.

6. Conclusion

In this paper two new methods to monitor mobile network state have been presented. In the first method, lower level SOMs of one variable are first build. Covariance matrices of the component planes of these SOMs are then used to train another

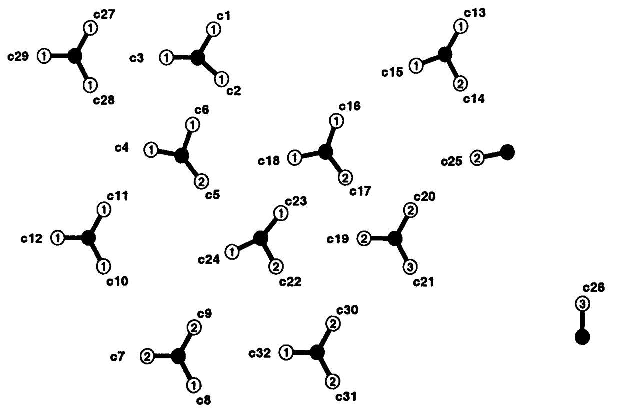


Figure 11. Locations and classes of base stations.

map, which reorders the mobile cells. In the second method, a lower level SOM, which represents general mobile cell model is built. Histograms of the states of the base stations are built using clusters of lower level SOM. The same clusters can be used later to find out histograms of new data. Thus, the operational mode of each cell and the whole network can be monitored. The first method is powerful when the correlation between the cells is of interest. The second method is used when information of the absolute performance of cells is required.

The data which is used to build the lower level SOM in the method based on class histograms should be selected carefully so that it represents well all the possible states of the cells. If it does not, the lower level SOM should be trained again using new set of data.

In this paper it has been demonstrated that SOM can be used in cell clustering. The possibility of finding similarly behaving cells will make the operators' optimization task more cost effective. Similar configuration parameter sets for cells within a cluster can be utilized. Furthermore, owing to the highly visual nature of SOM, the multidimensional performance space can be visualized more effective than with traditional tools. Thus the operators have means to get an interpretation of the service performance.

Acknowledgment

The study has been financed by Nokia Networks, Nokia Mobile Phones and the Technology Development Centre Finland (TEKES) which is gratefully acknowledged. Furthermore, the authors wish to thank their colleagues for valuable comments during the work.

References

[1] D.L. Davies and D.W. Bouldin. A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1(2):224–227, April 1979.

[2] B.S. Everitt. *Cluster Analysis*. Arnold, 1993.

[3] A. Flanagan and T. Novosad. Automatic selection of window add in a WCDMA radio network based on cost function minimization. In *Proceedings of International Symposium on Spread Spectrum Techniques and Applications*, 2002.

[4] S. Haykin. *Neural Networks, a Comprehensive Foundation.* Macmillan, 1999.

[5] A. Höglund and K. Valkealahti. Quality-based tuning of cell downlink load target and link power maxima in WCDMA. In *Proceedings of IEEE Vehicular Technology Conference*, Fall 2002.

[6] S. Hämäläinen, H. Holma, and K. Sipilä. Advanced WCDMA radio network simulator. In *Personal, Indoor and Mobile Radio Communications*, volume 2, pages 951–955, Osaka, Japan, September 12-15 1999.

[7] T. Kohonen. *Self-Organizing Maps*. Springer-Verlag, Berlin, 1995.

[8] T. Kohonen, E. Oja, O. Simula, A. Visa, and J. Kangas. Engineering applications of the self-organizing map. *Proceedings of the IEEE*, 84(10):1358–1384, October 1996.

[9] R. Kwan and M. Rinne. A comparison of WCDMA network performance results with frame vs slot resolution simulations. In *Proc. 5th CDMA International Conference & Exhibition*, volume 2, pages 478–482, Seoul, Korea, Nov 22-25 2000.

[10] J. Laiho, A. Wacker, and T. Novosad (ed.). *Radio Network Planning and Optimisation for UMTS*, chapter 8, pages 329–363. John Wiley & Sons Ltd., 2001.

[11] Y. Linde, A. Buzo, and R.M. Gray. An algorithm for vector quantizer design. *IEEE Transactions on Communications*, 28(1):84–95, January 1980.

[12] A. Ultsch and H.P. Siemon. Kohonen's self organizing feature maps for exploratory data analysis. In *Proceedings of the International Neural Network Conference*, pages 305–308, Dordrecht, Netherlands, 1990.

[13] K. Valkealahti, A. Höglund, J. Parkkinen, and A. Hämäläinen. WCDMA common pilot power control for load and coverage balancing. In *Personal, Indoor and Mobile Radio Communications*, volume 3, pages 1412–1416, 2002.

[14] J. Vesanto. SOM-based data visualization methods. *Intelligent Data Analysis*, 3(2):111–126, 1999.

[15] J. Vesanto and J. Ahola. Hunting for Correlations in Data Using the Self-Organizing Map. In H. Bothe, E. Oja, E. Massad, and C. Haefke, editors, *Proceeding of the International ICSC Congress on Computational Intelligence Methods and Applications (CIMA '99)*, pages 279–285. ICSC Academic Press, 1999.

[16] J. Vesanto and E. Alhoniemi. Clustering of the self-organizing map. *IEEE Transactions on Neural Networks*, 11(3):586–600, May 2000.

SHORT PAPER SESSION 3

Configuration and Architectures

Co-Chairs: Cynthia Hood
Illinois Institute of Technology, USA

Omar Cherkaoui
University of Quebec in Montreal, Canada

AN ARCHITECTURE FOR PROVISIONING IP SERVICES IN AN OPERATIONS SUPPORT SYSTEM

A. John[1], B. Sugla[2], H. Krishnan, E. Park, A. Raghu, R. Sequiera, A. Wanchoo
[1]ajita@avaya.com, [2]sugla@dset.com, DSET Corporation, Shrewsbury, NJ

Abstract: This paper discusses an architecture for a provisioning system that meets the challenges currently facing service providers in a service and subscriber-based OSS environment. The architecture makes a clear separation between a provisioning core, which is a general framework for provisioning services, and service definitions that model the provisioning view of a service. The architecture is distributed, scalable and extensible and is especially suited for scenarios where a large number of services is expected to be offered, deployed, and managed. The service definitions can be used by the other OSS components to correlate information to provide complete device-to-service-to-subscriber diagnostics for faults, performance degradations, and accounting. It is argued that this approach leads to natural, efficient and effective management solutions.

Key words: Provisioning, IP services, Operations Support System, OSS, VPN

1. INTRODUCTION

The evolution of Operations Support Systems (OSS) from basic device management components to service and subscriber-based ones has come with complexities in widely differing services, standards, and vendors and complexities in interactions between different OSS components. These complexities have made the provisioning and maintenance of IP services such as VPNs, firewall-based security, and web-based hosting a major hurdle. The major issues in provisioning are providing support for multiple services, dependencies among services, multi-vendor equipment, transaction and rollback, seamless interaction with other OSS components, scalability and customization requirements in different service provider environments. This paper discusses these issues and a provisioning architecture to address them. The architecture consists of service definitions that model the provisioning view of a service and a provisioning core that uses distributed service agents. The service definitions can be used by the other OSS components to correlate information to provide complete device-to-service-to-subscriber diagnostics for faults, performance degradations, and accounting.

With respect to related work, the representation of services has been explored in [1], [2]. Several approaches for provisioning services such as service object-based approach [3], profile-based service provisioning [4] and model-based configuration [5] have been looked at. This work undertakes a comprehensive provisioning architecture for multiple IP services in the context of the current challenges faced by service providers. While the notion of distributed service agents is not new, the contribution of this work is to show how the use of a distributed architecture and an effective decoupling of service definitions and the provisioning core can address the current challenges of IP service provisioning.

2. ISSUES IN PROVISIONING

A provisioning system provisions services on the service infrastructure of a service provider. The service infrastructure includes devices such as web servers, email servers, AAA servers, firewalls and routers. Services may be server-based services such as web hosting or network-based services such as VPNs. The architecture for a provisioning system should support multiple services on a single platform and the easy addition of new services on the same platform. It should be possible to modify the implementation of a service without affecting other services, compose existing services into new ones as in combining an authorization service (from a AAA server), a DNS service, and the provisioning of a web server into a web service. It should also support multi-vendor solutions. A provisioning system should easily interface with other OSS components. The provisioning system forms a key component of a modern OSS by providing the models of a service that should follow well-structured definitions that other OSS components can decipher. IP services such as VPNs may span the domains of multiple providers. The provisioning system for the service provider that accepts service requests should be able to negotiate with the provisioning systems of other service providers, under the agreements they have between them. The provisioning of a service may also require calls to legacy or proprietary systems. The system should be scalable in the number of simultaneous users of the system, the throughput of service requests, the number of supported service infrastructure elements such as routers, and the number of subscribers and services. The provisioning system should also enable transaction support for service requests by rolling back changes to a device if any part of the service request fails. However, the service provider may still want control over the rollbacks and hence it cannot be hard-coded into the system.

3. PROVISIONING SYSTEM ARCHITECTURE

Our architecture is separated into a *provisioning core* that is responsible for handling service requests and *service definitions* that model the provisioning view of services. The provisioning core interprets service definitions to provision requests for services. A service may be as complex as an MPLS VPN service or it may be a component of a complex service such as the configuration of an edge router or it may be an operational service such as sending email.

The provisioning core is a distributed framework for provisioning. It consists of entities called *service agents*. Each service agent is a provider of one or more services and accepts requests for provisioning these services. A service agent may, in turn, send requests for services to other service agents. Service agents register with a registry to advertise their identity, the services they are providing, and any distinguishing characteristics about the services provided. Depending on the needs of the service provider, there may be several provisioning systems, each composed of a set of service agents, for offering several services. Service agents may be deployed at different locations. For instance, a service agent at the Network Operation Center may request a service agent at a customer premise location to provision the customer's firewall. The communication between the service agents is secure and can traverse firewalls. Communication is asynchronous so that requesting service agents are not blocked until requests are provisioned.

Each service agent exports a set of interfaces for the services it provides. Each interface consists of: (1) a service-independent part containing the type of request (add/delete/edit), subscriber information etc. and (2) a service-dependent part (described below) containing the service parameters that have to be provisioned for the service. Recent industry efforts such as OSSJ [6] can be used for the service-independent part of the interfaces.

Service definitions model the provisioning view of a service by defining its data and process models. The three major components of a service definition are: (1) Service parameters capture the data that comes as part of a service order for that service. Some parameters for a VPN service are the VPN Identifier, the end points of the VPN, and the requested bandwidth. (2) Process model captures the decomposition of a service request into requests for sub-services that are provided by other service agents or the invocation of a software module for the allocation of system resources such as IP addresses. In addition, the process model specifies the transactional semantics for the service: what actions need to take place in the event of a failure returned by the invocation of a sub-service. (3) Service Instance is the information that needs to be stored in a service inventory for a service request after it is provisioned. It includes service parameter values and is associated with the subscriber for whom the service is provisioned.

After receipt of a service request, it is translated into requests for sub-services as specified by the process model. Replies from these sub-requests need to be managed and rollbacks specified by the transactional model may need to be invoked. After completion of this process, a service instance may be created in the service inventory. A workflow engine may be needed to manage the process model if the services provided by the service agent are composed of sub-level services with notions of transactions. Examples of this may be the IPSec VPN or MPLS VPN services. If the services provided by the service agent are simpler, a call to a simple routine may be sufficient.

The advantages of the proposed architecture are numerous. New services can be added without changing the provisioning core. Existing service definitions may be customized or composed to form new services without changing the provisioning core. The composition of services enables multi-vendor solutions to be supported and enables external provisioning systems to be invoked as a sub-service. Calls to proprietary or legacy systems from the process model of a service can be accommodated. Scalability requirements on a parameter (such as users, network devices) can be met through the distributed architecture where a group of service agents can collectively provide a service. The group may act as a cluster where load balancing over the parameter is done. Modifying the process model can accommodate the need for customization of rollbacks in transactions.

A provisioning system incorporating the proposed architecture and based on the J2EE specification is in place at DSET Corp. The provisioning core has been implemented using java. Communication between the different service agents is done through the Java Messaging System (JMS). The agent and service interfaces are represented using XML and the process model is represented using XSL. The service and subscriber information is partitioned between an LDAP-based directory and a relational database.

4. CONCLUSIONS

This paper discusses the issues in building a provisioning system for multiple IP services in an OSS. It lays out an architecture based on service definitions that model the provisioning view of services and a provisioning core that interprets the service definitions to provision service requests. The paper highlights how a flexible, extensible, and scalable architecture addresses many of the difficult issues faced in building a provisioning system. Additionally, a service definition approach helps other components in the OSS to infer service relationships between the subscribers and service infrastructure. This enables the offer of an end-to-end OSS architecture where faults and degradations from the devices flow into fault and performance management systems that correlate information using the service definitions to report affected subscribers and services.

REFERENCES

[1] Common Information Model (CIM). Core Model, DMTF, 1998.

[2] Serivce Level Management for Enterprise Networks. L. Lewis. Artech House, ISBN I-58053-016-8, 1999.

[3] Provisioning Voice Over Packet Networks: A Metadata Driven, Service Object-Based Approach. Jung Tjong, Prakash Bettadapur and Alexander Clemm. Proc. of the 12th International Workshop on Distributed Systems: Operations and Management DSOM'2001 France, Oct. 2001. Eds: O. Festor and A. Pras.

[4] Profile-based Subscriber Service Provisioning. F. Shen, A. Clemm. Proc. of the 8th Network Operations and Management Symposium, NOMS 2002, April 2002.

[5] Model-based Configuration of VPNs. I. Luck, S. Vogel, H. Krumm. Proc. of the 8th Network Operations and Management Symposium, NOMS 2002, April 2002.

[6] OSSJ: Java Specification for Operations Support Systems. Sun Microsystems.

WIRELESS TERMINAL MANAGEMENT ARCHITECTURES

R. State
The MADYNES Research Team, INRIA-LORIA, 615 Rue du Jardin Botanique, 54600 Villers-les-Nancy, France, Radu.State@loria.fr

Abstract: The advent of multi-technology networks offering ubiquitous services over advanced wireless network infrastructure demands an integrated management approach. We address in this paper an integrated management approach for wireless and mobile infrastructures by analysing the particularities of this environment and proposing a management architecture based on application based overlay networks. Our approach proposes the integration of a management agent within a service gateway as well as an additional notification support based either on a peer to peer infrastructure or on the session initiation protocol.

Key words: Terminal Management, OSGI, SyncML, Beyond 3G

1. INTRODUCTION

The advent of advanced wireless networks providing high quality multimedia services to mobile users creates an important market for value added services offered on top of these networks. Such services will be typically offered by service providers to end-users. Service providers will interact with different network operators in order to allow service deployment to be done efficiently over a large geographic area. One important problem that service providers are facing is concerned with the interaction with wireless network operators. The difficulty in this interaction lies mainly in service management capabilities offered to the service manager.

We address in this paper the management of future, next generation services. These services will be provided to the end-user seamlessly and in a technology and infrastructure transparent way. Nowadays, mobile users are bound to network operators and have a limited choice of services to which they can subscribe. Most of these services are dependent on the network operators to provide them. In most cases, the management of new services is limited.

The challenge to provide ubiquitous services to users demands a new way to manage the inherent dynamics.

2. CHALLENGES FOR TERMINAL MANAGEMENT

Although, management architectures have been widely developed for the management of fixed infrastructure network elements, the advent of personalized wireless technology demands the extension of the management towards a new dimension. This new dimension is given by several factors: We have to manage low resource type of equipment needing minimal software overhead for their management.

The connection with these devices will be done wireless, over unreliable links across several management domains. Roaming from within foreign access networks demands access to management agents across one or several domains. Such a situation is rarely encountered in current network management, where the connectivity network to the agent is within the same administrative domain.

The managed entities will not be always online. Due to coverage factors, equipment battery power and its user behavior, management actions should be performed asynchronously.

The network connectivity towards the terminal will be done over heterogeneous links. While current practical management solutions assume an SNMP over IP communication, the range of wireless access protocols (WAP: Wireless Access Protocol), GPRS [1] (General Packet Radio Service, where incoming connections to a terminal are not possible), HTML or even SMS (Short Message Service) demands a versatile communication paradigm to work over these technologies. There are a large variety of terminal related technologies. These multiple technologies are due to the converged usage of multiple access networks envisaged in the beyond 3G proposals where multimode terminals have support for WLAN/GPRS/DVB or subsets of this set, and services are delivered based on the coverage, service accessibility or desired service levels, but also to a personalized service environment where multiple devices cooperate in order to provide a global service to a user.

On a higher level, service and application level management are more important, since in a first phase most new functionalities will be application related. Early attempts have already started with ringing tones that can be downloaded by users and next steps could go more further. For instance, in a car, new software for the injection control could be upgraded. Terminal management would allow easier service management and hence facilitate the introduction of new attractive services.

3. MANAGEMENT ARCHITECTURE

While current terminal software architectures can vary considerably and are mostly vendor specific, we consider a management architecture as described next (see Figure 1), as a common minimal requirement. The environment on which services are dynamically deployed is an Open Service Gateway Initiative [2] framework. This framework allows a service to deliver services to a local network. Services are implemented in Java and encapsulated in software packages called bundles. Although, the initial framework was designed with the target scenario of service providers providing services to residential gateways, the low resources required to run the framework make it a candidate to be use on a wide range of J2ME (Java Micro-edition) devices like portable phones, and to more powerful targets like for instance embedded computers in vehicles or homes.

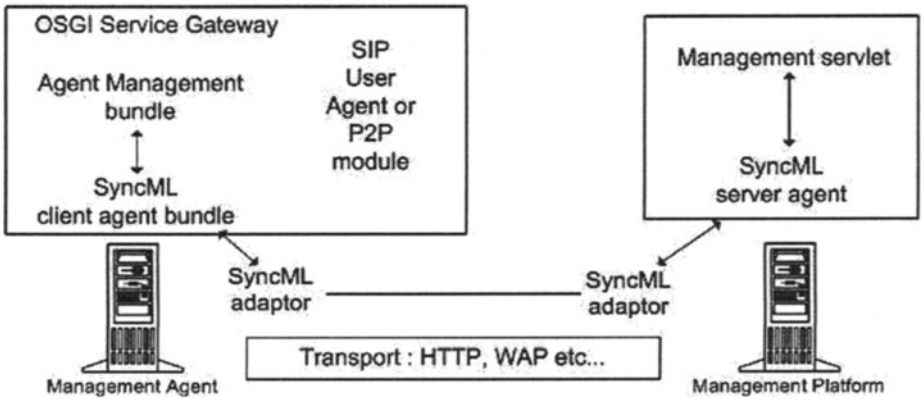


Figure 1. Management architecture

The framework uses a service registry, where bundles can locate needed services, and where each service can be associated to a set of properties. Events are generated whenever a service is registered, unregistered, started, stopped, or when some of its properties are changes. This mechanism allows building a management bundle responsible to manage the rest of the bundles relatively easy. Such a bundle relies on the framework to monitor the life cycles and changes in other bundles properties. It can be also used to change these properties. For our management purposes we require two special bundles to run in the framework. The first one is a management agent responsible for managing the OSGI framework, the second one is a SyncML client agent bundle, responsible for providing a communication stack to the management bundle.

The SyncML language is based on XML, and provides a synchronization protocol, as well as a device management protocol [3] and a set of device level standardized managed objects. Several characteristics make it a good choice for a management protocol. Firstly it was conceived with small footprint devices in mind and transport over different protocols (IP, HTTP, WAP, Bluetooth, SMTP) is possible. Secondly, most terminals will have a SyncML engine for synchronizing e-mail boxes, calendars etc. Therefore, it is natural to reuse this engine for the management plane. Thirdly, a synchronization protocol is well adapted to communicate with equipment being offline relatively often. If SyncML was initially designed for the configuration of devices, its extension towards OSGI type of service management is relatively straightforward. From a management point of view we need an agent capable of coping with a dynamic MIB.

The communication between the management bundle and a management application comprises a regular datapath for the management information and an additional notification support. The main objective of this notification support is to allow the management application to contact the agent and notify the latter that it must be managed. This notification should be capable to deal with dynamic addresses of the agent and with firewalls blocking management traffic. Two potential solutions for such a notification support are proposed: the first one uses a peer-to-peer infrastructure and the second one is based on SIP.

The Agent Management Bundle is responsible for the management instrumentation of the service gateway. Its design is conceptually similar to the dynamic MIB agent proposed in [5]. This bundle is capable to receive a XML encoded MIB of a bundle, store it in its MIB and perform management operations based on this MIB. This approach is an extension to the standard usage of SyncML where only simple device management is proposed. The extension concerns the usage to service configuration. The agent is responsible to dynamically build the bundle MIB for bundles that are installed or configured otherwise than through the agent.

4. REFERENCES

[1] C. Andersson. "GPRS and 3G wireless applications". Wiley, 2001 Publishing. 1999.

[2] OSGI Service Platform Release 2. The Open Service Gateway Initiative www.osgi.org.

[3] SyncML Device Management Protocol v1.1. SyncML forum. www.syncml.org.

[4] H. Handley, H. Shulzerinne, E. Schooler and J. Rosenberg. "SIP: session initation protocol". IETF (Proposed standard) 2543, March 1999.

[5] A. John, K. Vandween, B. Sugla. "XNAMI –An extensible XML Paradigm for Network and Application Management Instrumentation". Proc 10 th. International workshop on Distributed Systems: Operations and Management DSOM'1999 Zurich, Switzerland

A SCALABLE AND EFFICIENT INTER-DOMAIN QOS ROUTING ARCHITECTURE FOR DIFFSERV NETWORKS[1]

Haci A. Mantar*, Junseok Hwang+, Steve J. Chapin*, Ibrahim Okumus*
L. C. Smith College of Engineering and Computer Science
+*School of Information Studies, Syracuse University*
{hamantar, jshwang,iokumus}@syr.edu, chapin@ecs.syr.edu

Abstract: With Bandwidth Broker (BB) support in each domain, Differentiated Services (Diffserv) is seen as a key technology for achieving QoS guarantees in a scalable, efficient, and deployable manner in the Internet. In this paper we present a Route Server (RS) architecture that is compatible with the BB model for inter-domain QoS routing. The RS is responsible for determining QoS routes on behalf of all the routers and for exchanging routing information with its neighboring peers. It optimizes network resources by taking the intra-domain resource utilization state into account for selecting a route. It also achieves scalability by pre-computing a limited number of paths for each destination region and mapping all the packets to one of these paths regardless of their sources.

Key words: Bandwidth Broker (BB), Inter-domain QoS Routing, Route Server.

1. INTRODUCTION

Providing end-to-end QoS in a multi-domain environment is a very complex problem. Each domain is under different administration so that QoS *policies* and *services* in one domain might be significantly different from those in other domains [1,4,5]. A domain administration has control only of its own domain resources and discloses very restricted information about its internal network to others (e.g., competitors).

Recently, many studies [1,3-9] have addressed the important role of *central servers* for controlling access and managing resources in a domain. A well-recognized central server is a Bandwidth Broker (BB) [1]. The idea behind the BB model is to provide Intserv-type end-to-end quantitative QoS guarantees in Diffserv-enabled networks without per-flow state in the network core. With such a scheme, control functionalities such as policy control, admission control, and resource reservation are decoupled from routers into the BB and therefore the BB makes policy access and

[1]This work is supported by the National Science Foundation under grant ANI-0123939

increases the scalability of scalable as well as the likelihood of the deployment of QoS because of the minimum changes required in network infrastructure, and simplification of accounting and billing. Similarly, the IETF proposed COPS, where a Policy Decision Point performs tasks similar to those of a BB. In this sense, the BB-supported Diffserv model is realized to be a key technology by many researchers [1,3-9] for making end-to-end QoS possible across multi-domains networks.

Having said all this, the BB-supported Diffserv model still has the following open issues: 1) how to get dynamic link states without signaling with routers; 2) how to assure quantitative QoS guarantees with no reservation in core routers; 3) how to get QoS and cost information about the networks beyond its domain (e.g., which provider it should choose for border-crossing traffic); 4) how to perform service mapping in the inter-connection points; 5) how to manage its domain resources efficiently; 6) how to communicate and reserve resources with neighboring BB for border-crossing traffic; 7) how to achieve inter-domain scalability (signaling between BB and state in border routers). In our previous work [3,6,7], we presented solutions for problems 6 and 7. The concept of this paper is a base for the solution of problems 1-5.

We propose a route server (RS) architecture that works in conjunction with a BB reservation model (independently BGP-4) and makes scalable and efficient QoS routing possible in a multi-domain, multi-policy, multi-technology environment with restricted routing information. In particular, we define two *central* routing entities called Exterior Route Server (eRS) and interior Route Server (iRS) as main components of a BB (Figure 1). The eRS is mainly responsible for routing information distribution between domains and path computation and selection at a high level. The iRS maintains intra-domain QoS information and presents to the eRS in a simplified fashion. At run time the BB uses both iRS and eRS databases to perform admission control without invoking routing protocol and interaction nodes along the path.

2. DIFFSERV ROUTE SERVER ARCHITECTURE

2.1 Assumptions and Definitions

As shown in Figure 1, our BB architecture comprises three main components: the Exterior Route Server (eRS), the Interior Route Server (iRS), and the Reservation Module (RM). Before going into more detail on these components, we make the following definitions.

Service level agreement (SLA): We assume that domains have *bilateral* SLAs with their adjacent neighbors, which can act as both customer and provider.

Destination region: Due to signaling and state scalability problems with per-flow routing schemes, a destination region (***destID***) in RS represents a domain, a network, or a set of points identified by CIDR (IPv4/X where $X<32$).

Point-to-Destination Services (*dest_serv*) specifies the upper bounds of QoS constraints (such as delay, loss ratio) that identifiable traffic will be experienced from ingress point of the provider domain to the final destination region, regardless of the network utilization. Each domain assumed to have limited numbers of pre-defined (see next section) *dest_serv* for each destination region and all incoming traffic is mapped to one of these *dest_serv*.

Edge-to-Edge QoS Class defines upper bounds of QoS constraints that identifiable traffic will experience from the ingress nodes to the egress nodes of a domain, regardless of network utilization. Conceptually, this is very similar to Per-domain behavior (PDB) defined in [2]. (From now on, the terms "edge-to-edge QoS" and "PDB" will be used interchangeably.) Following [2], we assume each domain has limited number of PDBs.

Inter-domain routing information is solely maintained by the eRS of each domain, independently of BGP-4. As shown in Figure 1, each eRS communicates and exchanges routing information with its peers via the same TCP session that is already established between BBs for resource reservation (see SIBBS [3] for detail). A routing information message basically includes *destID*, and *dest_serv*, *<destID, dest_serv>*. For a *destID*, there might be multiple paths, each of which represents a different *dest_serv*. A *dest_serv* includes the service ID *(servID)* and the *cost* of the service in addition to QoS parameters.

Maintenance of intra-domain network state: The iRS acts as a link state protocol peer with the nodes in the network and gets link state data as part of the protocol operation. As shown in Figure 1, each node sends its state directly to the iRS instead of flooding it to all the nodes in the domain.

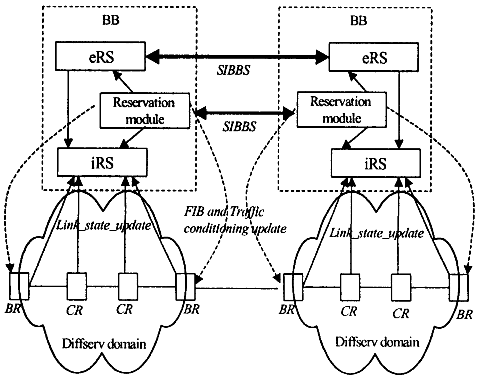


Fig.1. Diffserv QoS Architecture

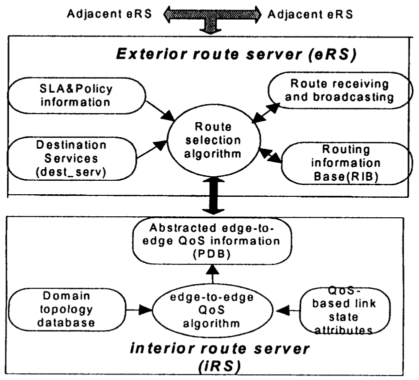

Adjacent eRS
Adjacent eRS
Exterior route server (eRS)
SLA&Policy information
Route receiving and broadcasting
Route selection algorithm
Destination Services (dest_serv)
Routing information Base(RIB)
Abstracted edge-to-edge QoS information (PDB)
Domain topology database
edge-to-edge QoS algorithm
QoS-based link state attributes
interior route server (IRS)

Fig.2: Functional decomposition of eRS

2.2 Overview

Based on the above assumptions and definitions, the RS model can be briefly described as follows: the iRS dynamically receives link utilization state from each node, computes the cost and available bandwidth (BW) of pre-established paths between each ingress-egress pair and then maintains this information in the abstracted edge-to-edge database (Figure 2). The eRS performs inter-domain route selection for each *<destID, dest_serv>* among the routes received from its peers by taking intra-domain network utilization into account. Note that the available BW in a route is not directly taken into account in route selection and it is not included in route advertisement. It is explicitly not the task of an eRS to locate routes that are guaranteed to have resources available at the time of route advertisement. This is motivated by the fact that even if available BW is included in route advertisements, it is not guaranteed to have available BW at the time of reservation because of dynamic network conditions.

Note that neither the iRS nor eRS algorithms affect existing reservations. They can be considered as modules that work in the background and maintain the databases as shown in Figure 2. In other words, they are not triggered by individual reservation requests. When a BB needs to make a reservation, it checks the abstracted edge-to-edge QoS database maintained by the iRS for the intra-domain admissibility test and RIB maintained by the eRS for selecting the next BB.

2.3 Edge-to-Edge QoS Routing Model

Quantitative Per Hop Behavior (PHB): We assume that each PHB is assigned to a certain share of link capacity and that each PHB can use only its share, no preemption. The surplus capacity of a PHB can only be used by best-effort traffic. With the exception of EF [1], all PHBs in Diffserv context define relative QoS assurance. To have *quantitative* QoS guarantees in a scalable manner, we associate an upper delay bound *d,* loss ratio bound *l,* and *cost,* *<d, l, cost>* to each PHB (e.g. $d < 3ms, l < 10^{-2}$). It is assumed that *d* and *l* are pre-determined at the stage of network dimensioning (configuration) [2,3,9], which is done over long time intervals (e.g., days, weeks). The cost is dynamically changed with link utilization.
PDB (edge-to-edge QoS) and Path: Similarly to PHB, PDBs are determined at the stage of network dimensioning [2,3,9]. A PDB is expressed by the sum of PHB constraints of all the nodes along the path. In offline, the iRS establishes a path between each ingress-egress pair, one for each PDB. Here, the path can be an MPLS tunnel or any other existing scheme. A path, in turn, a PDB is associated with delay bound *D,* loss ratio bound *L,* and cost value *COST, <D, L, COST>.*

Because of the scalability problem, it is clearly recommended that both PHB and PDB QoS constraints, for guaranteed services, be independent of network utilization [2]. To achieve this in a scalable manner, we introduce a *self-adaptive per-node provisioning* algorithm. The iRS installs PHB parameters (*d, l*) in each node. The algorithm, in each node, then dynamically monitors each PHB queue size and adjusts the scheduler rate and the buffer size for each PHB to meet pre-defined PHB constraints with respect to the traffic rate. It also calculates the current available BW and *cost* and sends them to the iRS. So, there is no need for a signaling scheme that configures each node along the path when a new reservation is granted. Due to communication overhead and the scalability concerns, we use a threshold-based triggering [4] of link state updates, where a new update is sent when the change in cost since the last update exceeds some predetermined threshold. By having the cost and available BW of each PHB in each node, the iRS dynamically computes the available BW and cost of each path (PDB) and stores this information in the edge-to-edge QoS database. Here, the cost is used by the eRS for route selection and the available BW is used by the BB for resource reservation. Although an iRS has detailed knowledge of its domain, it represents the domain in an abstract fashion to an eRS. The eRS sees the domain as if it consists only of border routers.

2.4 eRS Route Selection Algorithm

The eRS route selection algorithm is based on three main components, PDBs and their cost, destination services (*dest_serv),* and the route received from peers. As mentioned before, from the eRS point of view, PDBs and *dest_serv* can be considered as static components (modified at the stage of network dimensioning). The cost of PDBs and the route received from peers are dynamically changed with the network conditions. An eRS may have multiple peers from just a few up to hundreds, and each of them can be potential customers and providers. Upon receiving a route from its peer(s), it performs the following algorithm:

- Discard all the routes that failed in policy and SLA checks.
- Add PDBs to each *dest_serv(n)* received from peers, and compare with *dest_serv(1)*. All the results that satisfy *dest_serv(1)*'s requirements are selected as candidate routes.
- Store all candidate routes in RIB in the order of increasing cost, and send least costly one to peers.
- Repeat steps 2-3 for each *dest_serv(n)* (1,2...n).

2.5 Resource Reservation

By having the up-to-date intra-domain network state (available BW in each path) in edge-to-edge database and inter-domain routes in eRS's RIB, the resource reservation is very simple and straightforward. Upon receiving a resource reservation request (RAA), the BB first queries the eRS's RIB for possible next BB, which supports the given QoS requirements, and the associated egress router. It then queries the edge-to-edge QoS database for intra-domain resource availability. If both of the queries succeed, it sends RAR to next BB (see 3, 6 and 7 for details). It does not have to check individual links in its domain. Also, since the route in RIB has the cost associated with it, it chooses the least costly one among multiple possible routes.

3. CONCLUSION

In this paper we have presented a Route Server (RS) architecture that is compatible with the BB model for QoS routing. We addressed the role of RS in solving complex inter-domain routing in a scalable and efficient fashion. In particular, we first showed that how a Diffserv domain can provide quantitative QoS guarantees across its cloud and how a dynamic network state can be maintained in a scalable way. We also presented an inter-domain scheme that provides QoS route information to a possible destination region by taking inter-domain policies into account. Finally, we briefly describe the effeteness of RS in reservation state. In our future work we integrate RS with our existing inter-domain BB implementation.

References:

[1] K. Nichols et al. "A Two-bit Differentiated Services Architecture for the Internet" rfc2638

[2] K.Nichols, B. Carpenter "Definition of Differentiated Services Per Domain Behaviors" RFC 3086

[3] QBone Signaling Design Team, "Simple Inter-domain Bandwidth Broker Signaling (SIBBS)", http://qbone.internet2.edu/bb/

[4] G. Apostolopoulos et al. ``Server Based QoS Routing", Globecomm'99, Rio de Janeiro, Brazil

[5] P. Aukia et al. "RATES: A server for MPLS Traffic Engineering", IEEE network magazine, 2000

[6] H. Mantar, et al. "Inter-domain Resource Reservation via Third-Party Agent", SC 2001

[7] I. Okumus, J.Hwang, H. Mantar, S.Chapin, "Inter-domain LSP Setup Using Bandwidth Management Points", Globalcom 2001

[8] P. Auki et al. "RATES: A server for MPLS Traffic Engineering", IEEE network magazine, 2000

[9] P. Trimintzios et al.."An Architecture for Providing QoS in Differentiated Services " IM2001

SOFTWARE DISTRIBUTION FOR WIRELESS DEVICES

A reconfigurable approach

Gráinne Foley and Fergus O'Reilly
Cork Institute of Technology, Bishopstown, Cork, Ireland

Abstract: The dramatic rate of evolution and technology divergence from 2G to 3G and onwards broadens requirements for interoperability across software systems and wireless devices. These should interoperate seamlessly, transparent to users, who need only concern themselves with the final result. The challenge of providing transparent and reconfigurable content to wireless devices is addressed in this paper.

Key words: Wireless networks, Software distribution, Reprogrammable wireless devices

1. INTRODUCTION

Currently most European countries offer GSM and GPRS networks. Roaming on the GSM network is widespread and popular through a series of cooperating standards and agreements. The same system does not operate worldwide. In order to bridge the gaps and advance wireless telecommunications, 3rd generation (3G) networks were envisaged [1]. These networks require changes in network structure, user devices – hardware and software implementations. With the continued rapid development and rollout of wireless IP infrastructures and devices, the rate of obsolescence of user equipment is increasing dramatically. Currently 25% of handsets are replaced annually [2]. This trend will accelerate as new network architectures and technologies are introduced. Maturing these technologies will require many iterations and releases of software and hardware.

A system for software and device independence is explored here using Java. Java provides platform independence and is ideally suited to software delivery to divergent wireless devices over differing networks. It is predicted that Java will be present in 74% of wireless phones shipped in 2007 [3]. Java is implemented in the Services Archive (see below), making the solution presented Java end-to-end.

2. SERVICES ARCHIVE ARCHITECTURE

The Heterogeneous Services Archive illustrated in Figure 1 manages the distribution of software and firmware/hardware updates to wireless devices.

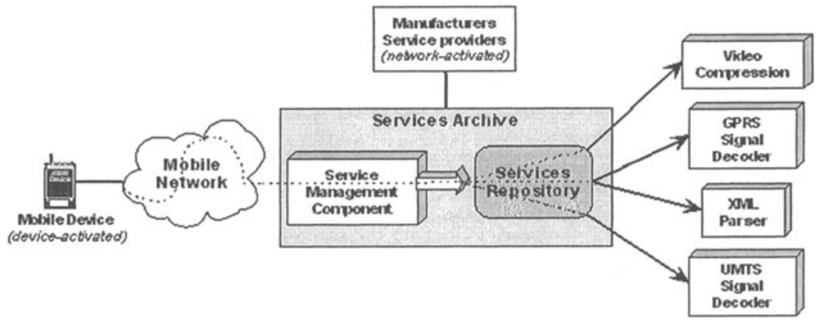


Figure 1. Heterogeneous Service Archive Architecture

User services may be device-activated, on a user service requirement basis, or network-activated, facilitating operators, manufacturers and service providers [4]. Thus, the mobile device must be able to support dynamic service downloading facilities, in addition to service execution. Two Java platforms are employed, the Java 2 Enterprise Edition (J2EE) in the Services Archive and the Java 2 Micro Edition (J2ME) on the wireless device [5].

3. WIRELESS DEVICE ARCHITECTURE

J2ME provides a hardware independent platform designed for limited memory footprint devices. It does this using the Connected Limited Device Configuration (CLDC) and the Mobile Information Device Profile (MIDP) [5], which respectively provide virtual machine features needed to support wireless mobile devices; and networking, user interface, persistent storage and application APIs. Code for the MIDP and CLDP resides in a vendor-customed Kilobyte Virtual Machine (KVM). It is suitable for devices with 16/32-bit RISC/CISC microprocessors/controllers, with as little as 512 KB of total memory available. The J2ME wireless device has a four-component architecture:

a) Management Component: Managing software and hardware components.
b) Services Component: Controls the execution of MIDP applications – MIDlets. A MIDlet, or a suite, is packaged in a Java Archive file (JAR) containing code and other resources. There may be many MIDlet suites on a wireless device.
c) Persistent Memory Component: For application and versioning data.
d) Hardware Component: Implemented using an Adaptive Computing Machine (ACM) [6]. The ACM provides advantages over other hardware technologies (DSP, ASIC and FPGA) enabling dynamic fast reconfiguration of evolving algorithms and standards without concerns of untimely obsolescence [7].

4. WIRELESS DEVICE UPDATES

Software updates are largely catered for within the J2ME specification. Updates to the underlying hardware are delivered from the Services Archive to the wireless device, Figure 2. Suitability is firstly verified by downloading a Java descriptor file.

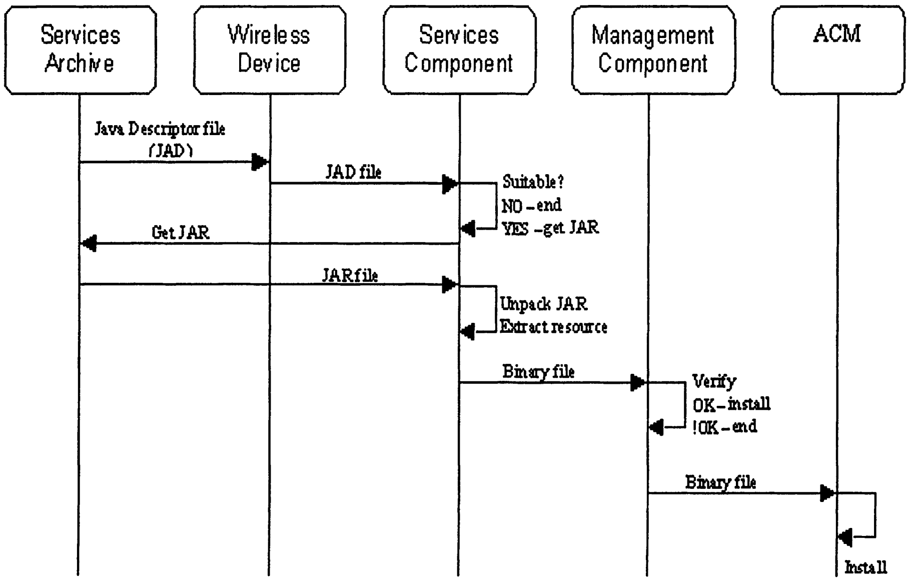


Figure 2. Hardware updates sequence diagram

The services component extracts an ACM resource (binary) from a JAR file. The management component then checks and installs the file, an MPEG-4 update for example. This mechanism requires operating system support and interaction but makes extendible, reconfigurable component based mobile devices a reality.

5. MEMORY AND PERFORMANCE EVALUATION

The MIDP specifies 32KB of run time memory for all Java applications. The statistics shown in Table 1 use the J2ME reference implementation.

Table 1. Execution statistics for MIDlet suites (bytes)

MIDlet Suite	Details	JAR file size	Executed byte codes	Dynamic objects	Heap Used
MyFileClient	Connect server, get file	3,266	120,182	43,664	35,020
Animation	Series- PNG images	12,690	336,718	64,488	36,080
WTK Demo	Show system properties	144,445	276,502	93,368	32,608

The JAR file is highly compressed for efficient delivery. It swells significantly at run time indicating that even very small suites may use considerable amount of memory. The amount of heap used for one MIDlet execution already exceeds 32KB in two cases. While manufacturers can optimize memory usage, they are already targeting total heap sizes in excess of 32KB. The Motorola iDEN phones specify a maximum JAR size of 50KB for one MIDlet suite, while the Nokia 3410 specifies 30KB.

Table 2. Typical download times for MIDlet suites with GPRS and UMTS

MIDlet Name	Function	Size (KB)	Transfer Time GPRS*	Transfer Time UMTS^
Nokia Tester	For testing Nokia phone features	5KB	0.19 sec	0.05 sec
Crossword Solver	Crossword puzzle solver	6KB	0.23 sec	0.06 sec
Live Weather	Graphical current weather	25KB	0.94 sec	0.25 sec
SoccerLeague	Online multiplayer soccer	88KB	3.3 sec	0.88 sec
Uemail	Mail for J2ME and WAP clients	164KB	6.2 sec	1.64 sec

* GPRS two channels – 26.8kbps ^ UMTS at likely 100kpbs throughput

From Table 2 we can see limits in Over-The-Air capacity may not significantly limit MIDlet suite size and functionality. If roaming from 3G to GPRS, performance rates are still reasonable. This is important, as JAR sizes will increase to deliver new services to devices.

6. CONCLUSIONS

In this paper we examined an overall architecture to provide transparent and reconfigurable content to wireless devices. The Services Archive manages the distribution of services activated from the network or from the device itself. The wireless device employs Java as a delivery mechanism for updates to software and firmware/hardware. Wireless devices, reconfigurable OTA, become more flexible and robust in a rapidly changing personal communications environment. This reduces the cost of technology evolution, an important consideration given the significant concerns over 3G-rollout expenditure [8].

REFERENCES

1. J. De Vriendt, P. Lainé, C. Lerouge, X. Xiaofang, "Mobile Network Evolution: A Revolution on the Move," *IEEE Communications*, Apr. 2002, pp 104-111
2. Nokia CEO, Jorma Ollila at Year End Strategy Update, December 3, 2002 Dallas Texas
3. Zelos Group Report "Wireless Java (Sept. 2002)" http://www.zelosgroup.com
4. Farnham, T., Clemo, et al.., 'Reconfiguration of Future Mobile Terminals using Software Download', IST Mobile Communications Summit 2000. pp.159- 168
5. J2EE and J2ME at http://java.sun.com/. Specifications at http://jcp.org/
6. Tuttlebee, W. (Ed.), *Software Defined Radio Enabling Technologies*, J Wiley & Sons Ltd, Chichester, England, 2002, pp. 272 - 288
7. Bing, B., Jayant, N. "A Cellphone for all Standards," *IEEE Spectrum*, May 2002, pp 34-39
8. "Ericsson sends shock waves across sector" Financial Times, 22/4/2002

VPDC: VIRTUAL PRIVATE DATA CENTER
A Flexible and Rapid Workload-Management System

Mineyoshi Masuda, Yutaka Yoshimura, Toshiaki Tarui, Toru Shonai, and Mamoru Sugie
Hitachi, Ltd., Central Research Laboratory

Abstract: Rapid server allocation implemented on Virtual Private Data Center (VPDC), which is an autonomous server allocation system for a three-tier web system, has been developed and tested. The test results show that with this new system elapsed time for application server allocation is about 20 seconds, and that for database server allocation is 140 seconds.

Key words: autonomous, server, allocation, rapid

1. INTRODUCTION

Autonomous server-resource management[1][2] is one of the key technologies for current information systems. Server resource management allocates or de-allocates servers in response to fluctuations of system workload. This technology can maintain adequate system capacity; thus, cost performance improves especially in the case of a web transaction system, in which system workload fluctuates frequently.

Adaptive server allocation is one of the key issues concerning easy managements of complex systems. Furthermore, *rapid* server allocation is another important issue. In particular, rapid server allocation for server-resource management has two advantages:

(1) adaptation to "Slash Dot Effect" (a phenomenon in which a web system receives tens times more accesses than usual in a short time) ,

(2) improvement in server utilization (because of reduction in extra server resources which act as a "buffer") .

Focusing on a rapid sever allocation, we have developed Virtual Private Data Center (VPDC), an autonomous management system for a three-tier web system. The present study showed that the rapid server allocation implemented on this VPDC system is effective.

2. RAPID SERVER ALLOCATION

System Overview

VPDC is a hosting utility in a data center; it manages several customer systems hosted in a server farm. Figure.1 shows an overview of the VPDC management system. It is composed of a single control server and agents working on each server (i.e., on both allocated and spare servers). The system has three main functions.

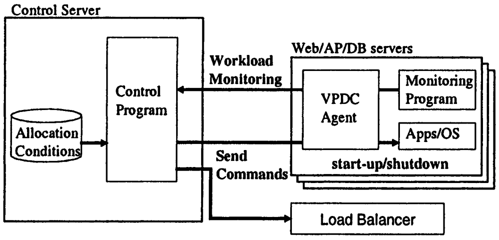


Fig. 1 VPDC management system

1. **Monitoring:** The agents on all allocated server periodically obtain load information about a server (i.e., CPU, network, disk utilizations, etc) and the control server gathers them.
2. **Decision making:** The control server analyzes load information in accordance with a predefined allocation policy for each customer and makes server allocation/deallocation decisions.
3. **Action:** The agent receives a set of commands for start/stop application, and then the control server changes the configuration of a load balance group.

Implementation

For rapid server allocation, the following two approaches are implemented on the VPDC.

(1) Cutting out data copy

Figure.2 shows typical and VPDC server allocation sequences (the length of each rectangle means a time cost). In the first and the third parts in the typical server allocation [shown in Fig.2 (a)], data copy to a local server takes a long time (several minutes). In VPDC server allocation, such data copying is cut out by the following methods.

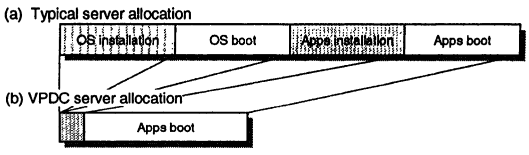


Fig. 2 VPDC rapid server allocation parts

- OS installation is not executed during a server allocation time. OS is pre-installed, and for the purpose of security maintenance, the OS is initialized during a server deallocation time.

  Shared file server is used for deploying application programs.

The VPDC server allocation parts are shown in Figure.2 (b). The total server allocation time is considerably shortened and the application program start-up time dominates.

(2) Eager allocation and lazy deallocation

The control server makes the decision to allocate servers as soon as it detects the signs of a slash-dot effect. This policy might make mistakes if, for example, the workload returns to a normal level in a short period. However, this policy is adapted, since VPDC gives preference to avoiding late server allocation in the case of the slash-dot effect. In contrast, to prevent repeated server allocation and deallocation in a ping-pong fashion, a server deallocation decision is made only after the workload has stayed low for a certain time.

3. EVALUATION

Prototype System

Figure.3 shows the construction of the VPDC prototype. The NFS server provides application programs and contents data. The VPDC agent software has been developed as a SNMP agent to which VPDC-specific MIB modules are installed. Agents run on all servers including spare servers. Apache runs on the web servers, Tomcat with a hand-maid session handover runs on the AP servers, and Oracle Parallel Server (OPS) runs on the DB servers.

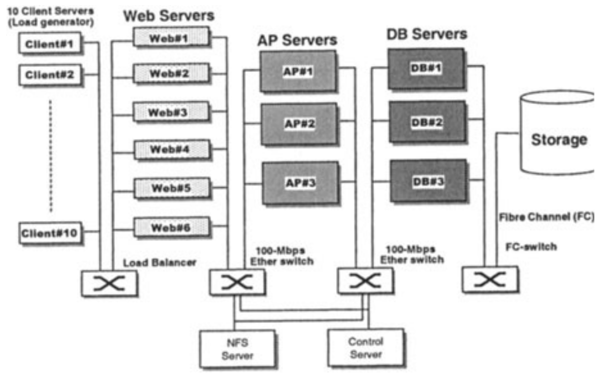


Fig. 3 VPDC prototype

Results

Experiments on the VPDC prototype are performed to confirm the effectiveness of VPDC rapid server allocation. The test program running on the prototype is a subset of TPC-W[3]. System workload is increased in order to load the DB and AP servers so that server allocations are triggered.

Figure.4 shows relative amount of accesses to the prototype, average response time measured on a client, CPU utilization of allocated servers, and elapsed time for server allocation per layer. Each vertical line means an event time such as a server allocation. These results show that elapsed time for the AP server allocation is about 20 seconds. This is because the control server takes 10-15 seconds to detect the sign of the slash-dot effect. On the other hand, the time for the DB server allocation is about 140 seconds. This is because DB application initialization takes a long period before the DB server is ready to receive requests. After the DB-layer bottleneck is removed, the AP-layer is able to send a larger amount of requests to the DB-layer. Thus, the control server allocates server AP#2.

4. CONCLUSION

A rapid server allocation implemented on VPDC was developed and evaluated. Due to server allocation without a disk copy, in the application layer, VPDC improves the site's response time to within approximately 20 seconds after the start of the slash-dot effect. Workload management of the DB layer improves the site's response time to within 140 seconds. Elapsed time for the AP server is rapid enough, but that of the DB server is rather slow because of the start-up time of the DB application.

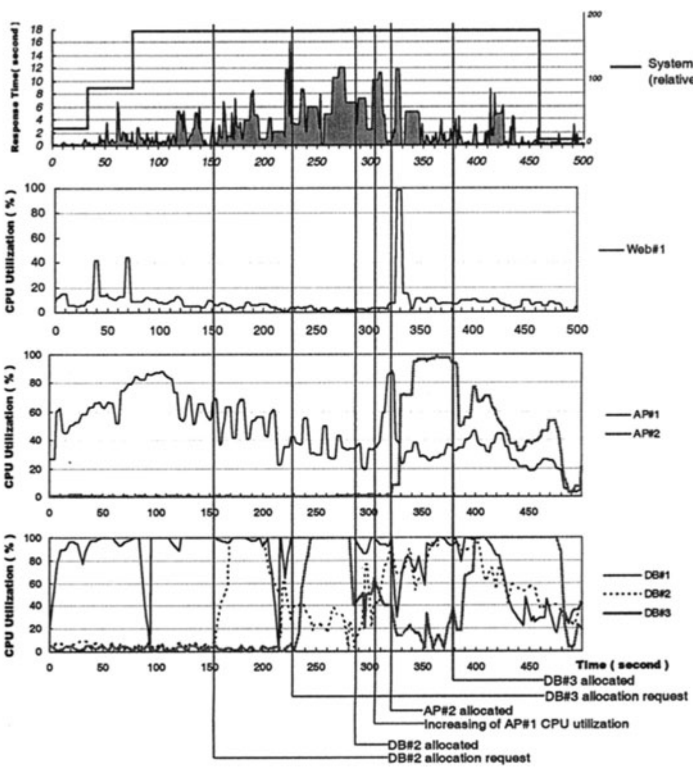


Fig. 4 Relationship between workload, response time, and server utilization

ACKNOWLEDGEMENTS

We would like to thank all the people who supported the VPDC development.

REFERENCES

[1] K. Appleby et al.: Oceano SLA Based Management of a Computing Utility, IEEE International Symposium on Integrated Network Management 2001, (2001)
[2] L.L. Fong et al.: Dynamic Resource Management in an eUtility, NOMS2002 (2002)
[3] "TPC-W" http://www.tpc.org/tpcw/default.asp

X-CLI : CLI-BASED MANAGEMENT ARCHITECTURE USING XML

Byung-Joon Lee, Taesang Choi and Taesoo Jeong
{bjlee,choits,tsjeong}@etri.re.kr, Internet Traffic Management Team, ETRI, Gajung-Dong, Yusong-Gu, Daejeon, Republic of Korea

Abstract: As Internet technology becomes more complex, the policy information for managing the Internet grows beyond the capability of a simple protocol like SNMP. IETF suggested COPS as an alternative, but it has not been widely accepted. For that reason, many administrators have developed network management systems which control network devices using CLI, but systems based on CLI have a maintenance problem: when the syntax of CLI changes, the implementation of the system must be modified. In this paper, we suggest X-CLI as a solution for this problem, and describe its design principles.

Key words: Network Management, CLI, XML, X-CLI, API

1. INTRODUCTION

Traditionally, SNMP has been a major management protocol for the IP network because of its simplicity. But for the new technologies such as MPLS, VPN or QoS, the policies for managing network have become too complex. IETF standardized COPS to cope with that complexity, but it is not being widely accepted.

For that reason, systems implemented on CLI (Command Line Interface) of the network devices are being widely used. Those systems translate a policy into a sequence of CLI commands, and send those commands to the devices using the TELNET protocol. However, those systems are dependent on the syntax of the CLI command: the syntax change of the CLI commands results in the implementation modification of the policy-to-CLI conversion logic.

In this paper, we suggest X-CLI (XML wrapper API for CLI) as an alternative to SNMP, COPS, and traditional CLI-based solutions. It is based on the concept of XML template which represents a group of CLI commands in a hierarchical manner. The template is converted into the actual CLI commands, and sent to the network devices by the X-CLI API interfaces.

2. XML TEMPLATE

The concept of 'XML template' corresponds to the concept of 'function' in general programming languages like C/C++: it maps a set of CLI commands to some specific configuration action. But the XML template differs from 'function' because it exists as a file, and designed to be able to represent hierarchical dependency, argument dependency and result dependency which are the basic characteristics of the CLI commands of most of the current network devices [1][2].

2.1 The Characteristics of CLI commands

Figure 1 shows the configuration steps for setting up an IBGP session between PE routers for the dissemination of the route information in the VRF table.

```
(config)# router bgp 55555
(config-router)# address-family ipv4 vrf VRF-SEOUL
(config-router-af)# neighbor 203.255.25.15 remote-as 5555
(config-router-af)# neighbor 203.255.25.15 activate
```

Figure 1. BGP configuration steps for VPN at CISCO PE router

As shown in the figure, the CLI commands are executed in a hierarchical manner. For example, the command "address-family ipv4 vrf VRF-SEOUL" cannot be executed without the successful execution of the command "router bgp 55555." We call this kind of behavior 'Hierarchical Dependency.'

In the command "router bgp 55555," the '55555' is the required argument of the CLI command "router bgp <as-number>". Without the argument, the CLI command cannot be executed. We call that kind of behavior 'Argument Dependency.'

As a result of a CLI command execution, one of the three following situations can happen: (1) CLI execution error (2) request more input from the administrator (3) successful execution. When (1) happens, most of the commands scheduled to be given to the devices cannot be delivered. For case (2), every scheduled command hangs until the additional input is given by the administrator. We name this kind of conditional behavior 'Result Dependency.'

2.2 XML Representation of the CLI Commands

As shown in the example XML template of Figure 2, an XML template is the hierarchy of the <cli></cli> XML tags. A containment relationship between <cli> tags represents a hierarchical dependency. Other kinds of the dependencies are represented by the attributes of the <cli> tag.

The attribute 'command' has CLI command string as its value. To represent the 'argument dependency', the command string can contain formal argument names which start with special character '$.' Parenthesis can be used with formal argument names to specify the optional part of the CLI command. The thorough description of the command string syntax can be found in [2].

```
<cli prompt1="#" command="config terminal" errorstr="^">
  <cli tag="bgp1" prompt1="#"
     command="router bgp $asnum" errorstr="^">
    <cli>
      <cli tag="bgp2" prompt1="#"
           command="address-family ipv4 vrf $vrfname"
           errorstr="^">
        <cli>
          <cli tag="bgp3" prompt1="#"
               command="neighbor $ipn1 remote-as $nasnum"
               errorstr="^">
          </cli>
          <cli tag="bgp4" prompt1="#"
               command="neighbor $ipn2 activate"
               errorstr="^">
          </cli>
        </cli>
        <cli prompt1="#" always="true" command="exit"></cli>
      </cli>
    </cli>
    <cli prompt1="#" always="true" command="exit"></cli>
  </cli>
  <cli prompt1="#" always="true" command="exit"></cli>
</cli>
```

Figure 2. XML template for Figure 1

The attribute ‘errorstr’ and ‘always’ express the result dependency. When a reply of the network device contains the value of the attribute ‘errorstr’, it is considered as an error. The attribute ‘always’ is a flag indicating that the CLI command can be executed in spite of the execution failure of the previous CLI command.

The attribute ‘tag’ is introduced for uniquely identifying <cli> tag. The prompt attribute ‘prompt1’ and ‘prompt2’ are needed to send a CLI command to the network device using TELNET. A client starts sending a CLI command to the server when the value of the attribute ‘prompt1’ is received from the server, and stops receiving replies from the server until the value of the attribute ‘prompt2’ is received. If not specified, the value of the attribute ‘prompt2’ is the same as that of ‘prompt1.’

A <cli> tag with no attribute is called a PAT (Pure Aggregation Tag). A PAT specifies that the enclosed <cli> tags can be converted into actual CLI commands repeatedly. The <cli> tags not enclosed by the PAT can only be converted once.

3. X-CLI API

X-CLI API is the interface for manipulating XML template. It provides functionalities for loading an XML template, ‘materializing’ it, and sending it to the network device. The ‘materialization’ is the process of converting an XML template into a sequence of CLI commands.

After loaded into the memory by the X-CLI API, a template is translated into the internal data structure of a tree topology. This tree is materialized by traversing it with the arguments given by the X-CLI application programmer [2]. This process is similar to the act of passing arguments to a function which generates some specific control flow. The materialization result of the Figure 2 is shown in the Figure 3. The arguments ‘55555,’ ‘VRF-A-Seoul,’ ‘203.255.255.13,’ ‘5555’ and ‘203.255.255.13’ are delivered to the XML template in sequence.

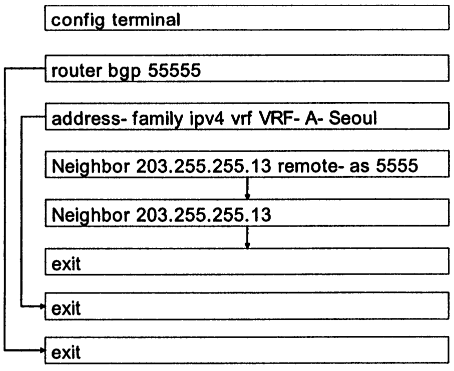

config terminal
router bgp 55555
address- family ipv4 vrf VRF- A- Seoul
Neighbor 203.255.255.13 remote- as 5555
Neighbor 203.255.255.13
exit
exit
exit

Figure 3. Materialized result of Figure 2

The materialized commands are sent to the network device sequentially. When an error happens, the failure branch target (depicted as an directed edge in Figure 3) is taken. After the branch, only commands which have 'true' value for the attribute 'always' can be sent to the device.

X-CLI API is greatly enhanced recently for the device monitoring. The <cli> tag is extended to express the monitoring actions, and the monitoring result is parsed automatically by the X-CLI API [2]. This feature aids the programmers who want to develop an application which monitors network device statistics using CLI.

4. CONCLUSION

X-CLI enhances the maintainability of the network management software by eliminating a dependency on the syntax of the CLI command from the software. The CLI syntax only exists in the XML templates. And besides, because the X-CLI API uses TELNET as its communication protocol, it can be easily secured by the SSH protocol. Adding support for the SSH to the X-CLI API can be easily done.

X-CLI API is being integrated with Wise<TE> [3], which is the network management server for traffic engineering, QoS and VPN. Wise<TE> is being tested against the several tens of the routers of Juniper and CISCO. The architecture of the Wise<TE> and the details about integration with X-CLI API is described in [2].

REFERENCES

[1]Byung-Joon Lee, Taesang Choi and Taesoo Jeong, "X-CLI: CLI based Policy Enforcement and Monitoring Architecure using XML", APNOMS 2002.

[2]Byung-Joon Lee, Taesang Choi and Taesoo Jeong, "X-CLI: CLI based Management Architecture using XML", Technical Report, 2002. http://oopsla.snu.ac.kr/~bjlee/document/x-cli/x-cli-techreport-2002.doc

[3]TS Choi, SH Yoon, HS Chung, CH Kim, JS Park, BJ Lee, TS Jeong, "Wise<TE>: Traffic Engineering Server for a Large-Scale MPLS-based IP Network", NOMS 2002

A DYNAMIC SNMP TO XML PROXY SOLUTION

Ricardo Neisse, Lisandro Zambenedetti Granville, Diego Osório Ballvé, Maria Janilce Bosquiroli Almeida, Liane Margarida Rockenbach Tarouco
Federal University of Rio Grande do Sul - Institute of Informatics
Av. Bento Gonçalves, 9500 - Bloco IV - Porto Alegre, RS - Brazil
{neisse, granville, dob, janilce, liane}@inf.ufrgs.br

Abstract: The network management area has some proposals to use XML to encode information models and managed object instances. In this paper we present a solution to dynamically create SNMP to XML proxies using a SAX parser and the translation facilities from the libsmi tools. We also present an analysis system that uses the management information provided by the proxies in XML.

Keywords: Web-based Network Management, SNMP, HTTP, XML, XPath

1. INTRODUCTION

The information used to manage computer networks are tipically defined according to some rules (e.g. SMIv2, SPPI, XML), and retrieved using some protocol (CLI, SNMP, COPS, HTTP). Currently, an important problem is that the set of different options for the definition of management information and protocols increases the complexity of managing a network, since there is no consensus in a single definition language and protocol.

If a unique definition language could provided (e.g. SMIng [1]) and accepted, the other problem will still remain: which unique protocol should be used? In our view, this question is unsolvable because we believe that several different protocols will be still required to manage older devices. However, from the network administrator point of view, the lack of consensus on a single protocol should not refrain the use of a single representation of the retrieved information. To allow that, protocol and information representation translations is needed.

Although the SNMP is the de facto TCP/IP management protocol, its management information is defined through SMIv1 or SMIv2, which is not suitable when we are searching for a common representation. XML, however, seems to be more appropriated, besides being already addressed by the SMIng working group. We developed a system that automatically generates SNMP/XML proxies that reside in HTTP/HTTPS servers. The proxy generating system receives a SMIv1 or SMIv2 MIB definition as source parameter and creates a PHP4 script file that is the proxy itself. The just created proxy can then contact a target device via SNMP and generates a XML-based result. We have used the `libsmi` [2] package to support the generation of the XML files, and the `expat` package to provide the PHP4 support for SAX (Simple API for XML). We have validated the proxy system through its use in a RRDTool-based [3] monitoring front-end.

2. ARCHITECTURE AND IMPLEMENTATION

Figure 1 shows, how a proxy operates **after** its creation. A network management station (NMS) retrieves information throughout a SNMP/XML proxy hosted by a HTTP/HTTPS server. Each server can hosts several proxies, and the selection of which proxy should be used in done in the URL passed from the NMS to the server. Additionally, the selected proxy receives the address of a target device and an SNMP valid community that are used to access the target device via SNMP. Normally, one single access to a proxy generates several SNMP accesses to the target device, mainly when the information to be retrieve is stored in MIB tables. After the SNMP information is retrieved from the target device, the proxy compiles such information into a single XML and sends it back to the NMS.

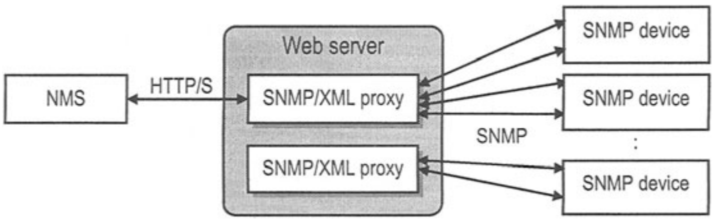


Figure 1. SNMP/XML Proxy operations

Comparing the amount of management information found in the NMS/proxy interactions, it is fewer than the amount of management information found in the proxy/target device interactions. Thus, pushing SNMP/XML proxies closer to the managed devices will reduce the overall amount of management traffic. Also, since we based our implementation in the `smidump` tool, the XML returned to the NMS contains not only the value associated to the management information, but also the whole description of such information originally defined in SMIv1 or SMIv2, allowing a new NMS to discover these definitions on demand.

The proxies are implemented as PHP4 scripts. New MIBs could be supported only through the development of new PHP4 proxies. With the great variety of available MIBs, creating new PHP4 scripts every time a new MIB is required would be a quite slow process. To solve that, we have automated the processes of creating new proxies in our solution.

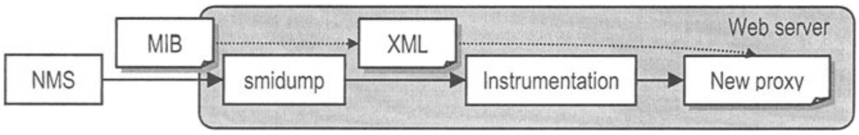


Figure 2. Architecture for SNMP/XML proxy creation

Figure 2 presents the steps to create new PHP4 SNMP/XML proxies. First, a SMIv1 or SMIv2 MIB is uploaded to the server that will host the new proxy. Inside the server, the `smidump` checks the passed MIB and if no errors are found it generates an XML temporary file. This file is then instrumented adding PHP4 code that can contact SNMP-enabled devices. The proxies is then stored in a standard directory in the server, as well as the original MIB (for documentation purpose) and the XML intermediate file.

3. ANALYSIS TOOL

We have also developed an XML analysis tool that uses the SNMP/XML proxies. We have used the RRDTool [3] to store performance data and the MySQL to store configuration data. Basically, the tool is a generic monitor that collects XML files addressed in URLs. Any information available in XML can be monitored, which includes, obviously, the SNMP data indirectly provided by the SNMP/XML proxies.

The tool is also based on Web technology and accessed through HTTP/HTTPS. The network administrator defines which information should be monitored, and which proxies have to be used. Other information required is the IP of the target device, the SNMP community string and an XPath expression which locates, inside the retrieved XML, the specific information to be analyzed. All this configuration data is then stored in the MySQL. For example, the configuration data required to monitor the incoming traffic in the interface 2 of the IP 200.132.73.54 throughout the interfaces.xml.php proxy hosted by noc.metropoa.tche.br are:

Target device:	200.132.73.54
SNMP/XML proxy:	interfaces.xml.php
SNMP comunity string:	public
Proxy Web server:	noc.metropoa.tche.br
XPath expression:	val[@oid="interfaces.ifTable.ifEntry.ifInOctets.2"]/@value

Figure 3 presents one possible configuration for the analysis tool and a proxy interaction. In this case, both analysis tool and the proxy are located within the same server. Due to this configuration there are no network traffic overhead between the proxy and the analysis tool.

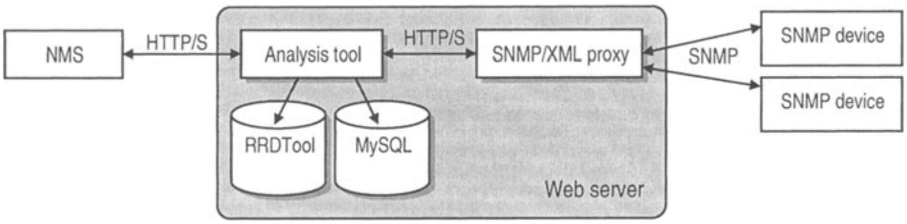
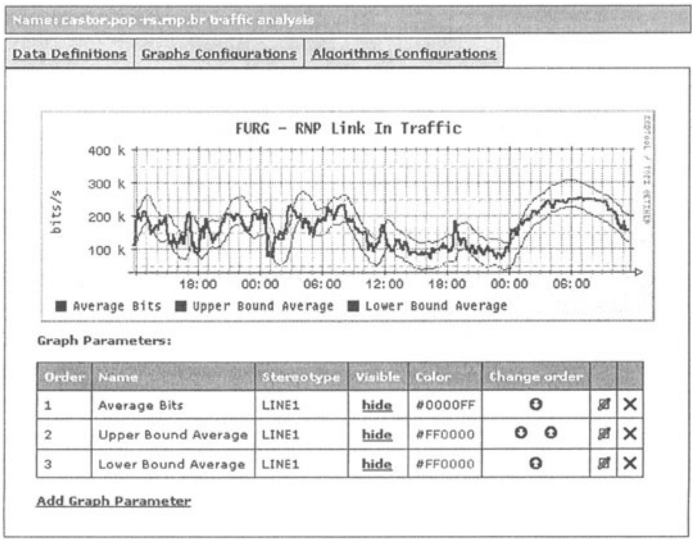


Figure 3. Analysis tool accessing an SNMP/XML proxy

Figure 4 presents a real traffic data analysis generated through the Aberrant Behavior Detection (ABD) [4] algorithm of a university campus link in the Brazilian National Research Network backbone. The thick line is the observed value of the incoming traffic and the thin lines are min and max bound values (confidence band).



Graph Parameters:

Order	Name	Stereotype	Visible	Color	Change order		
1	Average Bits	LINE1	hide	#0000FF	●		✕
2	Upper Bound Average	LINE1	hide	#FF0000	● ●		✕
3	Lower Bound Average	LINE1	hide	#FF0000	●		✕

Figure 4. Analysis tool snapshot for the Anomalous Behavior Detection

4. CONCLUSIONS AND FUTURE WORK

We presented in this paper a dynamically SNMP/XML proxy creating solution that produces SNMP/XML proxies from standard SMIv1 or SMIv2 MIBs. Since the created proxies reside inside Web servers, they act as intermediate managers that uses SNMP to retrieve management information and generates XML document as a result.

We have also presented the monitoring tool that uses the SNMP/XML proxies to analyze the network behavior. Proxies and the management tool could be located into a different device, differently from the example presented in figure 3, and no modifications are need to the architecture, as the access to the proxy is done through HTTP/HTTPS and, therefore, it is transparent to the analysis tool the physical location of the proxy.

One improvement for the SNMP/XML proxy is the implementation of a filter that would receive and XPath expression as an extra parameter in order to specify only the specific data that should be fetched and transferred to the management application. This would reduce the traffic between the NMS an the SNMP/XML proxy and also will reduce the processing overhead in the target device.

REFERENCES

[1] F. Strauss, J. Schoenwaelder. SMIng - Next Generation Structure of Management Information, draft-ietf-sming-02, July 20, 2001.

[2] F. Strauss. Libsmi - A library to access SMI MIB information, http://www.ibr.cs.tu—bs.de/projects/libsmi/.

[3] Oetiker T. Round Robin Database Tool (RRDTool) http://www.rrdtool.org

[4] Brutlag, J. D. Aberrant Behavior Detection in Time Series for Network Monitoring, Proceedings of the 14th Systems Administration Conference (LISA 2000) New Orleans, Louisiana, USA December 3-8, 2000.

INTERACT-DDM

A Solution For The Integration Of Domestic Devices On Network Management Platforms

A. E. Martínez, R. Cabello, F. J. Gómez, J. Martínez
Escuela Politécnica Superior. Universidad Autónoma de Madrid.
28049 Cantoblanco. Madrid. Spain. Phone: 34 91 348 2338

Abstract: This paper presents *Interact-DDM*, a solution that integrates domestic devices with traditional computer networks. The architecture proposal is based on TCP/IP network management standards: SNMP protocol and Management Information Bases, MIB. The centralized management operation has been enhanced with additional capabilities integrated on the agents. The design has been performed permitting a very flexible device definition and dynamic configuration. This is achieved by the *meta-definition* of devices in the system MIB. A laboratory experiment has been deployed to check and validate the design proposed, where multiple configurations have been tested, and the design modularity has been proved.

Key words: Ubiquitous Computing, Domestic Networks, SNMP management.

1. INTRODUCTION

The evolution of the electronics industry gives the possibility of inserting programmable control devices into home appliances at an affordable price. This simplifies their design and enhances their functionality [1], allowing their integration on a *domestic appliance network*. Additionally, their integration into a standard communication network opens a new scope of possibilities [2]. Some work have been done to provide this connection, using new communication architectures [3][4], or integrating the appliances into IP networks [5][6]. Some standards are emerging on this area, such as the Universal Plug and Play, the VESA Home Network initiative, and the Open Services Gateway Initiative. These architectures present a complete solution, well suited for complex devices, but expensive for low-end devices, which are important in the domestic environments.

The present work describes *Interact-DDM (Interact - Domestic Device Management)*, a solution for doing this integration that is based on IP network management standards and SNMP. It is built according to three basic principles: The meta-definition of the devices being controlled on the system MIB; the existence of a single focal point to act as a gateway between the domestic network and the user interface applications; and the autonomy of the device agents to perform some simple functions without the intervention of the central management application.

Interact-DDM is being used on the *Interact* project, which is being developed by a multidisciplinary research team at the Computer Engineering College of Universidad Autónoma de Madrid [8]. The project objective is to develop the technology to implement an intelligent domestic/office environment (*domotic environment*), where the user interaction with the system is performed in a natural way (speech, signs, actions...), taking into account the context of the task.

2. ARCHITECTURE AND DESIGN

Interact-DDM is based on SNMP. We consider SNMP a very good approach to solve the network management problem, especially in domestic environments, for the following reasons. First, SNMP is *simple*; SNMP agents are simpler than, for example, web servers, that are used in other domestic solutions [7]. Simpler agents reduce the total cost of the solution. Second, SNMP agents can send notifications to the clients if a special situation is detected, while other solutions (such as web servers) can't. Third, UDP datagrams used by SNMP as transport mechanism are well suited to communicate in a domestic network, where the media reliability is very high; connection oriented protocols are not needed. Fourth, the polling mechanism used by SNMP, that in some applications can consume a broad communications bandwidth, is not a problem in domestic environments, where the interactions with the system will be mainly done as results of human actions: the frequency of SNMP messages will be low. And finally, SNMP allows the integration of domestic devices in a more complex network management platform.

The basic system structure is presented on Figure 1. Devices and agents are connected to a Local Area Network, that is the domotic communications backbone. Depending on the location or specific characteristics of each device, an agent manages a single device or a group of devices. The agent and all the devices that are under its control are called an *Area*. Each area has its own IP Address.

The *Central Point of Control* (CPC) is the manager application. It interacts with the agents to query or modify the status of the devices. CPC can be used to control the devices by itself, or as gateway by Intelligent User Interface Applications to link to the *real world.*

The managed objects of an area are contained in the *Interact MIB*. The structure of the MIB is common for all the agents on the system, independent of the devices attached to it. The MIB has been defined to provide a very high level of flexibility on the definition of the environment. In the case of devices, for example, the MIB contains not only the value of the different parameters or attributes that define the status of a device, but also *the definition of the device*. In this way, the MIB can store any kind of device. Additions of new device types can be done without modifying either the Agent or the CPC. Only the User Interface applications must know how the device operates and the meaning of its attributes. This information is stored in the MIB. The MIB contains the *meta-definition* of the managed objects, in addition to the objects themselves.

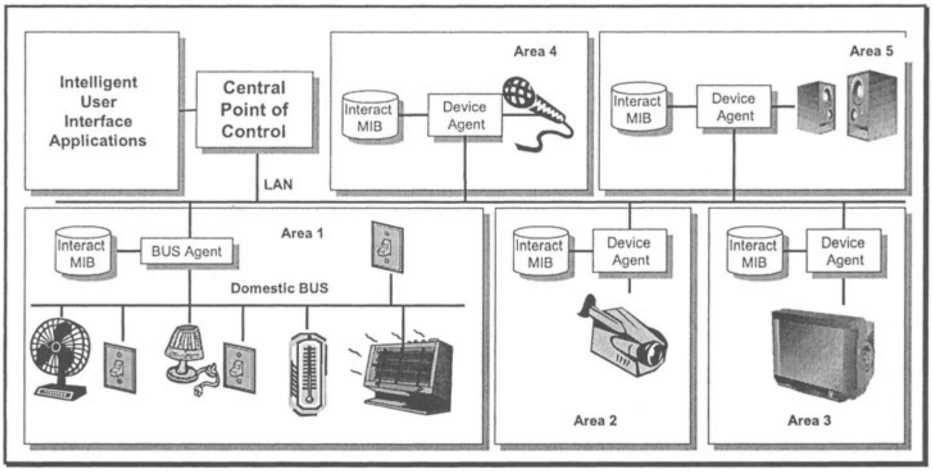


Figure 1. Basic System Structure

Agents have additional capabilities. They allow the definition of groups of devices and connections between them. Devices can be dynamically assigned to belong to a group. The whole group is then seen by the CPC as a single entity. When the value of an attribute is set for a group, the agent spreads this change to all the devices on the group. This reduces the workload on the CPC and the network traffic. A connection is a link between a sensor device that can receive an interaction from a user, and another device whose status can be changed remotely. An agent detects the user's interaction in a sensor, locates target devices or groups that should be informed of this modification, and sends them the appropriate command to modify their status. Agents send remote request to other agents to perform this function.

3. EXPERIMENTAL LABORATORY

To evaluate the proposed architecture and to prepare the MIB definitions for the devices that will be available at domotic environments, an experimental laboratory has been deployed. A World Wide Web access to the basic functionality and status of this laboratory is available [8]. The backbone of this laboratory is a 100 Mbps Ethernet network. Devices are connected directly to it or through a computer or an embedded computer. Devices currently connected are microphones, speakers, TV capture boards, VCRs, CD or DVD players, video screens, personal card readers, and simple domestic devices (lamps, switches, electronic locks, presence detectors, etc.) connected using the *European Installation Bus, EIB*, which is attached to the

Ethernet network through a gateway built on an embedded computer. An agent manages each area, and access to the devices by means of an Application Programming Interface (API), that is common for all the devices. For each device, an interface module has been developed to implement this API, converting standard calls into commands that are understood by the device. All the communications over the LAN have been implemented using TCP/IP protocols. Streaming services have been implemented using multicast transport technology.

4. CONCLUSIONS AND FUTURE WORK

Interact-DDM has an architecture that allows a dynamic definition and access to networked appliances, compatible with network management standards. As proof of concept, an experimental laboratory has been built. This experience has demonstrated that this architecture provides an easy method to perform the remote control of the domestic devices. Also, it can be considered the base for more complex applications developed to improve the man-to-machine interaction.

ACKNOWLEDGEMENTS

This work is partially funded by the Spanish Government on the project number TIC2000-0464.

REFERENCES

[1] Badami, Vivek V. and Chbat, Nicolas W., "Home appliances get smart", *IEEE Spectrum*, Vol. 34 No. 8, August 1998, pp. 36-43.

[2] Saif, U, Gordon, D, Greaves D.J. "Internet Access to a Home Area Network", IEEE Internet Computing, Vol. 5, Iss.1, January-February 2001, pp 52-63.

[3] Minoh, Michihiko and Kamae, Tak, "Networked Appliances and Their Peer-to-Peer Architecture AMIDEN", *IEEE Comm. Magazine*, Vol. 39, October 2001, pp. 80-84.

[4] Koon-Seok Lee, Hoan-Jong Choi, Chang-Ho Kim and Seung-Myun Baek, "A new control protocol for home appliances – LnCP", Proceedings of ISIE 2001, IEEE International Symposium on Industrial electronics 2001, Vol.1, 2001, pp.286-291.

[5] Kinne, A., "EIB IP Connectivity", Proceedings of the EIB Event 2001, Oct. 2001. URL: http://www.emt.ei.tum.de/eiba/Conference_01/Proceedings/04.pdf

[6] Desbonnet, J. and Corcoran, P.M., "System Architecture and Implementation of an Internet/CEBus Gateway", *IEEE Transactions on Consumer Electronics*, Vol. 43, Nov. 1997, pp.1057-1062.

[7] Riihijärvi, Janne, et. al., "Providing Network Connectivity for Small Appliances: A Functionally Minimized Embedded Web Server", IEEE Communications Magazine, Vol. 39, October 2001, pp. 74-79.

[8] http://odisea.ii.uam.es/

SESSION 9

Information Modelling

Chair: Alexander Clemm
Cisco Systems, USA

AN SMING-CENTRIC PROXY AGENT FOR INTEGRATED MONITORING AND PROVISIONING

Emmanuel Nataf, Olivier Festor, Guillaume Doyen
The Madynes Research Team
LORIA - University of Nancy 2 - INRIA
615, rue du Jardin Botanique
54602 Villers-lès-Nancy, France
Surname.Name@loria.fr

Abstract: The combined use of SNMP and policy based frameworks is growing very fast. From both an information model and programmation interface point of view, an integrated view is highly desirable. Several approaches have been proposed so far in the information model sphere. In this paper we present the experience gained in using the SMIng approach for building an integrated management environment that provides seamless integration of both policy provisioning and MIB monitoring. The developed management environment has been deployed for managing an active network called FLAME, dedicated to dynamic IP monitoring.

Keywords: SMIng, management platform, active networks, integration

1. INTRODUCTION

During the last few years, many attention has been brought to novel information modelling techniques, management services and protocols. Results from these efforts are very different in nature mostly because they emerge from different communities with very different requirements (e.g., backward compatibility with a legacy approach or technology conformance). As part of the Operations and Management area of the IETF, a working group was created back in 2000 to address the issue of the evolution of the SNMP Structure of Management Information Version 2 (SMIv2) [11, 12, 10]. Part of this evolution is also concerned with a subset of the policy management framework related to provisioning (COPS-PR [3]) and the associated policy information specification technique (SPPI [9]).

While working on a management framework for an active network environment called FLAME, we faced the need to deal within management applications with both management policies in a provisioning framework and monitoring services relying on SNMP. One of the goals we had in mind, when looking at possible solutions was to provide an unique view of objects to the management applications. Thus, we naturally looked at SMIng *Next Generation Structure of Management Information* which provides a neutral object model and offers mapping facilities to both SNMP-SMI and COPS-PR SPPI.

FLAME is an active network based architecture developed within our research group dedicated to the hosting of IPv6 monitoring and management services. It includes a BSD native execution environment which can host active applications, a set of servers from which active applications can be downloaded and a management environment dedicated to the configuration and monitoring of the active management framework itself. The management interface of a node is divided into three parts: a command line interface for configuration and activation purpose, an SNMP agent implementing all objects of MIB-II together with objects dedicated to the active node monitoring, and a policy enforcement point (PEP) through which the node configuration is downloaded and activity policies enforced. This environment has been used for various management purposes among which multicast monitoring [13].

In this paper we share the experience gained with SMIng at both management information specification and mapping levels, as well as at a programmatic and engineering level within a management platform. At the information modelling level, we both performed reverse engineering from SPPI and SMIv2 specifications to SMIng classes and we extended the model with specific SMIng objects that were later mapped to SNMP and/or COPS-PR respectively. For the platform part, we describe the design and the development of this SMIng environment and show how it was applied to the FLAME execution environment, used here as the managed environment and not as the management platform.

The remainder of the paper is organised as follows. Section 2 presents the needs of management and provisioning in FLAME. From theses needs, we present an SMIng specification in section 3. Section 4 describes the architecture of the proxy between SMIng objects and management/provisioning information on the network. Section 5 details how we built a proxy agent from SMIng specifications and how it was integrated with the Java JMX environment. The use of the proxy for FLAME is shown in the section 6. Related work and conclusions are given in sections 7 and 8.

2. FLAME MANAGEMENT AND PROVISIONING

As already described in the introduction, FLAME is an active network dedicated to host IPv6 monitoring services. The architecture of a FLAME node is illustrated in Figure 1. It is close to several other active nodes like the ASP EE [2] on which it was initially built.

Basically, a node is divided into three parts: the standard routing engine of the node on which the environment resides, the execution environment (EE) which offers the basic services to active applications (dynamic code loading, access to node resources, naming, ...). Active applications (AA) are executed on execution environments and use in addition to standard services, dedicated APIs provided by the FLAME environment (e.g., a packet capture API, a multicast routing table manipulation API, ...). These APIs are also globally named in FLAME and can be dynamically deployed through a FLAME specific management operation within a node. Each element within a FLAME node is under the control of a standard management entity through which configuration and control is performed. Each node today hosts one SNMP agent as well as a PEP for enforcing configuration choices. We use separate technologies for management and provisioning, with SNMP and COPS-PR, because some of their specific properties match well with the FLAME environment. Monitoring information is made of several counters modelling active node activity. The role concept of COPS-PR is a way we use to distinguish some FLAME node (border gateway, community of nodes, ...). Underlying TCP connections reduce the needed complexity for the critical operation of policy rules dowloading.

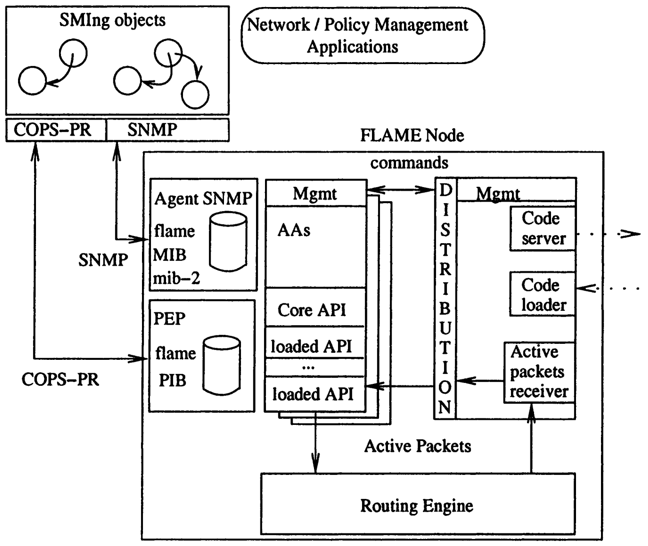


Figure 1. Architecture of a FLAME node

The need of a common information model appears with the fact that several faults can be deduced from network management data and can be reduced by policy-based configuration. For example, some policy rules are listed below:

- authorization policies, e.g., the active application `MRMMonitor` can be launched on the node, if its instantiation parameters are valid (e.g., identity of the code server for the active code, identity of the principal who wants to launch the application);
- obligation policies, e.g., an active application needs to end correctly on 99% of its executions, an active application cannot use more than 20% bandwidth;
- configuration policies, e.g., APIs that are loaded in the FLAME node, ...

The FLAME SNMP MIB includes MIB-II objects and several FLAME specific ones like: the number of incoming/outgoing octets per active application, the number of active messages processed by an active application, the owner of an active application, the number of shutdowns of an active application, ...

Many of theses informations could be related as when an active application shows a number of crashes, policy rules should be changed to stop any further launch of the suspicious application. With many such relations, management applications are less complex with an unified interface than with differents protocol APIs. SMIng appears to be one way to seamless integrate SNMP network management and COPS-PR/SPPI policy rules management. To unify the management information model dedicated to FLAME, we have built an SMIng specification for the FLAME specific objects put in the SNMP agent as well as SMIng classes for provisioning policies (upper left part of Figure 1).

3. SMING

SMIng (Structure of Management Information next generation) is a proposal that was submitted to the sming IETF working group [6]. This proposal is not the sole candidate for standardisation (SMI-DS *Data Structure* is an example of another possible approach) but has the advantage of being object-oriented like, and not being bound to any underlying approach while offering compatibility with both SNMP-SMI and SPPI.

While SMIng is described in the related drafts, we shortly present its structure through an example (Figure 2). It contains the definition of management information which model statistical measures of active application executions and together with policy rules used to decide if an active application could be launched.

```
(1)  typedef Counter32 {
(2)    type Unsigned32;
(3)  };
(4)  typedef Counter64 {
(5)    type Unsigned64;
(6)  };
(7)  typedef AAName {
(8)    type OctetString(255);
(9)  };
(10) typdef EEName {
(11)   type OctetString(64);
(12) };
(13) typedef AAVersion {
(14)   type OctetString(8);
(15) };
(16) class AARunning {
(17)   attribute Counter32 run;
(18)   attribute Counter32 crash;
(19)   attribute Counter32 end;
(20) };
(21) class AANetwork {
(22)   attribute Counter64 inOctets;
(23)   attribute Counter64 outOctets;
(24) };
(25) class AAStats {
(26)   attribute AAName name;
(27)   attribute AARunning run;
(28)   attribute AANetwork net;
(29) };
(30) class AAThresold {
(31)   attribute Counter32 crash;
(32)   attribute Counter32 bandwith;
(33) };
(34) class AAInstance{
(35)   attribute AAName name;
(36)   attribute EEName env;
(37)   attribute AAVersion ver;
(38)   attribute AAThresold thr;
(39) };
(40) snmp{
(41)   table ActiveApplStats {
(42)     oid flame-mib.2.3;
(43)     index(1);
(44)     implements AAStats {
(45)       object 2 name;
(46)       object 3 run.run;
(47)       object 4 run.crash;
(48)       object 5 run.end;
(49)       object 6 net.inOctets;
(50)       object 7 net.outOctets;
(51)     };
(52)   };
(53) };
(54) copspr{
(55)   prc AAInstancePolicyRule {
(56)     oid flame-pib.1.4;
(57)     pibindex (1);
(58)     implements AAInstance {
(59)       object 2 name;
(60)       object 3 env;
(61)       object 4 ver;
(62)       object 5 thr.crash;
(63)       object 6 thr.bandwith;
(64)     };
(65)   };
(66) };
```

Figure 2. SMIng management information specification

3.1 Type and class definitions

SMIng is designed for the definition of data interfaces. Thus, there is no procedural or functional statement support but only data oriented specifications. SMIng provides a set of basic data types like `OctetString`, `Unsigned32`, etc. From these types, new ones can be defined through the use of the `typedef` statement. Examples of such `typedef` statements are shown in lines 1 to 15 on Figure 2 (some `Counter` and `Name`).

SMIng provides a `class` statement to define object classes that are composed of `attribute` and `event` statements and can use simple inheritence (not shown in this

paper). Examples of class definitions are shown in lines 16-39 (AAStats, AAInstance and contained classes AARunning, AANetwork and AAThresold).
Class AAStats models statistical counters for executions status and network use of all active applications designed by its name in the AAName attribute. Class AAInstance models a policy rule for an authorized AA to be instanciated with its obligations (i.e numbers of crashes and network bandwith use).

Class attributes can be either of base types (defined through the typedef statement as shown in lines 17-19, 22, 23, 26, 31, 32, 35-37) or of another class (lines 27, 28 and 38).

At this level, one can specify when a new active application, say *aa* is ready to be downloaded by the FLAME node, an instance of the AAInstance class is created in all nodes that will use *aa* to enable launches. The name *aa* and other property values are given respectively to the defintion of the class (line 35). The first launch in a node will be followed by the creation of a AAStats with the name *aa* and counters start to be updated. Further launches of *aa* will equally update the AAStats instance.

A fault state appears when an active application shows a number of shutdowns in the crash attribute of AAStats greater than the same attribute in the AAInstance. This case could be detected by a polling procedure (notification reception through an SMIng event is possible but not used here) and the following action should be the deletion of the AAInstance object. Therefore, already launched applications could (maybe correctly) finish but no other launch of this application on this node can be done. On the other hand one can choose to increase the crash number threshold.

3.2 Mapping specification

The core of SMIng is protocol independent and can be used for both network and policy management as well as other domains. As these approaches strongly rely on dedicated information models, albeit sometimes very similar, and specific protocols, SMIng offers a facility to express the specification of a mapping from classes and their attributes to protocol specific information. As an example, we show both a SNMP (lines 40 to 53) and a COPS-PR (lines 54 to 66) mapping specifications. Each mapping specifies which SMI table or SPPI provisioning class of the existing MIB and PIB for FLAME implements some SMIng object class. The oid part (lines 42 and 55) gives the global object identifier (other SMIng constructs allow a full definition of the object identifiers for flame-mib and flame-pib). Following is the index column(s) specification of the table or PRC and the implemented SMIng class (lines 44 and 58). The object statement maps a column identifier to an SMIng class attribute. If an attribute is itself a class, a dotted notation is used to go until a simple value attribute is found (lines 46-50 and 62, 63).

4. A SMING-BASED PROXY AGENT ARCHITECTURE

Remember that our goal is to build a set of tools that enable SMIng specifications to be used in the core of management platforms as a unique data model. We therefore need to instantiate SMIng objects in order to use them as programmatic object instances.
We expect management applications to become less complex by hiding specific pro-

tocol dependent operations and as a consequence, to be more homogeneous when processing both policy and network management information.

The architecture that was designed can be seen as an SMIng distributed middleware hosting SMIng object instances. Groups of objects are hosted by several integration agents which are SMIng proxies. This is illustrated in Figure 3.

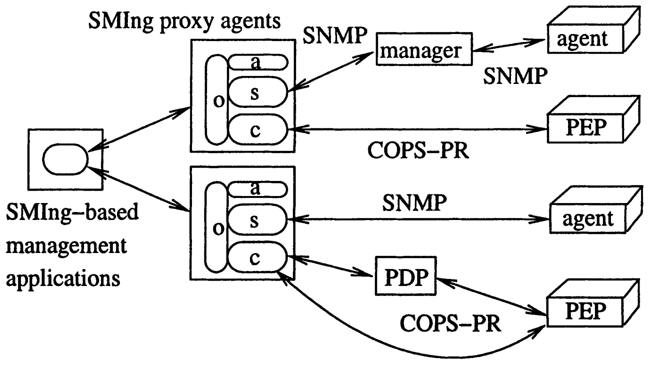


Figure 3. General architecture

An SMIng proxy agent contains an object view (interface **o**) that can be used by object-oriented management applications to access, modify, create or delete managed object instances. These managed objects can then be mapped to underlying resources through two main interfaces:

- interface s can be seen as an SNMP manager that has direct access to local or remote SNMP agents. Through this interface, mapping between SMIng and SNMP SMI is automated according to the rules defined in snmp statement implementation specifications;
- interface c can be seen as a policy decision point (PDP) that pushes policy information to policy enforcement points (PEP) using the COPS-PR protocol. Here we stick to the mappings defined in copspr statement implementation specifications.

A third interface (named **a**) represents access to internal SMIng objects.

5. IMPLEMENTATION

5.1 Generation of object instances

To implement the proxy architecture for SMIng objects, we chose the Java Management eXtension (JMX) framework[1]. Figure 4 contains an illustration of an SMIng proxy agent that includes managed objects taken from the example given in Figure 2.

Only Java objects corresponding to `typedef` values are protocol dependent (`AANName` or `Counter32`). Other SMIng objects are created after the instanciation of contained

[1]http://java.sun.com/products/JavaManagement/

objects (`AAStats` is created after `AARunning`) and linked to each other from the implementation specification. In doing so, the process of object instanciation never leads to "null pointer" troubles.

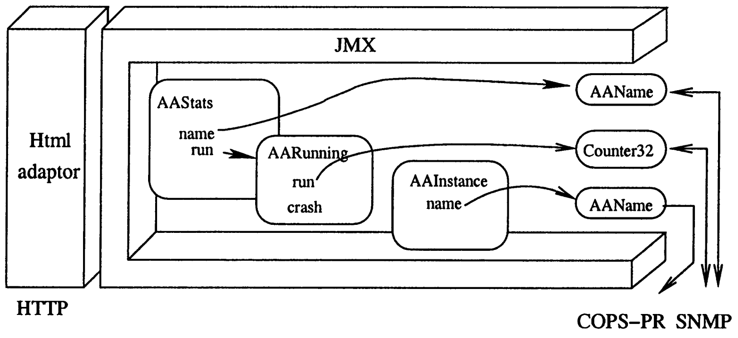
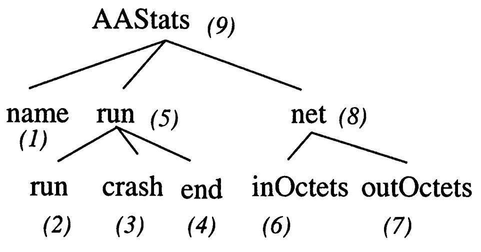


Figure 4. SMIng object instances in a JMX container

Figure 5 shows how the instance creation process is automatically generated from the implementation specification. On the left part of the figure there is a schematic representation of the implementation. It is always a tree structure that is given by our SMIng parser and followed by our proxy agent generator. The root is the name of the class. Intermediate nodes are always attributes that reference a class, while leaf objects are simple types. The right part shows a subset of the generated Java code for this tree. The code of the lines 1 to 6 is for the creation of SMIng `typedef` objects. Parameters of these objects are generated following the object-identifier naming of the implementation (here the `ActiveApplStats` as in Figure 2 line 41) and the remote SNMP agent from which the MIB is translated. Once these leaf objects are created, the generated code creates SMIng object instances (lines 7-9) and sets their attributes through its constructor method invocation The same code is generated for the `AAInstance` class with its attributes.



```
(1) name = new AAName(...);
(2) runrun = new Counter32(...);
(3) runcrash = new Counter32(...);
(4) runend = new Counter32(...);
(5) netinOctets =
                   new Counter32(...);
(6) netoutOctets =
                   new Counter32(...);
(7) run = new AARunning(runrun,
                   runcrash, runend);
(8) net =
        new AANetwork(netinOctets,
                      netoutOctets);
(9) aastat = new AAStats(name,
                         run, net);
```

Figure 5. Implementation scheme generation

This code is generated for each SMIng class implemented directly by an SNMP table. When the processed group is a table, there will be exactly as many object instances created as the number of lines in the table for each SNMP managed object. In order to do this, we use the `get-next` SNMP request, starting with the table SNMP object identifier and walk through the table in order to get the values we need to create the leaf objects first, and end with containing objects. When SMIng classes are implemented by a scalar group there should be only one object created by the SNMP agent proxy to SMIng and we use one `get` request for all needed values.

When the underlying mapping is COPS-PR, the approach is similar. Mapping to SPPI can even be seen as a subset of the mapping from SMIng to SMIv2 since only tables can be defined using SPPI (no scalar) and these tables all hold an unique and known index. A major difference is that SPPI values in PIB should not be read from the PEP, as it is the case for SMI values from agents MIBs. Instead the creation of such one SMIng instance should be mapped to a provisioning COPS-PR request for the corresponding rule in the PEP (or in a PDP that will forward rules as in Figure 3).

5.2 Naming scheme

Within a JMX container, object instances are named according to a standard naming scheme, very close to the OSI Distinguished Name pattern. The general form of an instance name is:

```
Domain::attr1 = value,attr2 = value ...
```

where `Domain`, each `attr`*i* (attribute) and `value` are character strings. Within our SMIng mapping entity, we use this naming convention to uniquely identify SMIng instances as follows:

- the domain identifier is the SMIng module name where the class is defined;
- attributes and values are defined as:
  - l = *SNMP agent or COPS-PR PEP DNS name or IP address*
  - c = *SMIng class name*
  - p = *position of the class in the containing class*
  - i = *instance number*

Every instance has a name which contains these attributes. Since JMX does not put any constraint on the order of attribute occurrences in a name (the lexicographic ordering is defined as the normal form), they can appear in any order. Figure 6 illustrates the naming with our FLAME example. The position of an instance in a containing instance is built from the containing SMIng class definition and from the instance number of the latter object according to its class identifier. For example the `AAStats` class (Figure 2 lines 25-29) has three attributes and their relative positions are 1, 2 and 3. We chose to give the value "1"to each SMIng object that is not contained in any other SMIng object. So objects referenced by `name`, `run` and `net` attributes have respectively "1.1", "1.2" and "1.3" for their p naming attribute value. Contained SMIng objects have the same prefix than the containing SMIng one. We chose to keep an SMI object identifier like notation to reduce the length of object name. If the class is not contained by any other class, its name is the class name followed by same value for the position and the instance number. We keep this redundant information to always have the same number of naming attributes. The instance number attribute is given by the proxy agent for each object creation. If an SMIng object is contained by more than one other object, this former should have as many names. In any case, one name provides a way to access to exactly one object instance.

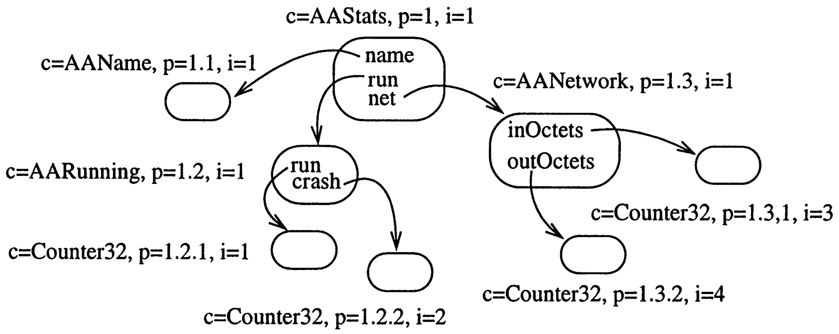


Figure 6. Naming of instances

5.3 Notification support

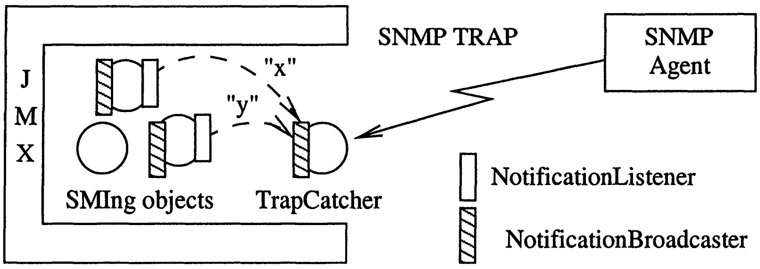
SMIng allows event definitions in object classes. Such a feature of object instances is only implemented by SNMP notification in the SNMP framework (COPS-PR does not support notifications). Figure 7 illustrates the use of events whithin SMIng objects. `System` and `If` are object classes that represent the system and the interface group of MIB-II. Events are defined in classes (lines 3 and 8), after attributes. The `snmp` statement (lines 10-21) contains the implementation of these events. Standard notifications (lines 11, 12 and 15, 16) are related to object events by a `signals` statement (lines 13 and 17) with a dotted notation (*ClassName*. *eventName*). If the trap carries SNMP object values that are also SMIng attribute values, a mapping can be specified (line 18). In this case, a `linkdown` notification received will update the `adminState` SMIng object attribute (line 6). Figure 8 shows how SNMP traps are catched by a dedicated ob-

```
(1)  class System {
(2)    ...
(3)      event warmStart;
(4)  };
(5)  class If {
(6)      attribute AdminState adminState;
(7)      ...
(8)      event linkDown;
(9)  }
(10) snmp{
(11)   notification warmStart {
(12)     oid snmpTraps.2;
(13)     signals System.warmStart{};
(14)   };
(15)   notification linkDown {
(16)     oid snmpTraps.3;
(17)     signals If.linkDown {
(18)       object If.adminState;
(19)     };
(20)   };
(21) };
```

Figure 7. SMIng events definition and implementation

ject playing the role of an SNMP trap catcher. Its role is also to map incoming SNMP notifications to java object that are forwarded to registered SMIng object instances. In the context of JMX, the trap catcher and SMIng objects (that are all JMX MBean objects) must implements specific interfaces. The `NotificationListener` interface allows SMIng object to receive notification signal by registering itself with the standard notification `oid` to a `NodificationBroadcaster` interface implemented by the `TrapCatcher`.

Generated SMIng events follow a similar operation, i.e., they are distributed using the JMX notification model.



x : 1.3.6.1.6.3.1.1.5.3 (linkDown NOTIFICATION-TYPE)
y : 1.3.6.1.6.3.1.1.5.2 (warmStart NOTIFICATION-TYPE)

Figure 8. SMIng events support

Some standard notifications are related with SNMP table entries because they contain an index value that specifies which line of the table is the root of the generated SNMP trap.

When an SMIng class containing event definition is implemented by an SNMP table and when there is a standard notification that carries an index value then each SMIng object (one object per line) should be able to capture the corresponding event. This index value is used by the `TrapCatcher` to identify the SMIng object.

In addition, some notifications carry values that should be updated in the object instance, or be interpreted as the need to refresh some object attribute. We exploit this in conjunction whith a regular polling service in charge of maintaining the consistency between SNMP MIBs and the proxy.

5.4 Code generation tools

For generating the Java classes, we use an SMIng extension of the MODERES framework. MODERES is basically a set of Open Source Java tools maintained by our research group dedicated to the parsing and processing of multi-approaches management information models (GDMO/ASN.1 SMIv1, SMIv2, CIM-MOF, ...). The toolkit is available on the group's web page[2].

Figure 9 shows the use of these compiler tools for the SMIng proxy generation. First a syntax and semantics check is performed on incoming SMIng specifications (both core definition and protocol mappings) by the parser. The specifications are then stored in a repository in the form of a decorated syntax tree. This repository is the source for the SMIng agent toolkit (`SMIngAtk`) that generates Java classes and interface mappings to SMIng classes. Note that SMIng `typedef` as well as events, are also mapped to Java classes and interfaces (dashed arrows). Additionally some Java classes are generated in order to realize the necessary operation to get/set values in a specific protocol (SNMP, COPS-PR).

[2]http://www.madynes.org

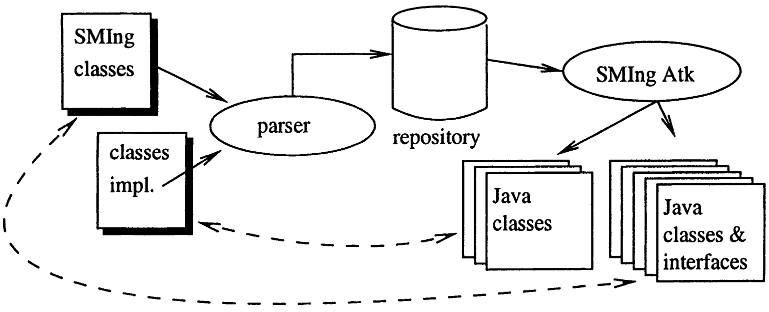


Figure 9. SMIng tools

6. APPLICATION TO FLAME

Having all components defined as SMIng objects was of great use especially for those applications which access objects mapped to the two worlds. For example, one policy application automatically updates the policy repository of all nodes, if it finds out that an application uses too many resources on a node (e.g., number of crashes). The updated policy forbids instantiation of the application with the parameters that cause the trouble in one node. Building such an application with our framework is straight-forward since one only needs to know the SMIng class that corresponds to this policy family and set up a monitoring service for the attribute that represents the number of crashes of an active application. To push a configuration policy that forbids the execution and further instantiation of an application, only the policy object `AAInstance` defined in Figure 2 lines 34-39 need to be instantiated. Once instantiated, the decision is mapped onto a COPS-PR service invocation and pushed towards the concerned PEP. A java.rmi adaptor to the SMIng proxy agent was designed to allow processing in our network management java application.

7. RELATED WORK

The object oriented network modelling is well described in [1]. The concept of Meta Managed Objects [14] contains the same base elements as those proposed by SMIng with a separate definition of data and their different representations. Other mappings exists from objects to TMN or SNMP management information [16, 4] or the opposite way e.g., from WBEM to OSI based management [5].

The *libsmi* project of the Technical University of Braunschweig[3] provides a library to access SMI information. A component of this library is an SMIng parser that allows a syntax and semantical analysis of modules. An HTML version can be tested on line at the Simple Web site[4]. An API provides access to MIB and PIB modules information to ease the development of management applications. Our project has a structure similar to the *libsmi* parser. The main differences are the programming language envi-

[3]http://www.ibr.cs.tu-bs.de/projects/libsmi
[4]http://www.simpleweb.org

ronment (C for *libsmi* and Java in our case) and the application domains (information model core tools for *libsmi* and agent toolkit in our case).

As mentioned in different parts of the paper, the FLAME environment configuration is done through policy-based management. The use of policy-based approaches for the management of active networks is investigated in several other places and is not explained in this paper. The reader will find studies on this topic in [15] and [8] for policies dedicated to resource management in active networks.

8. CONCLUSION AND FUTURE WORK

In this paper, we described a software architecture based on the SMIng approach and its application to the management of an active network infrastructure called FLAME (with is itself an environment dedicated to the management of IP networks).

Several lessons can be learned from this experience. First, the situation where both policy-based approaches and the standard SNMP framework need to be combined exists and the number of occurrences will probably grow in the next few years (e.g., the COPS model is proposed for provisioning and outsourcing in several 3G architectures). The second lesson is that SMIng appears to be a reasonable evolution in the standard framework, in the sense that it provides enough support for our needs namely object-orientation and automated integration of monitoring and provisioning. As it was demonstrated in this paper, the approach can be implemented to build an operational management framework. The third lesson, which is obvious, is that the design and development of applications that combine policy manipulation and MIB object access is much more convenient within a common framework. This has been demonstrated while we developed the applications for the active environment.

The first evolution of the presented work is to let the designed management framework follow the evolution of the outcome from future evolutions of IETF groups working on this topic. Work is still in progress on this subject and extended proposals will emerge. In parallel to this work, we are looking at how the mapping principles defined in SMIng can be reused in other approaches like WBEM since the same need for dynamic mapping will appear for these approaches as well. Finally, we will continue the refinement of policy definitions for managing the FLAME environment. The final goal is to end-up with a complete automated monitoring environment driven by pushing configuration policies into the active nodes.

References

[1] S. Bapat. *Object-Oriented Networks : Models for architecture, operations and management.* Prentice Hall, 1994.

[2] B. Braden, A. Cerpa, T. Faber, B. Lindell, G. Phillips, and J. Kann. ASP EE: An Active Execution Environment for Network Control Protocols. Technical report, USC/ISI, December 1999.

[3] K. Chan, J. Seligon, D. Durham, S. Gai, K. McCloghrie, S. Herzog, F. Reichmeyer, R. Yavatkar, and A. Smith. COPS Usage for Policy Provisionning (COPS-PR), RFC3084, March 2001.

[4] T.R. Chatt. TMN/C++: An object-oriented API for GDMO, CMIS, and ASN.1. In *[7]*, pages 177–191, 1997.

[5] O. Festor, P. Festor, L. Andrey, and N. Ben Youssef. Integration of WBEM-based Management Agents in the OSI Framework. 1999. in Integrated Network Management, VI, Sloman, M. and Mazumdar, S. and Lupu, E. editors,IEEE Press, Proceedings of the IFIP/IEEE 6th International Symposium on Integrated Management, Boston, MA, 24-29 Mai, 1999.

[6] F. Strauss J. Schoenwaelder. Next generation structure of management information for the internet. In R. Stadler & B. Stiller, editor, *Active Technologies for Network and Service Management, DSOM'99*

Zurich, Switzerland, pages 93 – 106. Lecture Note in Computer Science, IFIP/IEEE, Springer, October 1999.

[7] A. Lazar, R. Saracco, and R. Stadler, editors. *Integrated Management V*. IFIP, Chapman & Hall, May 1997.

[8] I. Liabotis, O. Prnjat, and L. Sacks. Policy-Based Resource Management for Application Level Active NEtworks. August 2001. Proc. Second IEEE Latin America Network Operations and Management Symposium, LANOMS'2001, Belo Horizonte, Brazil.

[9] K. McCloghrie, M. Fine, J. Seligson, K. Chan, S. Hahn, R. Sahita, A. Smith, and F. Reichmeyer. Structure of Policy Provisionning Information (SPPI), RFC3159, August 2001.

[10] K. McCloghrie, D. Perkins, and J. Schoenwaelder. Conformance Statements for SMIv2., April 1999. IETF, STD58, RFC 2580.

[11] K. McCloghrie, D. Perkins, and J. Schoenwaelder. Structure of Management Information Version 2 (SMIv2), April 1999. IETF, STD58, RFC 2578.

[12] K. McCloghrie, D. Perkins, and J. Schoenwaelder. Textual Conventions for SMIv2., April 1999. IETF, STD58, RFC 2579.

[13] H. Sallay, O. Festor, and R. State. A distributed Management Platform for Integrated Multicast Monitoring. pages 483–496, 2002. Proc. IEEE/IFIP Network Operations and Management Symposium NOMS'2002, R. Stadler and M. Ulema, editors, IEEE ISBN 0-7803-7382-0, Florence, Italy, April 2002.

[14] J. Seitz. Meta managed objects. In *[7]*, pages 650 – 660.

[15] M. Sloman and E. Lupu. Policy Specification for Programmable Networks. In S. Covaci, editor, *Active Networks: Proc. First International Working Conference, IWAN'99*, pages 73–84, Berlin, Germany, June 1999. Springer Verlag, LNCS 1653.

[16] N. Soukouti and U. Hollberg. Joint Inter Domain Management: CORBA, CMIP and SNMP. In *[7]*, pages 153–164, 1997.

TOWARDS XML ORIENTED INTERNET MANAGEMENT

Frank Strauß
Technical University Braunschweig, Germany
strauss@ibr.cs.tu-bs.de

Torsten Klie
Technical University Braunschweig, Germany
tklie@ibr.cs.tu-bs.de

Abstract: Internet Management is based on IETF specifications that have been developed and used during the past 14 years: There are multiple versions and options of the management protocol (SNMP), two versions of the language for specifying the structure of management information (SMI), and more than 160 Standard MIB modules. This altogether represents the most widely deployed management technology these days.

However, the SNMP centered management framework has some drawbacks especially related to efficient configuration management and efficient application development processes. Today to many people the whole family of XML technologies seems to promise a fancy way out of this trouble.

This paper presents an approach to automatically convert SMI MIB definitions and according SNMP agent data into XML Schema definitions and appropriate valid XML documents. Instead of a plain mapping of MIB trees to nested XML elements, we tried to adapt the XML philosophy of a rather flat element containment hierarchy and appropriate XML Schema type definitions. We also present an approach towards an SNMP/XML gateway and our thoughts on management applications based on XML technologies.

Keywords: Internet Management, Configuration Management, Information Model, Data Model, SNMP, SMI, MIB, XML, XML Schema, XSLT, SNMP/XML Gateway.

1. Introduction

Since 1988, when the first specifications of SNMP and the SMI and the first definitions of managed objects have been published by the IETF [1, 2, 3], the SNMP framework evolved dramatically in a way that problems have been identified and fixed, many details in specifications have been clarified and — probably the hardest part of the work — more than 160 MIB modules have been designed, refined and standardized by various IETF working groups. See the latest issue of the *Simple Times* [4] for an overview of all related IETF Standards documents.

The result of this solid standardization work is a fundamental technology upon which many hardware vendors, software manufacturers, network operators, and administrators have built software systems to manage a wide range of computer networks. These implementations also represent huge investments.

However, the SNMP framework has some drawbacks that cannot be solved without massive modifications or completely new technologies [5, 6]. Some people expect that at least two of the major problems can be solved by applying XML based technologies to network management tasks: (a) efficient and atomic transfer of configuration data is problematic with SNMP, and (b) usual development processes of SNMP agents and other applications are quite slow and expensive, since the abstraction layer of MIB definitions and the SNMP protocol is much lower than the typical tasks that have to be fulfilled, so that well experienced staff is essential.

Section 2 explains how some of the core XML technologies are related and how they could be applied to typical network management tasks. In Section 3 we present some related work on XML based techniques for network management. In Section 4 we present and discuss our work on a MIB compiler that generates XML Schema definitions out of SMIv2 MIBs. Subsequently, in Section 5 we present some thoughts on potential XML based network management tasks. We also describe our approach to retrieve real-world SNMP data as XML documents through a simple SNMP-to-XML gateway that complies with our automatically generated XML Schema definitions. The last Section concludes with some experience statements and a rough outlook on future directions.

2. XML Technologies in Network Management

Clearly defined and standardized data structures, encoding schemes and access interfaces are a key issue for any kind of open communication systems. During the past in several areas different base technologies have been applied in a quite successful manner, e.g. management data definitions in the worlds of SNMP and CMIP as well as many protocol definitions are based on ASN.1, business data is exchanged by EDI-compliant messages, databases are often accessible through well defined ODBC or JDBC APIs, remote procedure calls and remote object access can be defined and realized through RPC and CORBA technologies, etc.

While these and many other technologies evolved over time and work quite well today, they have been developed independently. They do not build upon each other and they hardly use basic concepts in a common fashion, although many of them have to solve similar questions like byte ordering, data framing, data encryption, etc.

In contrast, XML [7] is a core building block upon which other related technologies are being developed. This "toolkit concept" allows for more efficient development processes of according applications. For example, existing XML parsers can be used to develop XSLT [8] processors and XML Schema [9, 10, 11] based validators, since XSL and XML Schema themselves are XML compliant. XML Schemas can be defined and used by validating XML parsers to ensure XML data integrity. Specific XML compilers can be built as XSL stylesheets using any already existing XSLT processor.

These advantages of existing XML-based specifications and their already existing implementations are applicable as building blocks to a wide range of network management aspects as well:

- **Management data can be represented as XML documents.** While the SNMP framework focuses on a simplistic management protocol and small items of management data which are appropriate for monitoring, it has severe disadvantages when larger chunks of configuration data has to be retrieved from an agent or stored to an agent: The performance of large chunks of GetNext or Set operations is quite poor [12]. Even more critical is the lack of a transaction model to ensure data integrity across a sequence of protocol operations. When a complete set of management data is represented as an XML document without any

formal length restrictions it could be moved efficiently and handled atomically. Of course, this requires an appropriate transport protocol.
Comparing XML to plain ASCII information that can be retrieved from many devices through a command line interface, parsing tagged XML data is much easier than the error-prone processing of hardly structured ASCII information that is intended for human reading.

- **Widely deployed protocols can be used to ship the data.** E.g. TCP and HTTP are implemented in almost any network device these days. These protocols can easily be used to transfer XML documents. URLs [13, 14] can be used to address the requested data [5].
- **The DOM and SAX APIs can be used to access management data from applications.** Most XML parsers implement these standard APIs to access the contents of XML documents ([15], et al). They can be used by individual management applications [6, 16, 5].
- **Items within management data documents can be addressed through XPath expressions** [17] [6]. This could be useful e.g. when only parts of very large data are being transfered between entities or when management data is processed by XML based applications, e.g. through XSLT:
- **XSLT can be used to process management data.** Although a little cryptic, XSL [8] is a quite powerful stylesheet language. E.g., it can be used to filter XML data, to correlate data from several documents, to generate statistics, or to create concise HTML pages or reports in other text-based formats.
- **The structure of management data can be expressed as XML Schemas** [9, 10, 11]. This would allow, e.g., to ensure the integrity of configuration data documents through the use of a usual XML parser that checks whether the document is well-formed and valid according to the XML Schema definition.
- **High-level management operations can be defined through WSDL and called via SOAP** [6]. E.g., row creation and deletion in SNMP through RowStatus objects can be quite complicated. This task can be achieved by higher-level operations in a more convenient way. However, these operations have to be well defined by WSDL [18] definitions for SOAP [19, 20, 21] functions.

Figure 1 illustrates how the XML technologies and according tools that are discussed above relate to each other. Specifications and tutorials for all of these technologies as well as pointers to detailed literature and implementations are available from the W3C [22].

3. Related Work

This section presents a vendor product, three research projects on Web and XML based management and an open source software project, as well as the starting points of related standardization work within the IETF and an industry consortium.

3.1 JUNOScript from Juniper Networks

Recent releases of routers from Juniper Networks are equipped with a JUNOS operating system that supports the JUNOScript [23] subsystem. JUNOScript allows client

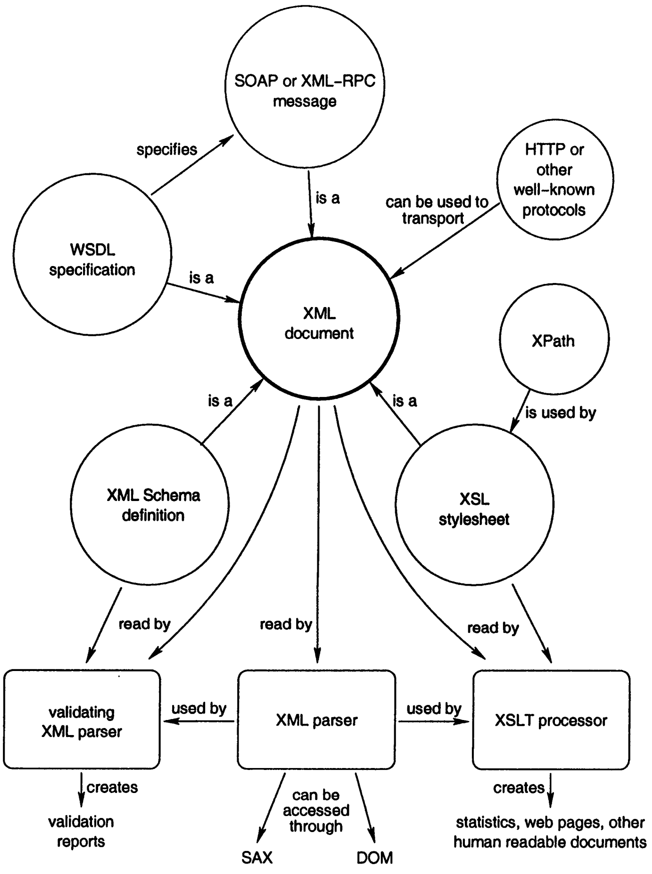


Figure 1. XML technologies and tools.

applications to connect to the Juniper router and exchange messages formed as XML documents. The grammars of these documents representing requests and responses are supplied by Juniper as DTDs and XML Schemas along with documentation that details the semantics of all message elements. A Perl module is supplied to ease the development of client applications communicating with the routers. Further processing of the XML documents can be achieved with third party XML tools that are available for Perl and many other programming languages.

Various protocols can be used to establish sessions between client applications and JUNOScript servers, e.g. TELNET, SSL or SSH. The messages sent by the client constitute RPC requests wrapped in `<rpc>...</rpc>` elements, upon which the server responds with `<rpc-reply>...</rpc-reply>` responses. The contained elements represent the requested actions, e.g. client authentication, configuration queries, configuration modifications, locking for exclusive access, etc. Collections of configuration data can be represented as nested XML elements within the messages as well as in the text format that is also used by the JUNOS CLI.

3.2 Avaya Labs Research on an XML based Management Interface for SNMP Enabled Devices

Concepts for an XML based interface to read and write management information of SNMP agents are being researched in a project at Avaya Labs [24]. To achieve these aims, a mapping from SMI MIB modules to XML Schema definitions has been defined and implemented. Furthermore, an XML-RPC based protocol is being defined and implemented for retrieving and modifying MIB information on SNMP agents. This protocol uses XPath to identify MIB variables within the agent.

The XML Schema definitions are derived from SMI MIB definitions in a straightforward way. For instance, the native XPath expression for addressing the first interface within an XML document derived from the IF-MIB would be `/IF-MIB/interfaces/ifTable/ifEntry[index='1']`. For one SMI MIB module the compiler generates a number of XML Schema files: one main file, one file containing all type definitions, one file per scalar object group, one file per additional MIB tree level and one file per MIB table. All these files are included by the main XML Schema file.

Much of the MIB information that is not necessarily required in XML Schema definitions is dropped, e.g. description clauses and display hints of type definitions. Some SMI types are mapped to more 'tolerant' XML Schema types, e.g. `IpAddress` is mapped to `xsd:string`. Elements representing scalar groups contain a redundant `index` attribute and, more seriously, tables with multiple index attributes are not yet supported. However, these problems could probably be fixed subsequently in this project.

The development and implementation of an appropriate SNMP-to-XML adapter based on the NET-SNMP and XML-RPC libraries has been underway during this writing.

3.3 POSTECH Research on XML-based Internet Management

Hong et al propose an XML-based Management (XBM) architecture [6] that is based on XML/HTTP as its management protocol. To allow the integration of SNMP managed devices they propose three methods to realize an XML/SNMP gateway [16].

The first approach is to implement a DOM interface that translates DOM function calls to SNMP operations. The results of the internally executed SNMP operations are then translated to XML nodes and passed back to the DOM based management application. In this approach the gateway is tightly bound to the management application through the DOM API.

In the second approach, a standalone gateway accepts HTTP requests from management stations that are based on URIs that may contain XPath or XQuery expressions to address specific agent object instances. The gateway parses these requests and translates them to SNMP operations issued to the SNMP agents. The SNMP responses are used to build a "response XML document" which is then sent back to the manager in response to its request. SNMP traps destined to the gateway can be processed in a similar way and forwarded as XML documents to managers through HTTP POST requests.

In the third approach, the gateway implements a SOAP RPC service. On receipt of a SOAP input message from a manager it is translated to SNMP operations. Vice versa, the SNMP responses are used to construct the SOAP output message. While this approach introduces the most protocol and processing overhead, it also gives the best possibilities to extend the gateway with more powerful operations like scheduled polling.

Although this project carefully examines valuable methods to realize an XML/SNMP gateway, the mapping of SNMP management data to XML documents is quite simple and driven by SNMP practice, similar to the Avaya approach (Section 3.2).

3.4 WIMA

The Web-based Integrated Management Architecture (WIMA) proposed by Martin-Flatin [5] integrates multiple management data models, namely at least SMI and CIM, without introducing another data model. This is achieved by defined mappings of specific data models at two levels: In case of SMI, on the model-level an SMI MIB module is mapped to a DTD, i.e., an XML document that complies to such a DTD represents management instance data, e.g., retrieved from an SNMP agent. On the metamodel-level the items within an SMI module are mapped to element and attribute values within a document that complies to a generic WIMA-specified DTD.

WIMA's model-level SMI mapping is comparable to the mapping of SMI to XML Schema definitions presented in this paper: The resulting documents describe a formal grammar of management instance data. On the other hand, WIMA's metamodel-level mapping is also comparable to the XML Schema mapping presented in Section 4 in a way that its vocabulary is generic (WIMA-proprietary vs. XML Schema) while its content represents MIB data models.

As with the approaches presented in Section 3.2 and 3.3, the mapping from SNMP and SMI to XML and XML Schema is quite simple and bound to SNMP requirements and not driven by native XML concepts.

3.5 SCLI

`Scli` [25] is a tool that implements a command line interface to interact with SNMP agents in an interactive or batch-mode fashion. Despite many other SNMP tools, `scli` is not a generic tool to process arbitrary MIB data. Instead, it is designed in a way so that it is aware of the structure and semantics of a number of MIB modules. This way, it is capable to present and accept information to and from the user in a much more human friendly form that often significantly differs from the underlying MIB structures. E.g., when a user requests information on a specific interface, it can be specified by its human-friendly name instead of the agent's notion of an `ifIndex` variable. The data that is displayed is formatted in a human-readable tabular form where numbers are printed with appropriate units and the items are gathered from different tables of different MIBs like the `IF-MIB`, the `IP-MIB`, and the `ENTITY-MIB` [26, 27, 28]. It requires a well experienced person to develop new `scli` modes, but the resulting functionality facilitates the work of less experienced users significantly.

Besides the plain text form, `scli` is also capable to dump its information in an XML form. This way, it is possible, for example, to post-process the data by XSL stylesheets. The XSL programmer can benefit from `scli`'s gathering of all related information and its pre-processing. Currently, XML output is only supported for a limited number of `scli` modes. XML Schema definitions for those XML dumps are not yet available.

3.6 The IETF XMLCONF BOF

During the 54th IETF meeting in Yokohama in July 2002, a BOF session concerned with XML configuration management (XMLCONF) has been held [29]. The goals were to discuss today's operator requirements for configuration management, to identify the disadvantages of today's IETF technologies to meet these requirements and to evaluate some XML related technologies with this respect, namely, SOAP/WSDL, SyncML, WBEM, and JUNOScript. There are some Internet-Drafts that represent a starting point to narrow down these requirements and evaluations. It has not yet been decided whether a new working group shall be formed to continue this work, but it looks very reasonable to work on the standardization of XML based network management within the IETF so that the gap between the wide range of existing SNMP environments and new XML based solutions can be bridged.

3.7 The OASIS Management Protocol Technical Committee

In July 2002, OASIS, a consortium that focuses on industry standards specifications based on XML, founded a technical committee [30] that intends to provide a web-based mechanism to monitor and control managed elements "based on industry accepted management models, methods, and operations, including, OMI, XML, SOAP, DMTF CIM, and DMTF CIM Operations".

4. Converting SMI MIBs to XML Schema Definitions

While the previous two Sections presented general concepts and actual efforts to use XML technologies for network management tasks, in this Section we will study how the large existing SNMP infrastructure can benefit from XML based management data processing. This obviously requires a representation of SNMP data as XML documents.

While the structure of SNMP data is formally described in SMI MIB modules, the structure of XML documents can be defined as a Document Type Definition (DTD) or an XML Schema definition. Whereas DTDs follow a rather simple grammar notation, XML Schema is more flexible to express characteristics of XML documents, e.g., there is a type system that allows to define derived types, and a powerful mechanism to express the format of string values as regular expressions. Furthermore, XML Schema definitions themselves are well-formed XML documents so that they can also be processed by XML parsers.

Consequently, a mapping from SMI MIB modules to XML Schema definitions is useful. While the approaches presented in Sections 3.2 – 3.4 use a straightforward way, we attempt to represent the data in a way that narrows the usual XML characteristics as closely as possible to make the XML instance documents as convenient for reading and processing as possible. For instance, we explicitly do not intend to stick with deep OID equivalent nesting hierarchies. On the other hand, the generated XML Schema definitions contain as much of the underlying SMI MIB module information as possible. The following list describes the most relevant characteristics of the XML documents and XML Schema definitions:

- The range of data that can be represented in an XML document shall be as flexible as possible. Hence, the root element is not bound to a specific MIB or agent or point in time.
- The root element may contain an arbitrary number of `<context>` elements at the second level that represent agent contexts which are identified by a tuple of an SNMP agent, a community string (in case of SNMPv1), and a time stamp to specify the point in time when a context has been created. This allows, for instance, to store data from multiple agents or time series of polled data in a single document.
- The third-level elements can either represent containers of scalar elements that appear at most once, or instances of objects that are derived from table entries and thus can appear multiple times. Note that the list of these elements is not limited to a single MIB module. While scalar container elements don't have any attributes, the table entry elements include one ore more index attributes to uniquely identify the instances. These attributes are derived from the MIB entry's `INDEX` clauses.
- Note that MIB modules are not represented by elements. Instead they are identified by namespaces in which the elements are defined. Since unique naming in SMI is based on modulename-descriptor pairs, we compile each MIB module into a separate XML Schema definition where each schema defines an according namespace.
- The fourth-level elements represent scalar objects or columnar objects. There is no deeper level of element containment (except for nested tables described below), since there is no need for a hierarchy such as with OIDs. This way the XML document hierarchy remains simple but powerful. Unique naming is purely based on namespaces, grouping names with indexing attributes, and leaf element names without any ambiguity.
- Augmentation tables and tables that share a common prefix list of index objects with another table are somewhat confusing SMI constructs to represent nested data structures. When these structures appear in a single MIB they are mapped to a native representation in XML: The columnar objects of augmentation tables are simply added as child elements of the element that represents the parent table. Similarly, "tables in tables" are represented by nesting the according elements.
- The text of leaf elements (and index attributes) is represented in a human readable fashion where possible: Integers are written as decimal numbers. Named number types, such as enumeration types and bit sets, are written as the according names. Strings are written in a human readable ASCII form, if the underlying MIB type has a display hint that 'suggests' an according representation of all octets of the value.
- Type and `TEXTUAL-CONVENTION` definitions in MIB modules are mapped to XML Schema types derived from base types with appropriate `<xsd:restriction>` clauses containing value restrictions for numbers (`<xsd:minInclusive>`, `<xsd:maxInclusive>`) or strings (`<xsd:minLength>`, `<xsd:maxLength>`). However, some limitations

such as length alternatives (i.e., not length ranges) cannot be expressed completely in XML Schema.

- Even complex display hints can automatically be translated to `<xsd:pattern>` constructs with regular expressions that formally limit the value set so that, e.g., a number of typos or otherwise illegal values in XML documents can be prevented.
- SMI MIB module information that is not necessarily required in the XML Schema definition is contained in `<xsd:appinfo>` clauses. This way, it remains available for special applications, e.g., XSLT-based MIB compilers or MIB browsers.
- Despite the representation of the status of MIB object instances an XML document can also represent a sequence of notifications. Hence, the XML Schema for a MIB module contains a `<xsd:choice>` construct where the second alternative is intended to form the grammar rules for arbitrary lists of notifications conforming to a MIB's `NOTIFICATION-TYPE` definitions.

Figure 2 shows an example of an XML instance document, while Figure 3 illustrates some XML Schema constructs compiled from the SMIv2 `IF-MIB` module [26]. The compiler implementation is based on `libsmi`[31].

```
<?xml version="1.0"?>
<!-- This module has been generated by mibdump 0.1. Do not edit. -->
<snmp-data xmlns="http://www.ibr.cs.tu-bs.de/projects/libsmi/xsd/IF-MIB"
           xmlns:IF-MIB="http://www.ibr.cs.tu-bs.de/projects/libsmi/xsd/IF-MIB"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://www.ibr.cs.tu-bs.de/projects/libsmi/xsd/IF-MIB
                               http://www.ibr.cs.tu-bs.de/projects/libsmi/xsd/IF-MIB.xsd">
  <context agent="ciscobs.rz.tu-bs.de" community="public" port="161"
           time="2002-12-12T23:42+0100">
    <IF-MIB:interfaces>
      <IF-MIB:ifNumber>7</IF-MIB:ifNumber>
    </IF-MIB:interfaces>
[...]
    <IF-MIB:ifEntry ifIndex="2">
      <IF-MIB:ifDescr>FastEthernet0/0</IF-MIB:ifDescr>
      <IF-MIB:ifType>ethernetCsmacd</IF-MIB:ifType>
      <IF-MIB:ifMtu>1500</IF-MIB:ifMtu>
      <IF-MIB:ifSpeed>100000000</IF-MIB:ifSpeed>
      <IF-MIB:ifPhysAddress enc="hex">00:03:fd:32:e4:00</IF-MIB:ifPhysAddress>
      <IF-MIB:ifAdminStatus>up</IF-MIB:ifAdminStatus>
[...]
      <IF-MIB:ifName>Gi1/0</IF-MIB:ifName>
      <IF-MIB:ifLinkUpDownTrapEnable>enabled</IF-MIB:ifLinkUpDownTrapEnable>
[...]
    </IF-MIB:ifEntry>
[...]
    <IF-MIB:ifStackEntry ifStackHigherLayer="2" ifStackLowerLayer="0">
      <IF-MIB:ifStackStatus>active</IF-MIB:ifStackStatus>
    </IF-MIB:ifStackEntry>
  </context>
</snmp-data>
```

Figure 2. An XML instance document conforming to the IF-MIB XML Schema.

```
<?xml version="1.0"?>
<!-- This module has been generated by smidump 0.4.2-pre. Do not edit. -->
<xsd:schema
```

```
    targetNamespace="http://www.ibr.cs.tu-bs.de/projects/libsmi/xsd/IF-MIB"
    xmlns:xml="http://www.w3.org/XML/1998/namespace"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:smi="http://www.ibr.cs.tu-bs.de/projects/libsmi/xsd/smi"
    xmlns:SNMPv2-SMI="http://www.ibr.cs.tu-bs.de/projects/libsmi/xsd/SNMPv2-SMI"
    xmlns:SNMPv2-TC="http://www.ibr.cs.tu-bs.de/projects/libsmi/xsd/SNMPv2-TC"
    xmlns:SNMPv2-CONF="http://www.ibr.cs.tu-bs.de/projects/libsmi/xsd/SNMPv2-CONF"
    xmlns:SNMPv2-MIB="http://www.ibr.cs.tu-bs.de/projects/libsmi/xsd/SNMPv2-MIB"
    xmlns:IANAifType-MIB="http://www.ibr.cs.tu-bs.de/projects/libsmi/xsd/IANAifType-MIB"
    xml:lang="en"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">
  <xsd:annotation>
    <xsd:documentation>
        The MIB module to describe generic objects for network
        interface sub-layers.  This MIB is an updated version of
        MIB-II's ifTable, and incorporates the extensions defined in
        RFC 1229.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:import namespace="http://www.ibr.cs.tu-bs.de/projects/libsmi/xsd/SNMPv2-SMI"
         schemaLocation="http://www.ibr.cs.tu-bs.de/projects/libsmi/xsd/SNMPv2-SMI.xsd"/>
[...]
  <xsd:element name="snmp-data">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="context" minOccurs="0" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="interfaces" type="interfacesType" minOccurs="0"/>
              <xsd:element name="ifEntry" type="ifEntryType" minOccurs="0" maxOccurs="unbounded"/>
[...]
            </xsd:sequence>
            <xsd:attribute name="agent" type="xsd:NMTOKEN" use="required"/>
            <xsd:attribute name="community" type="xsd:NMTOKEN" use="required"/>
            <xsd:attribute name="port" type="xsd:unsignedInt" use="required"/>
            <xsd:attribute name="time" type="xsd:dateTime" use="required"/>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
[...]
  <xsd:complexType name="ifEntryType">
    <xsd:annotation>
      <xsd:appinfo>
        <maxAccess>not-accessible</maxAccess>
        <status>current</status>
        <oid>1.3.6.1.2.1.2.2.1</oid>
      </xsd:appinfo>
      <xsd:documentation>
        An entry containing management information applicable to a
        particular interface.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element name="ifDescr" minOccurs="0">
        <xsd:annotation>
          <xsd:appinfo>
            <maxAccess>read-only</maxAccess>
            <status>current</status>
            <oid>1.3.6.1.2.1.2.2.1.2</oid>
          </xsd:appinfo>
          <xsd:documentation>
            A textual string containing information about the
            interface.  This string should include the name of the
            manufacturer, the product name and the version of the
            interface hardware/software.
          </xsd:documentation>
        </xsd:annotation>
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
```

```
            <xsd:pattern value=".{0,255}"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>
      <xsd:element name="ifType" type="IANAifType-MIB:IANAifType" minOccurs="0">
        <xsd:annotation>
[...]
  <xsd:simpleType name="InterfaceIndex">
    <xsd:annotation>
      <xsd:documentation>
          A unique value, greater than zero, for each interface or
          interface sub-layer in the managed system.  It is
          recommended that values are assigned contiguously starting
          from 1.  The value for each interface sub-layer must remain
          constant at least from one re-initialization of the entity's
          network management system to the next re-initialization.
      </xsd:documentation>
      <xsd:appinfo>
        <displayHint>d</displayHint>
      </xsd:appinfo>
    </xsd:annotation>
    <xsd:restriction base="smi:Integer32">
      <xsd:minInclusive value="1"/>
      <xsd:maxInclusive value="2147483647"/>
[...]
</xsd:schema>
```

Figure 3. An XML Schema file automatically generated from IF-MIB.

5. Applications

The previous Section presented an approach to represent management data as XML documents and their structure as XML Schema definitions. So, in this Section we will discuss some applications that could make use of these representations. Note that at this point in time these applications, except the one presented in Section 5.4, are just ideas that have not yet been implemented by the authors of this paper.

5.1 Configuration Management

SNMP is not very suitable for configuration management, since moving configuration data between devices and managers is inefficient and cannot be done in an atomic way without additional MIB support. Putting XML technologies on top of it cannot do any better. Furthermore, data dumped from arbitrary MIBs cannot be regarded as a configuration: To restore a configuration, e.g., if a broken device has been replaced by a new one, additional information like ordering of the required SNMP Set operations and the semantics of many MIB objects is required. However, it could be an advantage if such additional information on a specific MIB is provided in a DOM or XSLT based application and the dumped data is available as an XML document. Accessing data through the DOM API or addressing items through XPath expressions would be more flexible and more familiar to many developers than using any proprietary management toolkit.

5.2 Notification Processing

Not only the values of agent objects that can be read or written, i.e., instances of the MIBs' `OBJECT-TYPE` definitions, can be represented as XML elements. Instances of notifications emitted by an agent can be dumped as XML documents as well. This way, they can be filtered, searched, and post-processed by XPath and XSLT means, for example. Even event correlation tasks could be realized as XSLT applications and produce HTML output for a concise presentation to the operator.

5.3 Agent Validation

Today, there are many agent implementations of MIBs that do not conform to the underlying SNMP and SMI MIB module specifications. Typical cases are objects that have another base type than that in the authoritative MIB definition, string objects of illegal length or object values that the agent is not aware, but which are served as unspecified 'zero' values instead of just skipping the object.

To some degree such flawed implementations can be checked automatically based on a comparison of an agent's dump against the MIB specification. If the data is represented as an XML document and the MIB structure is represented as an XML Schema definition, this comparison can be done by any validating XML parser. Of course, this would require a very 'verbose' representation of agent data that keeps a lot of SNMP specific information in the XML document which is not necessarily required for other XML based post-processing.

5.4 An SNMP-to-XML Gateway

An obvious approach to make management data supplied by SNMP agents available as XML documents is the use of an appropriate translator or gateway. We have implemented a prototype of such a translator that conforms to the XML Schema characteristics described in Section 4. The example of an XML document seen in Figure 2 has actually been generated by this translator named `mibdump`. It works as follows: The SNMP agent (address, port, SNMPv1 community string) and the MIB module to be dumped are passed to `mibdump`. Then the SNMP session is initiated, the structure of the MIB is analyzed through `libsmi` means, and then sequences of SNMP GetNext operations are issued to retrieve all subtrees of the MIB from the agent. `Mibdump` collects the retrieved data in internal data structures, first. When the gathering phase has been finished, the contents of these data structures are dumped in the form of appropriately nested XML elements forming a valid document with respect to the XML Schema.

`Mibdump` has been developed as a first prototype of a translator to generate XML instance documents that comply to the XML Schema definitions we proposed in Section 4. Hence, for simplicity reasons it supports no other granularity than the level of MIB modules: neither single objects or tables of a MIB, nor the whole set of all MIBs implemented by an agent can be retrieved through `mibdump` at once.

Work is underway to develop a real gateway that translates HTTP requests for XML documents to SNMP operations and forms the resulting XML document out of the SNMP response messages. This approach is very similar to the second method proposed by POSTECH (Section 3.3, [16]). The gateway accepts HTTP GET requests for URIs that contain XPath expressions to address regions of the schema compliant XML documents with the full complexity of XPath. Generally, location paths of the XPath expressions can be interpreted in the gateway before sending SNMP requests in order limit SNMP operations, while the predicate parts have to be applied when the SNMP data has been received at the gateway to filter out the parts that the XML requestor wishes to receive. DOM is used to represent and access the XML documents at runtime in the gateway. In case of HTTP GET requests the DOM is built by the core translator based on the received SNMP response messages. In case of HTTP POST requests the applied document is parsed to build the DOM so that the translator can access it to issue appropriate SNMP Set operations. Traps can be logged (for later access by managers) and result in short XML documents sent to registered managers that listen as HTTP POST receivers. To enhance performance in case of subsequent XML operations on related objects, a short-term cache can be used. Problems related by caching and write operations to create or delete objects are subject for future work.

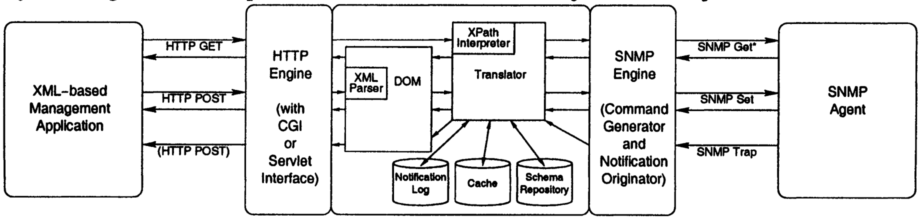


Figure 4. Architecture of an SNMP/XML gateway.

6. Conclusions and Outlook

This paper gave a condensed overview of some XML technologies and how they can be used to solve network management tasks that are partly difficult to address with the current SNMP framework as it is. A number of related projects and standardization efforts that are underway have been presented. However, it has been argued that a smooth bridging between the widely deployed SNMP infrastructure and the 'new generation' of XML based solutions is essential.

The presented approach to represent management data received from and stored to SNMP agents as XML documents and to represent their structure as automatically generated XML Schema definitions based on SMI MIB modules constitutes one step in this direction. In contrast to other attempts that define a simple straightforward mapping of SMI data models to DTDs or schemas the presented approach is driven by the goal to tap the full potential of XML and XML Schema leaving drawbacks of SNMP behind where possible. Furthermore, this paper presents some visions on other XML applications. Within the presented project, an SNMP-to-XML gateway is under development that represents management data conforming to the presented XML Schema model.

The standardization work done so far in the area of XML/SNMP integration is not much more than the discussion of requirements and the evaluation of some technologies. In the future, more work on specific schemas and protocol support has to be done to support the higher-level tasks in XML based network management and configuration management.

References

[1] M. Rose and K. McCloghrie. Structure and Identification of Management Information for TCP/IP-based internets. RFC 1065, TWG, August 1988.

[2] K. McCloghrie and M. Rose. Management Information Base for Network Management of TCP/IP-based internets. RFC 1066, TWG, August 1988.

[3] J. Case, M. Fedor, M. Stoffstall, and J. Davin. A Simple Network Management Protocol. RFC 1067, University of Tennessee at Knoxville, NYSERNet, Rensselaer Polytechnic Institute, Proteon, August 1988.

[4] J. Schönwälder and A. Pras. The Simple Times. An openly-available online publication on SNMP, http://www.simple-times.org/.

[5] J.-P. Martin-Flatin. *Web-Based Management of IP Networks and Systems*. Wiley, 2002.

[6] H. Ju, M. Choi, S. Han, Y. Oh, J. Yoon, H. Lee, and J. W. Hong. An Embedded Web Server Architecture for XML-Based Network Management. In *Proc. 2002 IEEE/IFIP Network Operations and Management Symposium*, April 2002.

[7] T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler. Extensible Markup Language (XML) 1.0 (Second Edition). W3C Recommendation, October 2000.

[8] J. Clark. XSL Transformations (XSLT) Version 1.0. W3C Recommendation, November 1999.

[9] D. C. Fallside. XML Schema Part 0: Primer. W3C Recommendation, IBM, May 2001.

[10] H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. XML Schema Part 1: Structures. W3C Recommendation, University of Edinburgh, Oracle, Lotus, May 2001.

[11] P. V. Biron and A. Malhotra. XML Schema Part 2: Datatypes. W3C Recommendation, Kaiser Permanente, Microsoft, May 2001.

[12] R. Sprenkels and J. P. Martin-Flatin. Bulk Transfer of MIB Data. *Simple Times*, 7(1), March 1999.

[13] T. Berners-Lee, L. Masinter, and M. McCahill. Uniform Resource Locators (URL). RFC 1738, CERN, Xerox Corporation, University of Minnesota, December 1994.

[14] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifiers (URI): Generic Syntax. RFC 2396, MIT/LCS, U.C. Irvine, Xerox Corporation, August 1998.

[15] L. Wood, et al. Document Object Model (DOM) Level 1 Specification Version 1.0. W3C Recommendation, Soft Quad, October 1998.

[16] Y. Oh, H. Ju, M. Choi, and J. W. Hong. Interaction Translation Methods for XML/SNMP Gateway. In *Proc. 13th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, October 2002.

[17] J. Clark and S. DeRose. XML Path Language (XPath) Version 1.0. W3C Recommendation, Inso Corp., November 1999.

[18] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. W3C Note, Microsoft, IBM, March 2001.

[19] N. Mitra. SOAP Version 1.2 Part 0: Primer. W3C Working Draft, Ericsson, June 2002.

[20] M. Gudgin, M. Hadley, N. Mendelsohn, J. Moreau, and H. F. Nielsen. SOAP Version 1.2 Part 1: Messaging Framework. W3C Working Draft, DevelopMentor, Sun, IBM, Canon, Microsoft, June 2002.

[21] M. Gudgin, M. Hadley, N. Mendelsohn, J. Moreau, and H. F. Nielsen. SOAP Version 1.2 Part 2: Adjuncts. W3C Working Draft, DevelopMentor, Sun, IBM, Canon, Microsoft, June 2002.

[22] W3C. W3C – The World Wide Web Consortium. WWW Page, 2002. http://www.w3c.org/.

[23] Juniper Networks. JUNOScript API Software. WWW Page, November 2002. http://www.juniper.net/support/junoscript/.

[24] Avaya Labs Research. XML-Based Management Interface for SNMP Enabled Devices. WWW Page, 2001. http://www.research.avayalabs.com/user/mazum/Projects/XML/.

[25] J. Schönwälder. Specific Simple Network Management Tools. In *Proc. LISA 2001*, pages 109–119, December 2001. http://www.ibr.cs.tu-bs.de/projects/scli/.

[26] K. McCloghrie and F. Kastenholz. The Interfaces Group MIB. RFC 2863, Cisco Systems, Argon Networks, June 2000.

[27] K. McCloghrie. SNMPv2 Management Information Base for the Internet Protocol using SMIv2. RFC 2011, Cisco Systems, November 1996.

[28] K. McCloghrie and A. Bierman. Entity MIB (Version 2). RFC 2737, Cisco Systems, December 1999.

[29] M. Wasserman. XML Configuration BOF. WWW Page, July 2002. http://www.ietf.org/ietf/02jul/xmlconf.txt.

[30] OASIS. Management Protocol TC. WWW Page, December 2002. http://www.oasis-open.org/committees/mgmtprotocol/.

[31] F. Strauß. Libsmi – A Library to Access SMI MIB Information. WWW Page, December 2002. http://www.ibr.cs.tu-bs.de/projects/libsmi/.

GRID OBJECT DESCRIPTION: CHARACTERIZING GRIDS

Gerd Lanfermann
Max Planck Institute for Gravitational Physics – Albert Einstein Institute
Am Mühlenberg 1, 14476 Golm, Germany
lanfer@aei.mpg.de

Bettina Schnor
University of Potsdam, Dept. of Computer Science
August-Bebel-Straße 89, 14482 Potsdam, Germany
schnor@cs.uni-potsdam.de

Edward Seidel
Max Planck Institute for Gravitational Physics – Albert Einstein Institute
Am Mühlenberg 1, 14476 Golm, Germany
eseidel@aei.mpg.de

Abstract: We present a new data model approach to describe the various objects that either represent Grid infrastructure or make use of it. The data model is based on our experiences and experiments conducted in heterogeneous Grid environments. While very sophisticated data models exist to describe and characterize e. g. compute capacities or web services, we will show that a general description, which combines *all* of these aspects, is needed to give an adequate representation of objects on a Grid. The ***Grid Object Description Language*** *(GODsL)* is a generic and extensible approach to unify the various aspects that an object on a Grid can have. GODsL provides the content for the XML based communication in Grid migration scenarios, carried out in the GridLab project. We describe the data model architecture on a general level and focus on the Grid application scenarios.

Keywords: Grid Computing Information Model, Interoperability, Grid Migration

Introduction

Computational Grids are becoming increasingly common, promising ultimately to be ubiquitous and thereby change the way global resources are accessed and used. We have investigated several large scale Grid scenarios for autonomic applications, based on a previous prototype, dubbed the "Cactus Worm" [1]. Two of the major ones are *nomadic migration* [18], which describes the autonomic migration of applications from one compute resource to another, and *application spawning* [20] which allows an application to accelerate the main execution thread by taking advantage of workflow parallelism. In this case internal routines are "outsourced" to external compute resources. These scenarios are realized through service environments, which offer file transfer, resource selection and job submission capabilities to autonomic applications by interfacing with a wide spectrum of existing infrastructure of Grid middleware, batch systems and basic transport and shell methods.

The "Grid substrate" that these services operate on is heterogeneous in availability and type. We address service availability issues in [19, 20] through a fault-tolerant service topology and focus here on the problem that all objects which constitute the Grid come with different access types, batch submission systems, resource selection processes etc. Clients of such an environment are e. g. characterized through their non-web service compliant functionality or resource requirements.

The problem of heterogeneous environments is well known and projects like Globus or Legion address it through a Grid middleware that provides a uniform access to machines. However, many sites do not install this software, they fail to maintain it properly, or test installations are never integrated into the normal production environment. Nevertheless, most of these sites support launching of jobs through the standard login procedure and batch submission systems. The direct interaction with Grid middleware is often the only way to access resources if sophisticated packages like Globus fail.

Before we can integrate existing Grid infrastructure as well as proprietary user applications, we need a way to describe such "objects" on a Grid. This description goes beyond the current scope of web service or resource characterization. While sophisticated information models exist for the individual aspects, we are lacking a lean data model, that combines these aspects into a single entity which is simple enough to be used not only for server communication but also within client applications. In the following case study, we will motivate such a common description system with a concrete Grid migration scenario.

1. The Grid Migration Scenario

Nomadic Migration denotes the self-controlled and automatic relocation of large applications to provide faster or more efficient code execution [20]. *Nomadic* illustrates the low frequency of this event and the self-contained operation: It is an application-initiated and self-determined process in a service environment. Nomadic migration is investigated by the Max Planck Institute for Gravitational Physics to relocate numerical relativity simulations to increase the job throughput of their large-scale computations [2].

Migration mechanisms and strategies were well studied in the context of cycle-stealing clusters in the mid 90s (see for example [22, 24, 3]). The motivation for migration has more or less stayed the same: *Administrative reasons*, when the compute requirements of an application exceeds the queue time limits offered at a compute center and *Performance reasons* when "better" compute capacity becomes available. Our scenario goes beyond the scope of migrating within a single cluster or a homogeneous intra-Grid machine pool: we relocate across global Grids.

Traditional Job Re-submission. If the resource demands of an application exceed the granted limits (e. g. memory or processors), the application needs to relocate to a different machine or batch queue. The user engages in a tedious process of instructing the application to checkpoint, securing the checkpoint files and archiving them if necessary. If the application can only be continued on another host, the checkpoint files need to be transferred. After deriving the new resource requirements the program is resubmitted to the queuing system. At all steps, the user's involvement makes the process prone to failure:

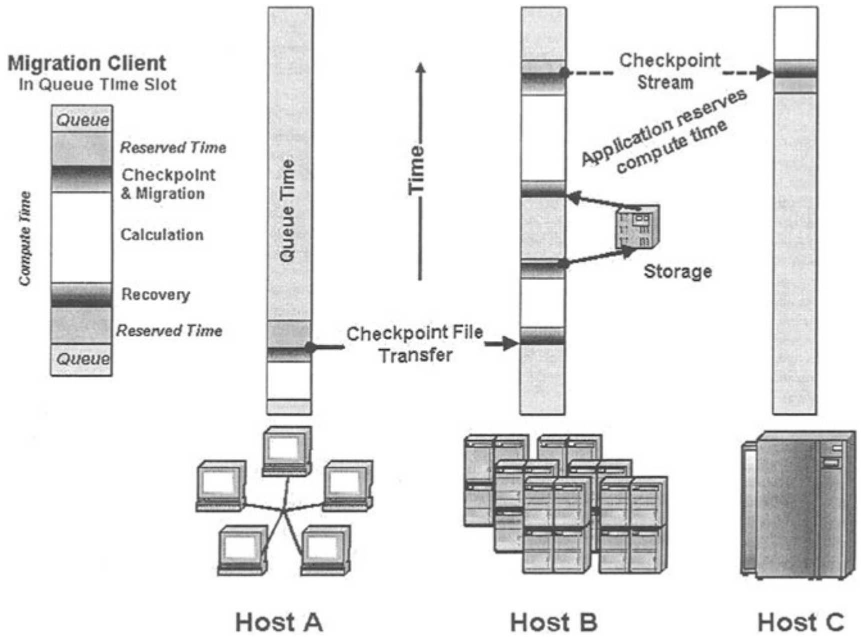


Figure 1. "Nomadic Migration".

1 Checkpoint files can be erased by disk purging policies and lack of bandwidth can make the checkpoint transfer unacceptably slow.

2 Resource requirements have to be correctly analyzed, otherwise the job will be rejected by the resource management system, either at submission time or worse – during runtime. Conservative estimates of resource requirements usually lead to longer waiting times.

3 The researcher is required to remember usernames and passwords as well as the interfaces to a wide range of different machines, architectures, queuing systems and shell programs.

From experience, the overhead of this procedure encourages the researcher to resubmit a job on the same machine, discarding potentially faster machines.

Automated Job Re-submission. The process of resubmitting to an arbitrary host, which fulfills the application's minimum resource requirements, is a prime candidate for automation: In Figure 1 we illustrate a migration process for an application. The migration progresses from left to right across three different host types *A*, *B* and *C*, indicating a network of workstations, a cluster of PCs and a traditional supercomputer, respectively. The left inset shows the different phases of an application in a queue: a migration client receives reserved compute time, called a "slot". Within this time slot, all code execution has to take place: the recovery from a previous checkpoint, the calculation phase, and saving the new state to a new checkpoint file. Applications which exceed their time slots are terminated.

In the illustration an application is started on host A. A migration server (not shown), which is responsible for the relocation of applications, receives information on the client's resource consumption and its location. It monitors the availability of new machines which meet the requirement. As "better" resources become available, the application on A is informed and it checkpoints. The checkpoint files are transferred to host B. The migration client is restarted or submitted to the queuing system. As the application runs out of compute time on B, the last checkpoint is archived in a storage facility and the program is resubmitted to the queue on the same machine. The application will execute a second time on B.

Some advanced applications are able to receive the checkpoint as a socket stream instead of reading from file. In combination with advanced reservation scheduling, this transfer mode allows for fast checkpoint transfer, shown in the migration to machine C: The program on B is aware of the expiration of its time slot and requests in advance a slot on machine C, which overlaps with the compute slot on B. By the time the application is about to finish on B, the migration server starts an uninitialized application on C, which receives the application state through the streamed checkpoint and continues the calculation. GridLab [25] is currently investigating these prototypes to study the functionality which they will make available in a generic way to a wide class of applications.
This case study identifies some of the problems that have to be solved:

- *File Transfer*: A migration service stages executables and other files to a new host. Therefore, it must be able to determine and support different file transfer methods for each of the source and target machines.

- *File Description*: The service must provide a compact description of multiple files, including information on where the files are located and how they can be accessed.

- *Job Submission:* The migration server is responsible for submitting the new job to individual queuing systems.

- *Resource Description*: It is essential to characterize resource requirements of an application as well as resource capacities of machines or queues. The migration service must be able to evaluate and compare them.

- *Hardware Description*: A migration service must describe the location of machines, including the ways to access it.

2. Related Work

Quite a number of different software packages exist to assist in interfacing with heterogeneous environments. Resource management systems are primarily aimed at users, while web services and related technology are designed for applications. In this section we list the most important packages and discuss how we can incorporate their functionality in our information model.

2.1 Resource Management Systems and Tools

These systems focus on describing the compute capacities of machines and the requirements of applications. They provide advanced scheduling policies to achieve high job throughput and provide a uniform access to machines. Typically each of these submission systems comes with its own way of characterizing resources.

Globus. Globus [13] makes a significant contribution to the Grid infrastructure. We use Globus functionality for our migration service wherever and whenever possible. Globus requires a working and maintained installation. Within the Globus installation base, a well specified job submission and resource selection system is supplied to the user.

The Globus Resource Specification Language (RSL) [23] provides a common interchange language to describe resources. It is used to define resource requirements for applications which are submitted through the Globus Resource Allocation Manager (GRAM) [16]. The Meta Directory Service (MDS) [10] is the information service component of the Globus Toolkit. MDS services can deposit and extract information on the current resource situation of a machine or queue. MDS is based on LDAP and represents information as a set of objects organized in a hierarchical namespace, in which each object contains key-value pairs. MDS introduces yet another set of vocabularies to express resources.

Condor. The Condor [4] system is designed as a "cycle-stealing" middleware to harvest unused compute capacity for single processor applications. It supports transparent checkpointing and operates primarily on homogeneous machines within a single administrative domain, called a 'flock". Condor-G [14] is a modified Condor package, that joins multiple condor "flocks" from diverse sites through Globus.

Condor Classified Advertisement (Class-Ad) is a sophisticated data model to describe resources in general and includes a matchmaking ability which makes it a valuable tool to compare resource requirements with constraints. Condor does not come with a fixed set of attributes but requires the information in the Class-Ad syntax.

Batch Submission System. Packages like PBS, LSF, Load Leveler, Sun Grid Engine, etc. are responsible for submitting the application to a computer. They come with a proprietary script syntax and offer different capabilities. Submission scripts are traditionally prepared by users – either manually or through graphical user interfaces.

2.2 Web Services

Most of the services listed above are designed to be accessed by a user – either from the command line or through graphical interfaces. They are not primarily designed to be accessed by an application. The *application-centric* utilization of services has been a very recent trend and builds on the concept of web services. Technologies like "eXtensibleMarkup Language" (XML) [6], "Simple Object Access Protocol" (SOAP) [5], XML-Remote Procedure Call (XML-RPC) [27], "Web Service Description Language" (WSDL) [7] and "Universal Description Discovery and Integration" (UDDI) [26] enable applications to autonomously connect to other programs to exchange information and services. WSDL, which is an XML language used to describe network service endpoints, has become the standard to describe web services. The recently released Open Grid Service Architecture (OGSA) [12], which is a specification that integrates the concepts of web services and Grid technology, builds heavily on these web service standards.

2.3 Related Information Models

There are some unification efforts for the description of Grid entities under way: Globus GRAM provides interfaces to address the most common batch submission systems. The Global Grid Forum features the Job Submission Description Language (JSDL) working group which investigates a standard description for job submissions to incorporate the different resource dialects. The Common Information Model (CIM) [8] is developed by the Distributed Management Task Force (DMTF). CIM's data model is an implementation-neutral scheme for describing management information in a network/enterprise environment. CIM has similarities regarding our goal to provide a common description system.

The JSDL is similar in spirit to what GODsL aims for, but is restricted to job description. The Globus RSL, which unifies access to different submission systems through Globus GRAM has only rudimentary support for file information. CIM information models on the other hand offer a functionality which reaches far beyond what we feel is necessary to describe entities on the Grid and which makes it difficult to deploy such models on clients. We follow a more simplistic approach to describe Grid Objects.

3. Grid Objects

The design of the Grid Object Description Language is based on our experiences with the migration service environment and in particular on these observations:

- To use an information model, we cannot require or rely on the global installation of a single Grid middleware. Because of the *administrative autonomy* of sites, we will be faced with anything from modern Grid web services to traditional interactive commands.
- Web service technology has excellent solutions for providing information on services. But our system must accommodate "legacy-services", like ssh-based machine access as well. Furthermore, we need to describe non-web service compliant user programs and have to associate service descriptions with file or machine information.
- We require a solution which abstracts a compute capacity from the hardware that provides it and which is able to describe resource requirements as well.
- A description of files must be included for migrating an application: a checkpoint file contains the "hibernating" state of a migrating application and is of equal importance as the application's executing state, which can be characterized as a service.

With this motivation, we do not attempt to reproduce any of the software packages or Grid middleware listed above. We target a uniform description of the various systems and their abilities. We motivate an information model that allows for a precise definition of an "object" on the Grid independently of the circumstances under which it is realized: this object can be a distributed file, a compute capacity offered by a machine or the resource requirements of an application. We attempt to capture these different aspects in a single data model.

3.1 Grid Object Examples

The following examples illustrate how the different aspects of items on the Grid are interchangeable:

1 The information on the name and directory of a data file is only sufficient if we know on which physical machine the file resides. However, we have no information on how we can access that particular computer, we are not able to look at the file, copy it or treat it in any other way.

2 The file on that machine may have been generated by an application. Unsurprisingly, the machine which hosts the service is identical with the machine location of the file.

3 The application has a minimal resource requirement, and a computer may provide resource capacities. The host can execute the application if and only if the hardware's resource capacity is greater or equal than the application's resource requirement.

4 If requirements and constraints are matching, the application can be hosted on that particular hardware. The description of the application remains the same, but the host information changes.

In these simple examples we can identify four core components of an object, which are service, hardware, file and resource properties.

3.2 Grid Object Description Language

We propose *Grid Objects* as a general description of objects on a Grid. We call the structure to describe such an entity the *Grid Object Description Language.* A Grid Object is a collection of sub-objects, termed a *Container*, each of which holds one or more *Profiles* that reflect the different properties of this object, as illustrated in Figure 2. The container structure in a Grid Object is optional and depends on the background situation that is described.

For our purposes we define a Grid Object as follows:

$$\begin{aligned}\text{Grid Object} \rightarrow \text{UId Label } &\{\text{Machine Profile}\}\ \{\text{Resource Profile}\}\\ &\{\text{Service Profile}\}\ \{\text{File Profile}\}\end{aligned}$$

The profiles and container can be optional. Multiple profiles are collected in a *container* structure. Profiles can be copied (example 2, in which the file's machine profile can be inherited from the description of the application) or compared (example 3, where two profiles are put in relationship). Profiles can be added or replaced (example 4, adding or replacing the description of the hardware as the application starts or migrates, respectively).

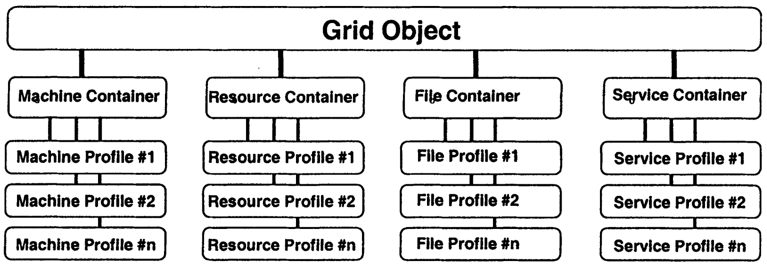


Figure 2. Grid Object structure

3.3 Machine Profiles

Machine profiles contain the information which are obligatory to address the hardware on the internet. Typical information items are the type of the hardware, the hostname, the domain name and the IP address. Multiple machine profiles can be grouped in a *Machine Container*. A machine container can e.g. describe a network of workstations or the participating machines of a meta-computer.

3.4 Resource Profiles

The *resource profile* structure characterizes the compute and storage capabilities or requirements of a Grid Object. Key attributes of a resource profile are e. g. the *number of processing elements*, *memory*, information on the *operating system* and *CPU* type.

The reason to treat the resource characteristics separately from a machine profile is justified by the fact that resource profiles are not necessarily tied to a hardware device. Instead they can be used to characterize abstract resource situations: An application may provide a resource profile to interested parties to relay information on its *resource consumption*. In this context the resource profile is not attributed to a hardware (machine profile) but to an application, which is described through its location (machine profile) and functionality (service profile). As hinted above, multiple resource profiles may be used to characterize the different batch queues which partition the total compute resource of a machine, specified through a machine profile.

3.5 Service Profiles

A *Service profile* is a description of a functional ability which is available on a machine or provided by an application, usually through a port or range of ports. We aim at describing modern web services as well as traditional interactive commands like secure-shell based access to a hardware or queue submission system. Key attributes, which are used to describe such a profile are *operation* for traditional command line programs, *binding* specifying contact information on web service based operations, *method* the name of the method under which a rpc-typed service can be accessed, followed by port information. Multiple service profiles can be grouped into a *Service Container*. Such a container may e. g. list all access methods (e. g. ssh, rsh, http, Globus gatekeeper) that permit access to a hardware.

The Grid Object information model that we suggest does not try to supersede or copy WSDL technology. Instead we look for a pragmatic way to include non-webservice compliant, legacy applications as well as proprietary user codes in the description of Grid entities. While WSDL has some capabilities to store service related information, we do not regard WSDL as a suitable place e. g. to store the description of a migratable entity (application, checkpoint, parameter files). We would suggest using WSDL for describing the interface to retrieve such an object description from a database.

3.6 File Profiles

File profiles describe properties of files and directories. Their main attributes are *Filename* and *Directory* to locate files on a filesystem. *Size* and *Compactification* information allow a transfer operation to favor certain services over others or to rule out potential target hosts, e. g. due to the lack of bandwidth.

3.7 Grid Objects

The different profiles and containers that we introduced above are combined to provide a unified description of the different aspects of objects on the Grid. The Grid Object structure collects the various containers into a single entity. Container information is optional, containers are appended to the Grid Object depending on the situation which is described. The Grid Object may hold a *Machine Container*, *Service Container*, *File Container* and *Resource Container*. Each component (profile, container, object) provides space for a human readable label and a unique identifier (UId) to distinguish the different components in a machine readable style. Multiple Grid Objects can be collected into an array of Grid Objects, called a *Grid Object Container*.

The classification into the four fundamental types is inspired by our underlying migration scenario, whose communication content we intend to express with this data model. For other scenarios, different profiles may become important while others can be omitted. In section 5 we suggest several important additions to the Grid Object information model.

3.8 GODsL Toolkit

The *GODsL Toolkit (GODsL-Tk)* provides a number of routines to manage Grid Objects, container and profile structures for the C programming language. It can be used to copy profile and container constructions and attach them to already existing objects. The toolkit provides routines to serialize the Grid Objects from C to XML and deserialize objects in XML into the appropriate C structures. Our migration service is such an XML-RPC based service, which uses XML representation of Grid Objects as data for the migration requests.

4. Grid Objects Applied: Grid Migration Service

In this section we give actual examples of how Grid Objects are used in the context of the migration service, which we introduced as our main motivation for developing this data model.

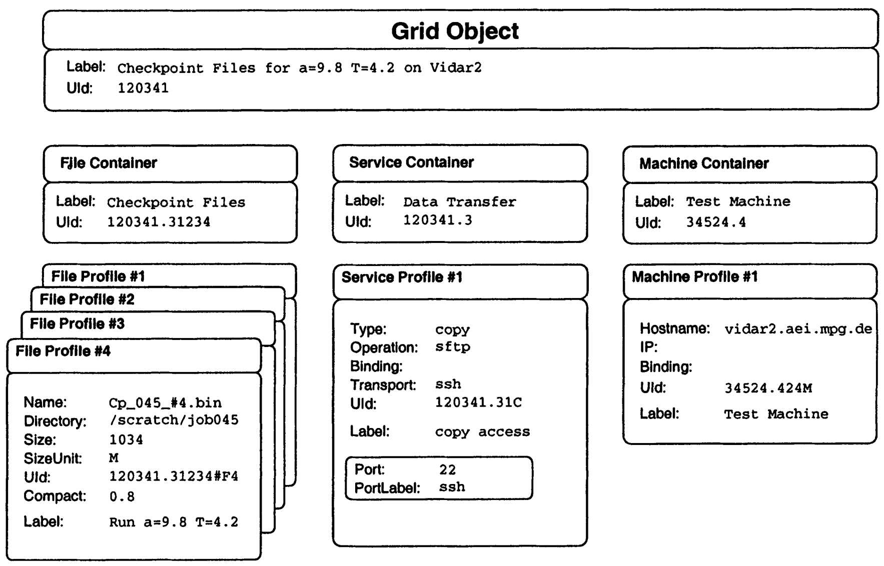


Figure 3. Grid Objects describe distributed files generated through parallel I/O as well as the hosting machine and the access methods.

4.1 Multiple Files on a Single Machine

Large data files in parallel applications are usually brought to disk through parallel I/O methods to speed up the process of data writing. It has the drawback in that it generates multiple files, as sketched in Figure 3, which shows the four checkpoint file chunks which compose a single logical checkpoint. In the same way that a service container can hold several access profiles for a machine, we can group multiple file profiles in a file container.

In Figure 3 we show the resulting Grid Object, which holds the file container, that stores the four file profiles. The Grid Object can be completed with information *where* the file is located through a machine profile and *how* it can be accessed: this service profile identifies secure-shell ftp as the only transfer method. We can now pass such a Grid Object as the source part in a copy request to a migration server, e. g. to store the files in a storage facility. With the method call, the copy service receives all the necessary information to move this file from the specified machine to a new site. The service can also decline the copy operation, if it does not provide the transfer methods which are required to access the files on the machine.

4.2 Resource Identification, Evaluation and Request

Before any job is executed on a supercomputer, a three stage process needs to be completed, which is traditionally done by the user, interactively and intuitively: The user determines the resource requirements of his application, typically through educated guesses or trial and error. Secondly, the user has to familiarize himself with the compute capacities on the machines in question. This knowledge is usually gathered by reading up on the site's batch submission configuration. The user then chooses a particular supercomputer, where he selects a queuing system, whose constraints will not be violated during the runtime of the program. The user requests the resources, usually by filling out a batch submission script, in which he states the requirements, like number of processors and memory. The user has to obey the particular queue syntax. The job is finally submitted.

If we express the diverse resource requests and constraints, job submission interfaces, etc. as profiles in a Grid Object, we have accomplished a first step to automate this user-centric process. In Figure 4 we show the use of Grid Objects in each of the three stages and explain the mapping between the different vocabularies of MDS [10], Condor's Class-Ads [9], PBS [17] and LSF [29]. The attributes of a resource profile (abbreviated `RP`), a service profile (`SP`) and a machine profile (`MP`) are being exchanged with the corresponding third party vocabulary. The automated process consists of these procedures:

Resource Identification. In the identification phase, the available resources are reported by monitoring systems like MDS, and the mapping of the MDS vocabulary to the Resource profile's attributes is performed. This step is illustrated in the top section. If more than one resource is obtained, the lookup service provides a Grid Object for each of the results. The resource requirements of an application can be measured at runtime with tools like Performance-API [21] and are also expressed in a resource profile of a Grid Object (bottom part).

Resource Evaluation. To determine which resource provides the best compute capacity for a given resource constraint, we prefer using existing technology, like Condor's Class-Ads: A migration service rewrites the resource profile of the application and the resource profiles of the available resources as Class-Ads. The Class-Ads parsing algorithm compares the job requirement to the constraints and returns a match if possible.

Resource Request. When a match is found, the migration server can proceed to submit the job and needs to fill out the proper batch submission script. Depending on the scheduling system found on the particular machine, the server provides the resource requirements as arguments to the batch scheduler: The content of the resource profile is now mapped onto the vocabulary of the batch submission systems. As shown for PBS and LSF in the bottom part of Figure 4, this is a straightforward translation process.
Reusing existing software is an indispensable ability and the Grid Object Description Language permits to converse with already existing software packages and grid middleware. For example, a statement on the amount of memory is expressed as `physicalmemorysize` (MDS), `Memory` (Class-Ads, user defined), `#BSUB -M` (LSF directive), `#PBS -l mem` (PBS directive). The mapping between different vocabularies is performed through modules in the individual programs.

5. Conclusions and Future Research

In this paper we have described the hierarchical and modular architecture of the Grid Object Description Language. Grid Objects represent the different aspects of an

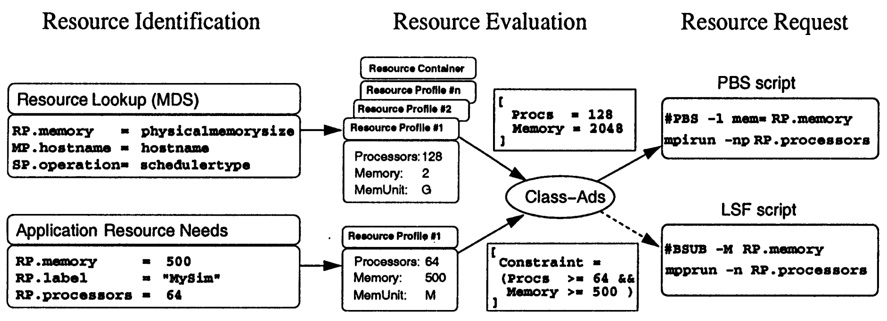


Figure 4. Grid Objects provide the translation capabilities to make use of already existing resource management technology. Resource profiles are used as a mediator between the different systems.

abstract entity on the Grid. Its hierarchical structure allows Grid Objects to scale with the increasing complexity of the described situation. Profiles and container components can be exchanged between Grid Objects which permits uncomplicated generation, updating and management of Grid Objects. The attributes of its profiles can be mapped onto existing Grid middleware in a straightforward manner, which is an important requirement to take advantage of sophisticated legacy middleware, which are omnipresent in today's Grid infrastructure.

We have illustrated in several examples that Grid Objects can be used to describe complex situations on Grids in a flexible way. These examples are taken from experiments with the Grid Migration Service, which uses Grid Objects to communicate the state of files, services and resources. Migration experiments have been conducted on the European Testbed (EGrid) [11] to develop and test communication through Grid Objects.

Open Grid Service Architecture. The Grid Object description approach falls short in respect to the capabilities of today's web service technology. For the future development of GODsL, the interfacing with web service technologies, like OGSA, will become an important field of research. The OGSA code base has been changing quite drastically over time and has now reached a stage where it has started to stabilize. We intend to take advantage of OGSA services in migration servers, for example to utilize Grid copy operations.

Network Profiles. The current version of GODsL does not support the characterization of network performance. The Network Weather Service [28] is a distributed system that periodically monitors and dynamically forecasts the network conditions like bandwidth. Currently, we investigate, how Grid Objects can be extended through *Network Profiles* [15], which will allow a migration service to select a potential migration host based on its network connectivity. Such a profile should also be capable to typify *internal* networks to give users a way of requesting a preferred network interconnect within a cluster or parallel computer.

Time. Future work will include the notion of time and time intervals. The present system of profiles is static in the sense that profiles do not timeout or become invalid.

Especially in the field of meta-computing and advanced reservation scheduling it is necessary to describe the beginning and expiration time of a resource. We are working on methods to complete the Grid Object structures to express these dynamic properties.

With GODsL we do not intend to provide the community with yet another way of solving the problems of resource detection or web service description. However, from our application background, we see the need to express a unifying view on general objects on a Grid: We see compelling demand to glue together both the advanced and historical software technologies, which are the fabric of today's Grid infrastructure, as well as user applications. The Grid Object Description Language suggests a solution to this problem.

Acknowledgments

We are pleased to acknowledge support of the European Commission 5th Framework program, which is the primary source of funding for the GridLab project, but also the German DFN-Verein, Microsoft, the NSF ASC project (NSF-PHY9979985), and our local institutes for generous support for this work. We also thank Gabrielle Allen, Thomas Dramlitsch and Ian Kelley for helpful discussions. We are grateful for using the compute resources of the EGrid and at NCSA.

References

[1] G. Allen, D. Angulo, I. Foster, G. Lanfermann, C. Liu, T. Radke, E. Seidel, and J. Shalf. The Cactus Worm: Experiments with Dynamic Resource Discovery and Allocation in a Grid Environment. *Int. J. of High Performance Computing Applications*, 15(4):345–358, 2001. `http://www.cactuscode.org/Papers/IJSA_2001.pdf`.

[2] G. Allen, T. Goodale, G. Lanfermann, T. Radke, E. Seidel, W. Benger, C. Hege, A. Merzky, J. Massó, and J. Shalf. Solving Einstein's Equations on Supercomputers. *IEEE Computer*, 32(12):52–59, 1999.

[3] A. Barak, A. Braverman, I. Gilderman, and O. Laaden. Performance of PVM with the MOSIX Preemptive Process Migration. In *Proceedings of the 7th Israeli Conference on Computer Systems and Software Engineering*, pages 38–45, Herzliya, June 1996.

[4] J. Basney, M. Livny, and T. Tannenbaum. High throughput computing with Condor. *HPCU News*, 1(2), June 1997.

[5] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Nielsen, S. Thatte, and D. Winer. Simple Object Access Data Protocol (SOAP) 1.1. W3C Note, May 2000. `http://www.w3.org/TR/SOAP/`.

[6] T. Bray, J. Paoli, C. Sperenberg-McQueen, and E. Maler. Extensible Markup Language (XML) 1.0. W3C Recommendation, October 2000. `http://www.w3.org/TR/REC-xml`.

[7] E. Christensen, F. Curbera, G. Meredith, and S. Weerarawana. Web Service Description Language (WSDL). W3C Note 15, March 2001. `http://www.w3.org/TR/wsdl`.

[8] Common Information Model (CIM) Standards. The DMTF webpage: CIM Specification v2.7 and Standards, September 2002. `http://www.dmtf.org/standards/standard_cim.php`.

[9] The Condor Classified Advertisement. The Condor Webpage. `http://www.cs.wisc.edu/condor/classad/`.

[10] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid Information Services for Distributed Resource Sharing. In *Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing, (HPDC-10)*. IEEE Press, August 2001.

[11] T. Dramlitsch, G. Lanfermann, E. Seidel, A. Reinefeld, et al. Early Experiences with the EGrid Testbed. In *IEEE International Symposium on Cluster Computing and the Grid*, 2001. Available at `http://www.cactuscode.org/Papers/CCGrid_2001.pdf.gz`.

[12] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Service Architecture for distributed systems integration, June 2002. http://www.globus.org/ogsa/.

[13] Ian Foster and Carl Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputing Applications*, 11(2):115–128, 1997.

[14] James Frey, Todd Tannenbaum, Ian Foster, Miron Livny, and Steven Tuecke. Condor-G: A Computation Management Agent for Multi-Institutional Grids. *Journal of Cluster Computing*, 5:237–246, 2002.

[15] C. Fricke. Characterizing networks through the Grid Object Description Language. Student's Thesis, University of Potsdam, 2003. To appear.

[16] Globus Resource Allocation Manager. The Globus GRAM Webpage. http://www.globus.org/gram.

[17] R. Henderson and D. Tweten. Portable Batch System: External reference specification. Technical report, NASA Ames Research Center, 1996.

[18] G. Lanfermann, G. Allen, T. Radke, and E. Seidel. Nomadic Migration: A New Tool for Dynamic Grid Computing. In *Proceedings of the Tenth IEEE International Symposium on High Performance Distributed Computing, HPDC-10*, pages 435–436. IEEE Press, August 2001. http://www.cactuscode.org/Papers/HPDC10_2001_Worm.ps.gz.

[19] G. Lanfermann, G. Allen, T. Radke, and E. Seidel. Nomadic migration: Fault tolerance in a disruptive grid environment. In *Proceedings of the Second IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 280–281, 2002.

[20] Gerd Lanfermann. *Nomadic Migration – A Service Environment for Autonomic Computing on the Grid*. PhD thesis, University of Potsdam, Potsdam, 2003. To appear.

[21] K. London, J. Dongarra, S. Moore, P. Mucci, K. Seymour, and T. Spencer. End-user tools for application performance analysis, using hardware counters. In *International Conference on Parallel and Distributed Computing Systems*, August 2001. http://icl.cs.utk.edu/projects/papi.

[22] S. Petri and H. Langendörfer. Load Balancing and Fault Tolerance in Workstation Clusters – Migrating Groups of Communicating Processes. *Operating Systems Review*, 29(4):25–36, October 1995.

[23] The Globus Resource Specification Language RSL v1.0. The Globus Webpage. http://www-fp.globus.org/gram/rsl_spec1.html.

[24] B. Schnor, S. Petri, R. Oleyniczak, and H. Langendörfer. Scheduling of Parallel Applications on Heterogeneous Workstation Clusters. In Koukou Yetongnon and Salim Hariri, editors, *Proceedings of the ISCA 9th International Conference on Parallel and Distributed Computing Systems*, volume 1, pages 330–337, Dijon, September 1996. ISCA.

[25] E. Seidel, G. Allen, A. Merzky, and J. Nabrzyski. GridLab – A Grid Application Toolkit and Testbed. *Future Generation Computer Systems*, 18(8):1143–1153, 2002.

[26] Universal Description, Discovery and Integration (UDDI). http://www.uddi.org.

[27] D. Winer. XML-RPC Specification, June 1999. http://www.xmlrpc.com/spec.

[28] Richard Wolski. Dynamically forecasting network performance using the network weather service. *Cluster Computing*, 1(1):119–132, 1998.

[29] Songnian Zhou, Xiaohu Zheng, Jingwen Wang, and Pierre Delisle. Utopia: a load sharing facility for large, heterogeneous distributed computer systems. *Software - Practice and Experience*, 23(12):1305–1336, 1993.

SESSION 10

SLA / Quality of Service

Chair: Heinz-Gerd Hegering
University of Munich, Germany

POLICY SPECIFICATION AND ARCHITECTURE FOR QUALITY OF SERVICE MANAGEMENT

Nathan Muruganantha and Hanan Lutfiyya *
Department of Computer Science
The University of Western Ontario
hanan@csd.uwo.ca

Abstract: An application's Quality of Service (QoS) requirements refers to non-functional, run-time requirements. These requirements are usually soft in that the application is functionally correct even if the QoS requirement is not satisfied at run-time. QoS requirements are dynamic in that for a specific application, they change. The ability to satisfy an application's QoS requirement depends on the available resources. Since an application may have different QoS requirements in different sessions, the resources needed are different. A differentiated service must be supported. Since an application's QoS requirement is soft, it may not always be satisfied. It must be possible to dynamically allocate more resources. In an overloaded situation, it may be necessary to allocate resources to an application at the expense of other applications. Policies are used to express QoS requirements and actions to be taken when the QoS requirement is not satisfied. Policies are also used to specify actions to be taken in overloaded situations. Policies dynamically change. Supporting these policies is done through a set of distributed managed processes. It must be possible specify policies and have these policies distributed to managed processes. This paper describes how these policies can be formally specified and a management architecture (based on the IETF framework) that describes how the policies are distributed and used by the management system. We conclude with a discussion of our experiences with the management system developed.

Keywords: QoS Management, Policies, Application Management

1. Introduction

An application's Quality of Service (QoS) refers to non-functional, run-time requirements. A possible QoS requirement for an application receiving a video stream is the following: "The number of video frames per second displayed must be 25 plus or minus 2 frames". A QoS requirement is *soft* for an application, if the application is functionally correct even though the QoS requirement is not satisfied at run-time. Otherwise, the QoS requirement is said to be *hard* (e.g., flight control systems, patient monitoring systems).

The allocation and scheduling of computing resources is referred to as *QoS management*. QoS management techniques (e.g., [25]), such as resource reservation and admission control can be used to guarantee QoS requirements, since resource reserva-

*This work is supported by the National Sciences and Engineering Research Council (NSERC) of Canada, the IBM Centre of Advanced Studies in Toronto, Canada and Canadian Institute of Telecommunications Research (CITR).

tions are based on worst-case scenarios. This is useful for applications that have hard QoS requirements, but often leads to inefficient resource utilisation in environments that primarily have applications with soft QoS requirements.

Application QoS requirements are often dynamic in that they may vary for different users of the same application, for the same user of the application at different times, or even during a single session of the application. We developed a QoS management system that deals with soft and dynamic QoS requirements by providing management services (implemented by a set of management processes and resource managers) that support the following: (1) Detecting that an application's run-time behavior does not satisfy the application's QoS requirements. This is called a *symptom* and is considered a manifestation of a fault in the system; (2) Fault location, using a set of symptoms, determines a hypothesis identifying possible faults causing the violation of the QoS requirements; (3) Based on the fault and the system state, adaptation actions are chosen that may take either the form of resource allocation adjustments or application behavior adjustments. These adaptations depend not only on the cause of the violation, but also depend on the constraints (*administrative* requirements) imposed on how to achieve the QoS requirement. This is especially important in the case of overloaded conditions.

It should also be possible for the QoS management system to communicate to an application that it should adapt its behavior (e.g., change video resolution) if it is not possible to reallocate resources. The implication of different QoS requirements suggests that a differentiated allocation of resources should be allowed.

Policies are used to express requirements. A *policy* can be defined [20] as a rule that describes the action(s) to occur when specific conditions occur.

The QoS management system should be able to support the strategy described above as well as deal with changes in QoS requirements.

Our earlier work [11] described a policy-based QoS management framework and the initial results indicated that this approach is worth pursuing. This paper focusses on the definition and specification of different categories of policies and describes how they can be distributed to the appropriate entities in the QoS management system. This paper is organized as follows: Section 2 describes the different types of policies needed and the architecture needed to support the QoS management system to support the QoS management strategy described earlier. Section 3 briefly describes the implementation. Section 4 discusses the lessons learned. Section 5 discusses the related work. Finally a conclusion is presented.

2. Architecture

This section describes different categories of policies and architectural components (depicted in Figure 1) needed to support the strategy to QoS management described in Section 1. The management architecture used in this work is considered an adaptation of the IETF policy framework [14, 19, 22] and is a generalization of our architecture described in [11]. An example QoS requirement that will be used throughout the paper is the following:

EXAMPLE 1 *A video client is to receive video at a frame rate of 25 frames per second, plus or minus 2 frames.*

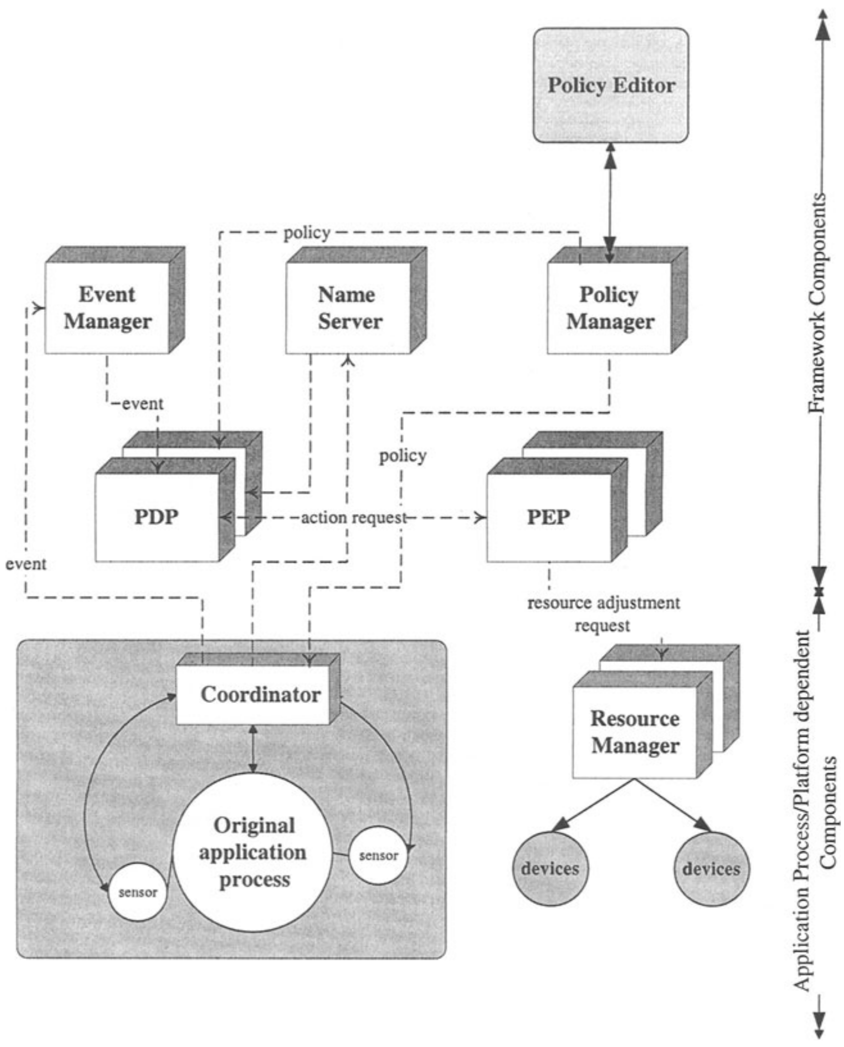


Figure 1. Policy-Based QoS Management Architecture

2.1 Expectation Policies

An *expectation* policy is used for stating how QoS requirements are determined and where to report the violation of a QoS requirement and any other possible actions. Expectation policies are specified using Ponder obligation policy formalism (for more information see [3]). In this formalism, the policy specifies the action that a subject must perform on a set of target objects when an event occurs. An expectation policy type associated with Example 1 is the following:

EXAMPLE 2

```
type oblig QoSreq_spec (target Coordinator, ScriptType FindQoSReqs){
  subject PolicyManager;
  on requestQoSRequirements(ProcessInfo)
  do FindQoSReqs(in ProcessInfo,
                 out EventIdentifier,
                 out AttributeConditionList,
                 out ActionList); -->
     Coordinator.Initialize(EventIdentifer,EventManagerIdentifier,
                            AttributeConditionList, ActionList);
}
```

This can be instantiated as follows:

```
inst VideoClientProcessReq=
    QoSreq_spec(VideoClientProcessCoordinatorSet,DetermineAllowedFrameRate);
```

It is assumed that each process has a management coordinator. The target domain for policy VideoClientProcessReq is the set of management coordinators of processes instantiated from the executable at .../syslab/rockyroad/videoClientExecutable (where ... refers to /ca/uwo/csd/syslab). It is not assumed that all instantiations are executing on rockyroad. DetermineAllowedFrameRate is a script that computes the frame rate per second that the process is allowed to have. Its input is process information (represented by ProcessInfo) that includes the user of the process and the time. The script returns the QoS requirements in AttributeConditionList. This is a list of attribute conditions. Application QoS requirements can be specified as a conjunction of the conditions specified in the attribute conditions. The general form of an attribute condition is the following: (anAttribute, comparisonOperator, aThreshold, EventIdentifier) where anAttribute denotes the attribute being monitored, aThreshold denotes a threshold that the value of anAttribute is being compared to, and comparisonOperator denotes the comparison operator by which the value of anAttribute and aThreshold are to be compared. EventIdentifier is the identifier of the event that is generated when this specific attribute condition is not satisfied (and hence the the QoS requirements are not satisfied). If an application process is to have the QoS requirements stated in Example 1, then the attribute condition list would be the following: ((current_fps, $\leq$, 27,fps_high),(current_fps, $\geq$,23, fps_low)). The attribute identifier, current_fps, represents frames per second. The event, fps_high, is generated when current frames per second is above 27 and fps_low is generated when the current frames per second is below 23.

The "calculation" or determination of QoS requirements can be guided by policies. A simple policy can be informally stated as follows: "if the process belongs to the GroupA users then it gets a target frames per second of 25 plus or minus two frames; if the process belongs to other users besides GroupA users then the target frames per second is 20 plus or minus two frames". This can be stated using Ponder formalism.

The ActionList denotes the action(s) the coordinator is to execute if the conjunction of the attribute conditions in the attribute condition list is found to be true. The general form of an action is the following: action = (targetObject, (actionMethod, actionMethodParameter)) A specific example is the following: (EventManagerHandle, (notify, current_fps, target_fps,...). Basically, this action specifies that an event handler process, that can be referenced using EventManagerHandler, is notified of the values of specific application attributes.

2.2 Monitoring

Example 2 illustrates that application QoS requirements are defined in terms of application-specific attributes. Monitoring of the application is needed to collect values of the attributes (e.g., current_fps). One monitoring mechanism is through instrumentation of the application which is briefly described here.

An attribute is associated with a sensor (see Figure 1). A sensor is a class with variables for representing threshold and target values. Sensors are used to collect, maintain, and process a wide variety of attribute information within the instrumented processes. The sensor's methods (*probes*) are used to initialise sensors with threshold and target values and collect values of attributes. Probes are embedded into process code to facilitate interactions with sensors.

As an example, consider the sample psuedo-code for a video playback application in Figure 2 that has the QoS requirement stated in Example 1 and sensor s_1. This QoS requirement suggests an upper threshold of 27 frames per second and a lower threshold of 23 frames per second. Sensor s_1 includes probes such as the following: (1) An initialisation probe (line 3 of Figure 2) that takes as a parameter the default threshold target value, and the lower and upper bounds. When the coordinator was instantiated (line 2 of Figure 2) it communicated with the QoS management system to get the application's QoS requirements in the form of the AttributeConditionList. Thus when the sensor requests this information, the coordinator will already have retrieved it. (2) A probe that (i) determines the elapsed time between frames and checks to see if this time falls within a particular range defined by the lower and upper acceptable thresholds. Unusual spikes are filtered out; and (ii) informs the coordinator if the frames per second fall below the lower threshold or is higher then the upper threshold.

When the coordinator retrieved the QoS requirements, it also retrieved the action list consisting of the actions in the form of the ActionList to be taken if the QoS requirement is not satisfied. In this example, this action is to notify the Event Manager process of the QoS management system whose handle is EventManagerHandle. The coordinator generates an event whose identifier is the event identifier found in the received action list.

We note that not all sensors measure attributes that are directly used in the specification of a QoS requirement. For example, a sensor may be used to measure the current size of the communications buffer.

2.3 QoS Management System

In the IETF framework, a Policy Decision Point (PDP) is used to retrieve stored policies (which is stored in the repository), and interpret and validate them. PDPs also make decisions on actions to be taken based on the receipt of an event which is generated from the monitoring of the environment. A Policy Enforcement Point (PEP) applies these actions. In this section, we will describe how PDPs and PEPs are used in this work and the additional components needed to support the QoS management strategy described in Section 1.

Name Server. When an applications starts up, it instantiates its coordinator. The coordinator then registers with the Name Server. The Name Server receives and maintains in a repository registration data from other components and application processes. It

```
Given: Video application v.
       QOS expectations e.
----------------------------------------
1.  Perform initialization for v.
2.  Initialise coordinator c.
3.  Execute $s_1 \rightarrow$ init_probe(e)
4.  while (v not done) do:
5.      Retrieve next video frame f.
6.      Decode and display f.
7.      Execute $s_1 \rightarrow$ probe_framerate()
8.  endwhile
```

Figure 2. Instrumentation Example

assigns a unique instance identifier for each registered process. The Name Server coordinates the interaction between the QoS management components and the application's coordinator.

Policy Decision Points (Policy Manager). Within an administrative domain, there is one Policy Decision Point (PDP) that communicates with a repository where policies are stored and information about users is kept. This PDP is referred to as a *Policy Manager*. The Policy Manager is responsible for retrieving and mapping high-level policies to rules and communicating the rules to the appropriate management entities. The need for this should become clearer later in this section. The Policy Manager keeps track of where policies were deployed to.

After the application has registered with the Name Server, it requests its QoS requirements from the Policy Manager. It does this by generating an event requestQoSRequirements(ProcessInfo). Upon receiving the request, the Policy Manager determines the policy that applies. If the policy that applies is Example 2 then the Policy Manager will execute the script DetermineAllowedFrameRate. The Policy Manager sends the QoS requirements and actions to be taken in the case of QoS violation to the application's coordinator as a condition list and action list which were described earlier.

The Policy Manager is the only type of PDP to receive events directly. The reason for this is the assumption that within an administrative domain there is only one Policy Manager and that the only type of event that it can receive is the requestQoSRequirements event.

Event Manager. The Event Manager receives events and allows other management entities to register an interest in an event. The Event Manager can receive events from the coordinators that are generated in response to a violation of QoS requirements and it can receive events that represent a regular report of monitored data e.g., resource usage information.

Policy Decision Points for Violation Location. In Section 1, we identified the need for violation location. In this work, the location process is event-driven in the sense that the location process does not begin until an event indicating that a QoS requirement is violated is generated. Conditions on the state of the system are evaluated to help in diagnosis.

PDPs are used to respond to a violation of a QoS requirement. This violation is an event. The PDP uses this event and other monitored information to diagnose the problem. Diagnostics are guided by *diagnosis* policies. A diagnosis policy associated with Example 1 is the following:

EXAMPLE 3

```
inst oblig fps_low1
{
   subject s = /ca/uwo/csd/brown/pdpServer;
   target t = /ca/uwo/csd/syslab/rockyroad/PEP;
      on fps_low(VideoClientInfo, current_fps, low_target_fps
                current_buffer_size, current_buffer_trend);
      do t.CPUIncrease(VideoClientInfo.ProcessIdentifier,current_fps),
      when registered(VideoClientInfo.ProcessIdentifier) and
                     (current_fps < low_target_fps) and
                     (buffer_size > bufferThresholdValue) and
                     (current_buffer_trend = UP);
};
```

This policy states that if the current frame rate (current_fps) is lower then the target frame rate (target_fps) and if the size of the client's communication buffer (current_buffer_size) is higher then a specific buffer threshold and if the client's buffer size (current_buffer_trend) tends to increase it should be requested that the resource manager at host *rockyroad.sylab.csd.uwo.ca* should increase the CPU priority of the video client. A policy fps_low2 is used to specify that if the event fps_low has occurred and the size of the buffer is relatively low and the CPU load of the machine that hosts the video server is high, then the manager process on the video server's host machine is to be instructed to increase the CPU priority of the video server process. These related policies are grouped into one composite policy.

There is a similar composite policy for when the current frame rate is higher then the higher bound allowed on the frame rate.

PDPs get the diagnosis policies associated with an application after the application registers with the Name Server. Upon receiving a registration request from an application process through its coordinator, the Name Server registers the application process with a PDP. There may be several PDPs. It is assumed that the administrator has determined how application processes are assigned to PDPs (which can be formalized as policies). The Policy Manager retrieves the diagnosis policies. However, Ponder policies are not executable. The user of Ponder makes it relatively easy for an administrator to use it, but it is too high-level to be executable by a PDP. An alternative formalism for specifying policies is needed. We refer to an executable policy as a rule. In this work, the alternative formalism was chosen to be JESS. An example of a JESS corresponding to the policy in Example 3 is the following:

EXAMPLE 4

```
( defrule fps_low_local
    ( registered process_id ?pid )
    ( fps low ?current_fps ?target_fps)
    ( and ( buffers high ?current_buffer )
         ( buffers up_trend ?current_buffer )
```

```
     )
=>
 bind ?amount ( resource_distance ?current_fps ?target_fps 10 )
( if ( > ( resource_conditions ?pid rtcpu ?amount ) 0.5 )
     then
     ( adjust_resource ?pid rtcpu ?amount ) )
  ( if ( < ( possibility_condition ) 0.1 )
     then
     ( assert ( service_diff rtcpu ) )
     )
  ( set_reset_delay 1 )
)
```

We will not go into details of what each specific part means. The point being made here is that formalisms that can be executable are usually more difficult to understand by administrators. Hence, the need for different formalisms.

JESS actually refers to a low-level rule engine and language. The Policy Manager will map the diagnosis policies specified using Ponder to JESS rules and then send the rules to the requesting PDP. Also sent to the PDP are the corresponding event identifiers that trigger the execution of rules. The PDP then registers its interest in the event identified by the event identifier with the Event Manager.

A PDP uses the set of rules to determine the corrective action(s). This involves determining the cause of the policy violation and then determining a corrective action(s). The process of determining the rules to be applied is called inferencing. Inferencing is used to formulate other facts or a hypothesis. Inferencing is performed by the Inference Engine component of the PDP. The inferencer chooses which rules can be applied based on the fact repository. The inferencing that can take place can either be as complex as backward chaining (working backwards from a goal to start), forward chaining (vice-versa) or as relatively simple as a lookup. In this work we used forward chaining.

A PDP is also used to authorize actions for PEPs on behalf of applications. This will be explained in more detail.

Policy Enforcement Point. A PEP is a management process that is responsible for monitoring the device that it is on and executing actions. A PEP receives requests from a PDP for a resource allocation. It verifies that it may execute the action, through the use of a PDP. This PDP uses *administrative* policies that formalize the administrative requirements.

Administration requirements can be specified using Ponder authorization policy constructs. In Ponder authorization policies, a policy is imposed by the target object, where the target can prohibit an action on itself based on different criteria such as subject, date and time. An authorization policy type associated with Example 1 is the following:

EXAMPLE 5

```
type auth+ authCPUIncreaseT (subject s, target t)
{
  action  CPUIncrease(ProcessIdentifier,normalizedvalue)
    when belongs(GroupA, ProcessIdentifier) and
    CPUResourcesAvailable(normalizedvalue);
}
```

This can be instantiated as follows:

```
inst auth+ allocateResourcePolicy=
  authCPUIncreaseT(/ca/uwo/csd/syslab/rockyroad/PEP,
              /ca/uwo/csd/syslab/rockyroad/ClientVideoProcess);
```

This policy states that PEP on .../syslab/rockyroad is allowed to increase the CPU priority for a video client process if the video client process belongs to GroupA and if there are enough CPU resources which is calculated by CPUResourcesAvailable. The parameters to this action include the process identifer (ProcessIdentifier and a normalized value (normalizedvalue) representing the difference between the attribute's current value and the expected value. The actual control of allocating CPU resources is encapsulated by a *resource manager*. There is a resource manager for each resource.

This is a simple policy and it is not application specific. It is host machine specific. A possible application-specific policy could be the following:

EXAMPLE 6

```
type auth+  authChangeApplicationResolution (subject s, target t)
{
  action ChangeResolution(ProcessIdentifier);
  when not(CPUResourcesAvailable(normalizedvalue));
}
```

This can be instantiated as follows:

```
inst auth+ authChangeApplicationResolution=
 authChangeApplicationResolution(/ca/uwo/csd/syslab/rockyroad/PEP,
                                 /ca/uwo/csd/syslab/rockyroad/ClientVideoProcess
```

This policy specifies that /ca/uwo/csd/syslab/rockyroad/PEP requests that the video client process's resolution is to be changed if there are not enough CPU resources available. This is an authorisation policy since this action should be prohibited except under specific circumstances.

Policy Editor. The Policy Editor provides a Graphical User Interface (GUI) for the administrators to compose and store high-level management policies (through the Policy Manager) in a policy repository.

QoS Management System Organization. It is assumed that each administrative domain has one policy server and at least one PDP. It is possible to have a configuration with one PDP on each device, a configuration with one centralized PDP or a configuration with more than one PDP, but not one on each device.

3. Implementation

We have built a C++ instrumentation library that implements a hierarchy of sensor classes. The base of the hierarchy provides a registration method that provides the ability to have all sensors register with the coordinator in a uniform manner. Other base methods are for enabling/disabling and read/report. C++ was used since we have an instrumentation library that has been developed and evolved over a period of time. There is a prototype of a Java instrumentation library that can be used for Java programs to interact with the QoS management system.

The repository used was LDAP. LDAP is used since this seems to be the choice of the IETF. The PDP, Event Manager, Name Server, and Policy Manager were all written in Java. The rule engine used by the PDP is JESS. The PEP has two components. The first component, implemented in Java, interacts with PDPs. The second component has specific resource managers such as the CPU manager and the Memory Manager. These resource managers actually manipulate the resources. This is written in C and is specific for Solaris. Currently, the CPU manager is integrated into the prototype.

The Policy Editor has a GUI that allows the user to enter policies The GUI was designed so that the user could be guided through the development of the policy by selecting whether they want to define event names, events, actions, attribute conditions (through the constraints). The policies were put into XML and references to the XML file were placed in the LDAP directory. XML was the language syntax used to express the Ponder policies. The user is able to directly put Ponder policies (in the Ponder format) in files and ask that they get loaded into the LDAP repository.

Translation is based on the existence of templates for each type of Ponder policy. The Policy Manager has a JESS template of rules associated with each Ponder policy. The Policy Manager uses this template to guide the transformation of a parsed Ponder component to a JESS rule or part of a JESS rule.

We have deployed the prototype within our laboratory and instrumented a number of applications. Currently, the prototype works as expected.

4. Discussion

This section describes our observations and experiences.

1 Section 2 describes how an application can be instrumented. It will not always be possible for a user or administrator to instrument an application themselves. They may have to rely on the output of logfiles that many applications generate. It is still possible to use the architecture of the QoS management system described in Section 3. The coordinator refers to a process that reads the logfiles generated by an application or uses the operating system to retrieve information about the application behavior.

2 Currently, PEPs determine if they are authorized to carry out an action by sending a message to the PDP. Potentially, the PDP could become a bottleneck. The PEP could have a rule engine to directly determine if it can carry out the action. This would reduce the load of the PDP, but increase the management load on the device that the PEP is on. This is not a problem in the sense that our architecture can easily handle this. Alternatively, multiple PDPs might be necessary. One PDP focusses on diagnosis and the other PDP focusses on messages from the PEP.

3 In our earlier work [11], diagnosis was distributed. This assumed that there were management processes on each host machine as well as a 'global' management process for an administrative domain. Rules to diagnose the cause of a violation of a QoS requirement were found in the local management processes as well as the 'global' management process. These management processes are similar to the PDPs. We found the translation of diagnosis policies to rules and the deployment of those rules was more complicated. This is the result of the translation having to take into account that if management process determines that it cannot resolve the problem, it needs to send an event to another management process so that it might try to resolve the problem. On the otherhand,

the use of one PDP results in many rules being sent to that PDP and it could become a bottleneck. More work is needed in understanding how to optimize the configuration of PDPs.

4 It is possible to have the different management entities interested in a specific event to directly register with the component generating the event i.e., the generator of the event acts as its own event manager. The disadvantage is that this assumes that these management entities know *apriori* where the events are coming from. Another possiblity is to have the policy specifications explicitly specify the source and receivers of a specific event. However, if we change the configuration of PDPs by changing their location and/or number then the policy specification should be changed. By separating these out, we can continue to use the same policy specification even if the underlying management system changes.

5 It is assumed that there is agreement on attribute names. It is assumed that the attribute names of values that are sent as part of a notify message from the coordinator are the same as what is expected in the diagnosis policies. This requires that policies entered are checked for consistency.

6 The policy stated in Example 5 is quite detailed in that it assumes that the administrator will specify the normalized value (specified using normalizedvalue) is to be used to determine if there are enough CPU resources available. We believe that we can simplify this by assuming that this is part of the mapping of the policy to rules.

7 We find the specification of expectation and administrative policies relatively easy. Diagnostic policies are more difficult and require a good understanding of the applications and how they behave. The administrator must specify a good deal of information. There is a good deal of work on diagnostics (e.g., [10, 9, 7, 8]) that use a variety of techniques. There exists efficient software to support these techniques. We believe that the diagnostics process should be able to take advantage of these techniques and tools. This will likely require changes in the way diagnostic policies are specified. Potentially, this could reduce the number of rules needed by the rule engine.

8 JESS is a rule-based engine. Initially, we found that JESS had a lot of overhead and had an impact on the application processes on the executing on the same machine. However, this was optimized and we no longer have this problem. It seems to be fast enough, but we do not have experience with an environment with hundreds of applications and potentially thousands of policies. However, in this case there would likely be more PDPs. This would distribute the load and thus minimize the number of policies at a specific PDP.

5. Related Work

Our work involved the use of policy formalisms and the development an architecture for distribution and enforcement of policies. Relatively little of this work examined all of these issues in an integrated framework that addresses end-to-end QoS management. Our work does this. In this section, we describe the related work done in policy specifications and architectures. We describe how we differ and how our work relates.

Policy Specification.

IETF is currently developing a set of standards (current drafts found in [14, 19, 22]) that includes an information model to be used for specifying policies, a standard that extends the previous standard for specifying policies for specifying QoS policies and a standard for mapping the information model to LDAP schemas. Informally, speaking an IETF policy basically consists of a set of conditions and a set of actions. The standards focus on low-level details related to policy encoding, storage and retrieval. We have experience in using the IETF standards. We did examine the use of IETF rules for specifying policies. IETF rules are reasonable for expectation policies, but we found that it was much more difficult to specify the administrative and diagnostic policies. Generally, we found easier to use. It is possible to specify policies using Ponder and translate to the IETF format.

Other language standards by IETF, as well as other network policy languages (summarized in [21]), include RPSL for describing routing policy constraints, PAX for defining pattern matching criteria in policy-based networking devices and SRL for creating rule sets for real-time traffic flow measurement. These languages primarily focus on policies applied to a network device. None of this work addresses the use of policies applied to applications.

Other policy specification languages [2, 6, 5, 15] focus on features that enable the specification of security related policies. The Ponder Policy Specification language [3] has a broader scope than most of the other languages in that it not only was designed with specifying security as the primary objective, but also general management policy. Ponder allows the administrator to use declarative statements and is independent of underlying systems. This is the primary reason why we have chosen to use Ponder to describe a high-level specification representation of policies.

Policy languages vary in the level of abstraction. Some languages focus on the specification of policies that describe what is wanted, while others focus on how to achieve what is wanted. This has resulted in different levels of policies, which has led to several attempts at a policy classification. This includes a policy hierarchy and a formal definition of policies defined by [24]. Our work defines two levels and would seem to correspond to the bottom two levels of the hierarchy in [24].

Architectures, Policy Distribution and Enforcement.

There has been some recent work in service level management [17, 16] that describes an architecture and policies for management of a differentiated services network so that users receive good quality of service and fulfill the service level agreements. This work does not make use of administrative or diagnostic policies.

There has been quite a bit of work (e.g., [1, 18, 23, 12]) that has looked at the translation of policies to network device configurations which are then sent to the PEPs so that the PEPs can configure the network devices. The focus is on network devices. In most of this work, policy distribution is initiated by an administrator and focusses on one device. Our work differs in that it focusses on applications and it is the initiation of an application session that causes the distribution of polices to management components.

A deployment model was developed by [4] for Ponder. Each policy type is compiled into a policy class by the Ponder compiler. The instantiation of a policy class is a policy object. Each policy object has methods that allow a policy object to be loaded to an *enforcement* agent and unloaded from an agent. Additional methods include enable and disabling of objects. Agents register with the event service to receive relevant events (as specified by the policies) generated from the managed objects of the system. There is no discussion on configuration of devices or applications. The roles of PDP and PEP is similar to that of an enforcement agent. We decided though that use of JESS would make certain tasks easier. For example, although Ponder assumes that the policies are independent in the sense that the order they are presented in does not matter. We assumed that order of policies mattered when translating to JESS rules. This meant that not every single JESS rule that is interested in a specific event has to be executed. This is important for the following reason: Example 3 described a policy called fps_low1 that is triggered when the event fps_low is generated. fps_low2 is also to be triggered when the event fps_low is generated. The difference is that that the action to be taken depends on different conditions of the state. If the conditional statement in the when clause is true in Example 3 this implies that the fault has been found and there is no reason to look at fps_low2. JESS's implementation allows for this. In the Ponder deployment model, all policy objects that register for a specific event will receive notification that the event has occurred and conditions are checked.

Generally, we found that very little work focusses on applying policies at the application level. The difficulty with the application level is that different policies will be needed for different sessions of the same application.

6. Conclusions

The initial deployment of the prototype has proven to be successful. The experimental results are similar to that reported in [13, 11]. The prototype allows a user to specify Ponder policies from a console and distribute them to the distributed components of the management system successfully. This work is one of the few that addresses the use of policies to application QoS management.

Future work includes addressing some of the issues brought up earlier in the Discussion section as well as the following:

1 We chose to specify our framework based on the IETF policy-based management framework. We are currently working on integrating network resource management.

2 We would like to do more work on ensuring that policy specifications are consistent. This requires many changes to the mapping approach used.

3 We are examining an environment with a mix of applications such as soft IP phones and web servers.

References

[1] M. Brunner and J. Quittek. MPLS Management using Policies. *Proceedings of the 7th IEEE/IFIP Symposium on Integrated Network Management (IM'01)*, Seattle USA, May 2001.

[2] F. Chen and R. Sandhu. Constraints for Role-Based Access Control. *Proceedings of 1st ACM/NIST Role Based Access Control Workshop*, Gaithersburg, Maryland, USA, ACM Press, 1995.

[3] N. Damianou, N. Dalay, E. Lupu, and M. Sloman. Ponder: A Language for Specifying Security and Management Policies for Distributed Systems: The Language Specification (Version 2.1). Technical Report Imperial College Research Report DOC 2000/01, Imperial College of Science, Technology and Medicine, London, England, April 2000.

[4] N Dulay, E. Lupu, M. Sloman, and N. Damianou. A Policy Deployment Model for the Ponder Language. *Proceedings of the 7th IEEE/IFIP Symposium on Integrated Network Management (IM'01)*, Seattle USA, May 2001.

[5] J. Hoagland, R. Pandey, and K. Levitt. Security Policy Specification Using a Graphical Approach. Technical Report Technical Report CSE-98-3, UC Davis Computer Science Department, UC Davis Computer Science Department, July 22, 1998 2002.

[6] S. Jajodia, P. Samarati, and V. Subrahmanian. A Logical Language for Expressing Authorisations. *Proceedings of the IEEE Symposium on Security and Privacy*, 1997.

[7] S. Katker. A Modelling Framework for Integrated Distributed Systems Fault Management. In *Proceedings IFIP/IEEE International Conference on Distributed Platforms*, pages 186–198, 1996.

[8] S. Katker and H Geihs. A Generic Model for Fault Isolation in Integrated Management Systems. *Journal of Network and Systems Management*, 1997.

[9] S. Katker and M. Paterok. Fault isolation and event correlation for integrated fault management. In *Proceedings of the 5th International Sypmposium on Integrated Network Management*, 1997.

[10] S. Kliger, S. Yemini, Y. Yemini, D. Ohsie, and S. Stolfo. A Coding Approach to Event Correlation. In *Proceedings of the 4th International Sypmposium on Integrated Network Management*, 1995.

[11] H. Lutfiyya, G. Molenkamp, M. Katchabaw, and M. Bauer. Issues in Managing Soft QoS Requirements in Distributed Systems Using a Policy-Based Framework. *Proceedings of the Policy 2001 Workshop:International Workshop on Policies for Distributed Systems and Networks, Bristol, UK, Springer-Verlag LNCS*, pages 185–201, January 2001.

[12] P. Martinez, M. Brunner, J. Quittek, F. Strauss, J. Schoenwaelder, S. Mertens, and T. Klie. Using the Script MIB for Policy-Based Configuration Management. *Proceedings of the IFIP/IEEE Symposium on Network Operations and Management Symposium (NOMS'02), Florence, Italy, 2002*, pages 461–468, January 2001.

[13] G. Molenkamp, H. Lutfiyya, M. Katchabaw, and M. Bauer. Resource Management to Support Application-Specific Quality of Service. *IEEE/IFIP Management of Multimedia Networks and Services (MMNS2001)*, October 2001.

[14] B. Moore, J. Strassmer, and E. Elleson. Policy Core Information Model – Version 1 Specification. Technical report, IETF, May 2000.

[15] R. Ortalo. A Flexible Method for Information System Security Policy Specificatio". *Proceedings of 5th European Symposium on Research in Computer Security (ESORICS 98)*, Louvain-laNeuve, Belgium, Springer-Verlag, 1998.

[16] P. Pereira, D. Dadok, and P. Pinto. Service Level Management of Differentiated Services Networks with Active Policies. *3rd Conferencia de Telecomunicacoes.*, Rio de Janeiro, Brazil, December 1999.

[17] P. Pereira and P. Pinto. Algorithms and Contracts for Network and Systems Management. *Proceedings of the 1st IEEE Latin American Network Operations and Management Symposium (LANOMS99)*, Rio de Janeiro, Brazil, December 1999.

[18] Alberto Gonzalez Prieto and Marcus Brunner. SLS to DiffServ Configuration Mappings. *Proceedings of the 12th International Workshop on Distributed Systems: Operations and Management DSOM'2001*, Nancy France, October 2001.

[19] Y. Snir, Y. Ramberg, J. Strassner, and R. Cohen. Policy Framework QoS Information Model. Technical report, IETF, April 2000.

[20] Startdust.com. Introduction to QoS Policies. Technical report, Stardust.com, Inc., July 1999.

[21] G. Stone, B. Lundy, and G. Xie. Network Policy Languages: A Survey and New Approaches. *IEEE Network*, 15(1):10–21, January 2001.

[22] J. Strassner, E. Ellesson, B. Moore, and Ryan Moats. Policy Framework LDAP Core Schema. Technical report, November 1999.

[23] P. Trimintzios, I. Andrikopoulos, G. Pavlou, and C. Cavalcanti. An Architectural Framework for Providing QoS in IP Differentiated Services Networks. *Proceedings of the 7th IEEE/IFIP Symposium on Integrated Network Management (IM'01)*, Seattle USA, May 2001.

[24] R. Wies. Polcies in Network and Systems Management – Formal Definition and Architecture. *Journal of Network and Systems Management*, 2(1):63–83, 1994.

[25] D. Yau and S. Lam. Adaptive Rate-Controlled Scheduling for Multimedia App lications. *Proceedings of the 1996 ACM Multimedia Conference*, Boston, Massachusetts, November 1996.

RESOURCE ACCESS MANAGEMENT FOR A UTILITY HOSTING ENTERPRISE APPLICATIONS

J. Rolia, X. Zhu, and M. Arlitt
Hewlett-Packard Laboratories
Palo-Alto, CA, USA, 94304
jerry.rolia@hp.com, xiaoyun.zhu@hp.com, martin.arlitt@hp.com

Abstract: In this paper we introduce a Resource Access Management (RAM) framework for resource utilities that facilitates Class of Service (CoS) based automated resource management. The framework may be used to offer resources on demand to enterprise applications that have time varying resource needs. The classes of service include guaranteed, predictable best effort, and best effort. The analytical apparatus we exploit requires the notion of application demand profiles that specify each application's resource requirements. These profiles may be statistical in nature. Consequently a policing mechanism is introduced to constrain each application's resource usage within its profile. A case study that exploits data from 48 data center servers, is used to demonstrate the framework. We show that our techniques are effective in: exploiting statistical multiplexing while providing service level assurances, limiting application demands in the presence of hostile application behaviour, and providing for differentiated service levels as planned.

Keywords: Resource management, Grid computing, Utility computing, Enterprise applications

1. Introduction

Enterprise applications, such as enterprise resource management, customer relationship management, and store fronts, can benefit from utility computing in the same way they currently benefit from storage and server consolidation exercises. Consolidation helps to decrease costs of ownership and increase return on investment. Resource utilities aim to provide such benefits on a large scale by automating management processes and increasing the effectiveness of consolidation. In this paper, we introduce a resource access management (RAM) framework for resource utilities that support enterprise applications.

Utility and Grid computing offer an appropriate starting point for automating consolidation processes for enterprise environments. Our approach relies upon: *utility computing*, where complex infrastructure is provided to enterprise applications on demand [2] [4] [10] [14] [19]; and *Grid computing*, which offers open services that help to match resource demands with supply, and that implements protocols for reserving, acquiring, and releasing resources adaptively [6] [7].

With this view, we assume that infrastructure service providers, which include data centers that may be internal to medium and large enterprises, will offer information technology resources as a utility service. Customers with applications, for example a department within an enterprise, will anticipate or characterize the demand of their applications and then discover, via Grid services, which utility is best able to satisfy the demands. The application will then be deployed and acquire and release resources as needed.

This paper presents a resource access management (RAM) framework for resource utilities that support Grids for enterprise applications. The framework helps to increase the effectiveness of consolidation processes by balancing utility asset utilization with Quality of Service (QoS) for resource access by applications. Differentiated QoS allows an infrastructure provider to be more flexible regarding the business models and services they offer to their customers. The framework provides Class of Service (CoS) based admission control and resource allocation for applications and implements a policing mechanism that governs access to resources. The CoS include guaranteed, predictable best effort, and best effort. For predictable best effort, the utility provides access to resources with a statistical assurance level θ. A case study demonstrates features of the framework.

The remainder of the paper is organized as follows. Section 2 describes related work. Our RAM framework is introduced in Section 3. A case study illustrates the behaviour of the framework for a hypothetical data center in Section 4. Summary and concluding remarks are given in Section 5.

2. Related Work

Grid resource management systems, such as LSF [22], Condor [11], and Legion [12], provide appropriate scheduling support for batch style computing jobs. However, they do not address the needs of enterprise applications. Enterprise applications operate continuously, have the potential for large peak-to-mean ratios in resource demands, and may have variable numbers of users. Resource management systems that support enterprise applications can increase asset utilization by exploiting the notion of statistical multiplexing. However, service level assurances are essential; enterprise applications must have confidence they will have access to resources when needed.

MUSE [3] is an example of a utility that treats hosted Web sites as services. All services run concurrently on all servers in a cluster. A pricing/optimization model is used to determine the fraction of cpu resources allocated to each service on each server. The over-subscription of resources is dealt with via the pricing/optimization model. When resources are scarce, costs increase thereby limiting application demand. Commercial implementations of such goal driven technology are emerging [5][18].

Garg *et al.* describe an SLA framework for QoS provisioning and dynamic capacity allocation [8]. They describe a mechanism that links application QoS, SLAs, and pricing. It includes the notion of penalties for utilities that do not live up to their obligations and incentives for applications to release resources when they are not needed. We also assume that applications have an incentive to relinquish resources that are not needed.

Our demand management approach differs from the above in the following way. Instead of relying on a dynamic pricing model we use historical and/or anticipated load information to specify a statistical demand profile (SDP) for each application [16]. An SDP bounds an application's expected resource requirements with a time ordered sequence of probability mass functions (pmfs), with one pmf per resource type. This enables support for statistical multiplexing and corresponding statistical assurances regarding resource availability. Furthermore, we separate the resource demand specifications of an application from mechanisms that control that demand. Each application is solely responsible for delivering an appropriate quality of service to its customers. It must translate a quality of service to a quantity of resources required to achieve that level of service [21]. The utility is responsible for providing resources on demand with a particular level of assurance (i.e. a probability) to its applications. We believe this separation of concerns is practical for many kinds of enterprise applications.

Urgaonkar *et al.* also consider quality of service issues [20] regarding resource access, but they do not characterize statistical assurance for a utility. Also they do not take into account the time varying nature of demands.

Hellerstein *et al.* illustrate the use of ANOVA [9] models and second order auto-regressive models [17] to characterize an application's request behaviour. Auto-correlation analysis is used to identify cycles of behaviour, such as weekly cycles versus daily cycles. ANOVA models give an additive workload model for daily, weekly, and monthly cycles. The second order auto-regressive models provide for a detailed characterization of residual behaviour once such cycles are removed. Their approach is to anticipate metric threshold violations over short time scales so that some preventative action can be taken before the violation occurs. The policing approach we present in this paper differs. It specifies directly when a utility may throttle an application's demand. It is a credit-based system, similar to a leaky-bucket approach [13], but based on time varying historical application behaviour.

In [16] we model and explore the impact of correlations among application demands on estimates for the number of resources needed for resource pools and the accuracy of our statistical assurance. In the next section we build upon [16] to introduce a RAM framework, classes of service, and a policing mechanism for a resource utility that supports enterprise applications.

3. Resource Access Management

This section describes our framework for Resource Access Management (RAM). The framework includes three components: admission control, policing, and CoS. *Admission control* decides whether an application will be permitted to execute within a utility. *Policing* mechanisms govern application requests to acquire and release resources as they execute within the utility. The CoS provide differentiated service to the admitted applications.

The RAM framework relies on SDPs. An SDP describes the expected resource usage of an application over time. Together, the SDPs of many applications include information required by a utility to estimate the number of resources required to satisfy their aggregate requirements while taking into account statistical multiplexing. SDPs are used to implement admission control and are reviewed in Section 3.1. Policing and CoS are introduced in Sections 3.2 and 3.3, respectively. Policing relies on the notion of application entitlement profiles (EP). These are used by the utility to decide whether an application's per-slot requests for resources are consistent with its SDP. If a request is not consistent then the utility can reject the resource request. The RAM framework as a whole is described in Section 3.4.

3.1 Statistical Demand Profiles (SDPs)

SDPs [16] represent historical and/or anticipated resource requirements for applications. For each unique resource type used by an application, we model the corresponding quantity of resources required as a sequence of random variables, $\{\mathbf{X}_t,\ t = 1, ..., T\}$. Here each t indicates a particular time slot, and T is the total number of slots used in this profile. For example, if each t corresponds to a 60-minute time slot, and $T = 24$, then this profile represents resource requirements by hour of day.

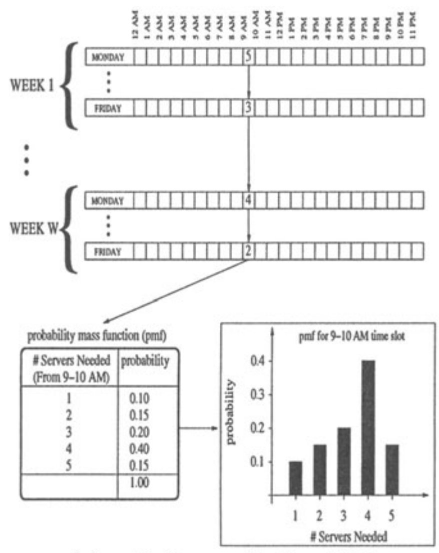
Our assumption here is that, for each fixed t, the behaviour of $\mathbf{X}_t$ is predictable statistically given a sufficiently large number of *observations* from historical data. This means we can use statistical inference to predict how frequently a particular quantity of resources may be needed. For each slot we use a probability mass function (pmf) to represent this information.

Figure 1(a) shows the construction of a pmf for a given time slot (9-10 am) for an SDP of an application. In the example only weekdays, not weekends, are considered. The application required between 1 and 5 servers over W weeks of observation. Since there are 5 observations per week, there are a total of 5 W observations contributing to each application pmf. Figure 1(b) illustrates how the pmfs of many applications contribute to a pmf for the utility as a whole for a specific time slot. As with applications, the SDP for the utility has one sequence of pmfs per resource type. The aggregate demand on a shared pool for a particular resource type is modeled as a sequence of random variables, denoted as $\{\mathbf{Y}_t,\ t = 1, \ldots, T\}$.

A complete description of SDPs, confidence intervals for probabilities in pmfs, correlations between application demands, and a mechanism for estimating the size of a resource pool needed to offer a particular level of service is offered in [16].

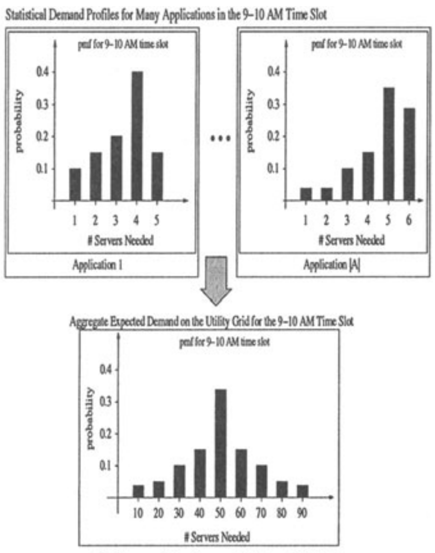
3.2 Policing and Entitlement Profiles (EPs)

SDPs provide a mechanism for characterizing expected resource requirements for each time slot t. They are appropriate for sizing the resource pools needed by the utility. However to provide the expected levels of statistical assurance, applications must behave according to their SDPs. Though a pmf of an SDP limits the maximum number of resources an application is entitled to for its corresponding time slot, we still require a method to ensure that each application adheres to its pmfs over time. The purpose of our *policing* mechanism is to specify an application's entitlement to resources over both short and longer time scales, to recognize when an application exceeds its entitlement,



# Servers Needed (From 9-10 AM)	probability
1	0.10
2	0.15
3	0.20
4	0.40
5	0.15
	1.00

(a) pmf of an application SDP



(b) pmf of a utility SDP

Figure 1. Creating statistical demand profiles

and to enable the framework to deal with surplus requests for resources in a systematic manner.

For short time scales, a *horizontal profile* characterizes the percentage of resource usage H_t^w that may occur over a sliding window of w slots ending at time slot t, for one or more values of w for all time slots. For example, H_t^w for time slot t = 9am to 10am and $w = 3$, describes the interval between 7am and 10am on the same day. If the maximum possible demand for the interval, based on the application's demand profile, is Δ units, then the horizontal entitlement H_t^w is a percentage of Δ, for example 80%.

For longer time scales, a *vertical profile* characterizes the percentage of resource usage V_t^w that may occur over a sliding window of w instances of the same time slot t, for one or more values of w, for all slots. For example, an application may be entitled to a total of Δ resource units over O instances of a 9am to 10am time slot. The vertical profile of V_t^w may specify that no more than 50% of the Δ units may be consumed in $w = O/3$ successive instances of a slot and no more than 100% of Δ units may be consumed in O successive instances of the slot. In this way the vertical component captures medium to long-term entitlements.

We define the H_t^w and V_t^w for an SDP's time slots for multiple values of w as an *entitlement profile* (EP). We use the EP to implement our *policing mechanism* that governs per-interval requests for resources. If an application requests more resources than it is entitled over any of the values for w then RAM can treat the requests according to some policy. For example it may reject the surplus requests without contributing to the utility's service level violations or treat them as best effort.

As with the construction of SDPs, the EP should be based on automated observation of the application's demand behaviour. They should be derived while constructing SDPs.

Lastly, it could be the case that an application acquires resources but holds them longer than expected thereby exceeding its next time slot's H_t^w or V_t^w for some w. In this case the application may be required to release some resources. We refer to this as *clawback*.

3.3 Classes of Service

This section describes classes of service for applications. Without loss of generality, in this paper we assume that an application acquires resources according to one CoS, and that its access to resources is constrained by its EP. In general, an application is expected to partition its requests across multiple classes of service to achieve the level of assurance it needs while minimizing its own costs.

We define the following CoS: *Guaranteed*, the application receives a 100 percent assurance it will receive the resources specified by its SDP; *Predictable Best Effort* (PBE) with probability θ, the utility offers resources to applications of this CoS with probability θ as defined in Section 3.1; and *Best Effort*, an application has access to these resources when they are available but must release these resources to the utility on demand.

For the PBE CoS, per-slot access to resources is governed by application EPs. For the Guaranteed CoS, an application's SDP need only contain the peak requirement for each time slot. It is always entitled to its per-slot peak requirement.

Consider a set of applications that exploit the utility. Using the techniques of Section 3.1, for the same applications, the utility will require a larger resource pool for a guaranteed CoS than for a predictable best effort CoS. Similarly, larger values of θ will require larger resource pools. In this way CoS has a direct impact on the cost of resources offered by the utility.

3.4 Resource Access Management Framework

This section describes the overall resource management framework. Figure 2 illustrates the admission control and policing process from the perspective of applications.

Figure 2(a) shows admission control and resource access steps for an application that aims to be hosted by the utility. The application presents its SDP and EP with a specific CoS. The utility performs an admission control test using the SDP to determine whether it has sufficient resources to accept the application while satisfying its obligations to existing applications. If accepted, the application may acquire resources, become deployed and begin its execution. Once in execution the application acquires and releases resources as needed but in accordance with its EP. As time progresses, each application's actual resource usage is characterized. For each time slot, for each sliding window, we compute each application's history of actual resource usage C. Actual resource usage is computed in the same manner as H_t^w and V_t^w but with recent measurements. C and its EP are presented for a policing test by the utility. An application is

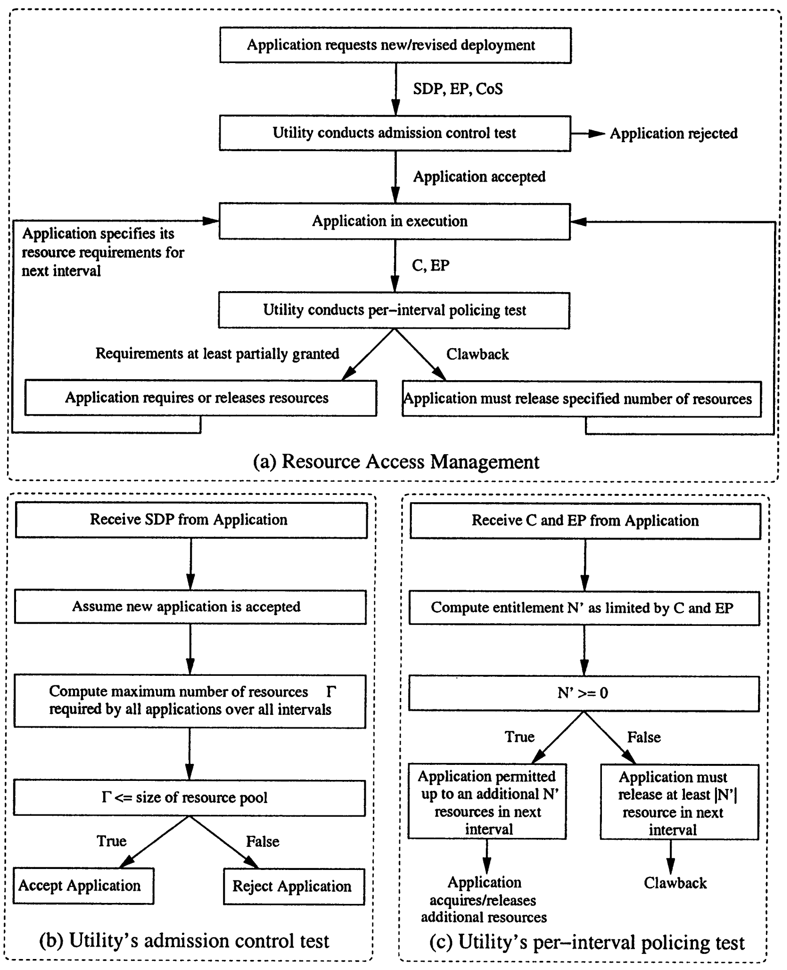

Application requests new/revised deployment
SDP, EP, CoS
Utility conducts admission control test
Application rejected
Application accepted
Application in execution
Application specifies its resource requirements for next interval
C, EP
Utility conducts per–interval policing test
Requirements at least partially granted
Clawback
Application requires or releases resources
Application must release specified number of resources
(a) Resource Access Management
Receive SDP from Application
Assume new application is accepted
Compute maximum number of resources Γ required by all applications over all intervals
Γ <= size of resource pool
True
False
Accept Application
Reject Application
(b) Utility's admission control test
Receive C and EP from Application
Compute entitlement N' as limited by C and EP
N' >= 0
True
False
Application permitted up to an additional N' resources in next interval
Application must release at least |N'| resource in next interval
Application acquires/releases additional resources
Clawback
(c) Utility's per–interval policing test

Figure 2. Resource access management processes

always permitted to release resources. The policing test decides the number of additional resources an application is entitled to or how many it must return (clawback). Even when permitted to request additional resources, depending on the application's CoS it may not actually receive all of the resources it requests. Receipt depends on resource availability, CoS based arbitration, and utility policy.

Figure 2(b) illustrates the admission control test. To begin, we assume that the application will be accepted. We unfold the SDP of the application onto the time slots of the utility's calendar, and then determine the total number of resources, Γ, needed to support all applications of all CoS over the future time slots. [16] explains how the pmfs of SDPs can be used to bound the potential impact of correlation in application demands on Γ when admitting a new application to a utility. Γ is then compared with the actual size of the resource pool to determine whether it is possible to accept the additional application. If so, then the changes to the utility's calendar are made permanent; otherwise, the request for admission is rejected and the application does not receive any resources. For the PBE CoS we use the techniques of [16], based on the SDPs of Section 3.1, to compute the number of resources required by each PBE CoS resource pool separately. For the Guaranteed CoS we need a pool large enough to satisfy peak application demands on an interval by interval basis. The number of resources needed by the utility, Γ, is chosen as the sum of the number of resources needed by each of its CoS separately.

Figure 2(c) illustrates the policing test. Given an application's history of resource usage C and its EP we compute the maximum number of additional resources N' to which the application is entitled. If N' is less than 0, the application must return $|N'|$ resources to the utility (clawback).

4. Case study

This section presents a case study that demonstrates how the RAM framework may be applied in practice to: quantify application demand requirements, explore the impact of policing, observe the impact of service differentiation, and illustrate resource availability. For this case study, we consider cpu utilization information from 48 servers in an enterprise data center. Sections 4.1 through 4.5 describe our hypothetical resource utility and demand characterization, the classes of service chosen for our study, our experimental design, and results.

4.1 Hypothetical Resource Utility

For the purpose of our study we were able to obtain cpu utilization information for a collection of 48 servers. The servers have between 2 and 8 cpus each, with the majority having either 4 or 6 cpus. The data was collected between September 2, 2001 and October 24, 2001. For each server, the average cpu utilization across all processors in the server was reported for each five minute measurement interval. The information was collected using MeasureWare (Openview Performance Agent) [1].

We *interpret* the load of each server as an application for a resource utility for enterprise applications. Whereas the groups' servers have multiple cpus in our study we assume the utility has many servers with one cpu each. If a server only required one cpu in an interval then its corresponding application requires one server in the utility. We exploit the fact that changes in server utilization reflect real changes in required activity for its applications. Our applications have the same changing behaviour. However since our purpose is simply to validate our techniques we are not concerned with whether it is feasible for the loads on the multi-cpu servers to be hosted on many single cpu servers. Similarly we do not scale the loads with respect to processor speed and/or memory capacity. Further details and results for our study are available in [15].

4.2 Statistical Demand Profiles and Entitlement Profiles

For this case study we chose to characterize the application SDPs by weekday and by hour of day. We consider all weekdays to be equivalent, and we omit data from weekends. As a result, our profile consists of 24 one hour time slots. Since we have utilization data for 35 days, there are $O = 35$ data points that contribute to each pmf in each application's SDP. We used the maximum of the utilizations observed at 5 minute intervals as the utilization value for a corresponding hour.

Values for H_t^w are based on observation of the 100-percentile of total servers used over a window size of $w = 4$, which contains four 60 minute time slots. The values for H_t^w are typically in the 70-90% range. For example, for a sliding window of $w = 4$ slots, the sum of the peak values for the corresponding pmfs may be δ whereas the observed maximum number of resources used for the window may have been only 0.7δ. Values for V_t^w are 100% of the total demand Δ as observed over the $w = 35$ observations used to characterize the SDP.

4.3 Classes of Service

For our case study we consider the following classes of service: guaranteed, predictable best effort with $\theta = 0.999$, denoted as PBE(0.999); and predictable best effort with $\theta = 0.99$, denoted as PBE(0.99).

We use a simulator that generates streams of resource requests according to the pmfs of application SDPs [16]. We submit applications to the utility for the three classes. Each class is an instance of the 48 applications. All applications are accepted. Their SDPs, along with their classes of service, determine the total number of resources for the simulated resource pool. We keep track of unused resources and assume these are offered as part of a best effort service to batch style jobs or other applications that can tolerate resource clawback. For each experiment, we simulate 1000 days of resource access.

4.4 Experimental design

For our experiments, we examine how effectively our policing mechanisms curtail bad behaviour. Policing determines which entitlement mechanisms are used. *Bad behaviour* is defined as the case where certain applications demand more servers than which they are entitled. Finally, in our experiments we exploit pool sharing. Each PBE class is given access to unused surplus servers of the other PBE class. The number of *surplus servers* for a CoS for a time slot t is defined as the difference between the total size of the server pool for that CoS (over all slots) and the number of servers needed to offer the assurance level θ for slot t. PBE(0.999) applications are given higher priority of access than PBE(0.99).

For bad behaviour, we consider the cases where all applications of the PBE(0.99) CoS run at the peak of their pmfs between time slots 10 and 18 (9am to 6pm). These are the slots with the greatest aggregate demand. Applications start their bad behaviour at random within the first 35 simulated days. Though this is not the worst possible behaviour, we believe it represents significant hostile behaviour with respect to applications in a resource utility.

For policing, we chose entitlement profiles as described in Section 4.2. In this study, when servers are clawed back an application must return them immediately.

In each case unused servers from the guaranteed and predictable best effort classes of service are made available to applications via a best effort CoS. Note that as a result the number of servers available for best effort service is time varying.

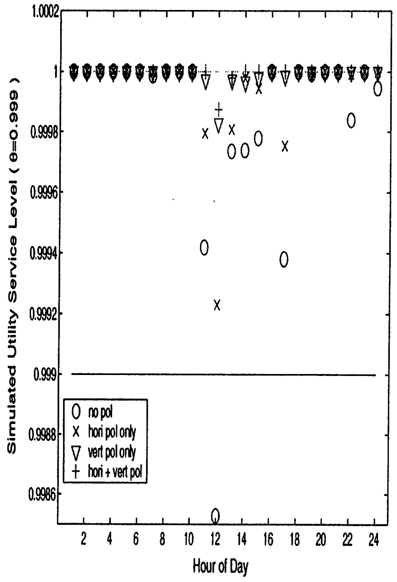
We illustrate and evaluate the RAM framework based on the the following metrics: utility service level violations for the predictable best effort classes (θ achieved for the utility), and, application service levels for the predictable best effort classes (θ perceived by each application). These metrics verify that the utility provides the levels of service that are planned, that applications receive qualities of service distinguished based on CoS, and that policing has the desired impact. We also show that that there remain significant opportunities for exploiting servers via a best effort CoS.

4.5 Results

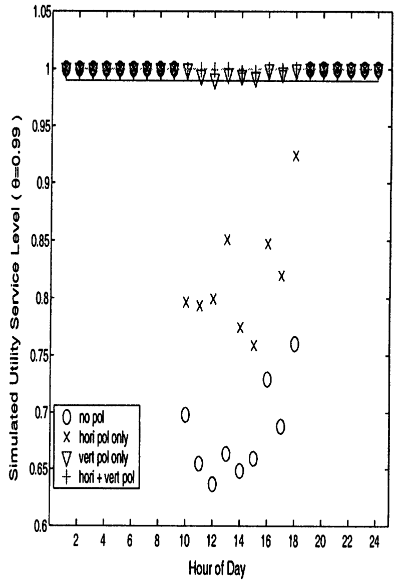
Figures 3(a) and (b) illustrate service levels achieved by the utility over the 1000 day time scale with applications of the PBE(0.99) CoS exhibiting bad behaviour between the 10th and 18th time slots. The impact of horizontal, vertical, and a combination of horizontal and vertical policing are shown. The horizontal line at θ represents the boundary between acceptable and unacceptable performance. Without policing, there are a large number of service level violations for the PBE(0.99) class. Vertical policing is effective as a mechanism to limit application demands. The combination of horizontal and vertical policing appears to further reduce service level violations. The PBE(0.999) is spared from the bad behaviour because its applications have higher priority access to resources.

Figure 4 provides Cumulative Distribution Functions (CDF) to illustrate service levels obtained by the individual applications. For the figure, x-axis labels have been chosen on a case by case basis to best illustrate density between θ and 1. The figures correspond to the scenarios of Figure 3. As in Figure 3, Figure 4(b) shows that vertical policing is an effective mechanism for ensuring application service levels. The combination of horizontal and vertical policing further reduces service level violations. Policy mechanisms for the utility may be used to ensure that an application does not receive a better quality of service than its entitled to even if resources are available.

Figure 5 provides CDFs for the number of servers available for the best effort CoS. These represent the unused servers from the Guaranteed and PBE classes of service. Figure 5(a) shows resource availability with pool sharing for the scenarios without bad behaviour. We note that over all time slots, there are between 150 and 170 servers that are unused – which is approximately 30% of the total resource pool. These can be used to support true best effort workloads, where the servers may be clawed back on demand, or they could be used by the utility to support unexpected demands by applications. Figure 5(b) shows

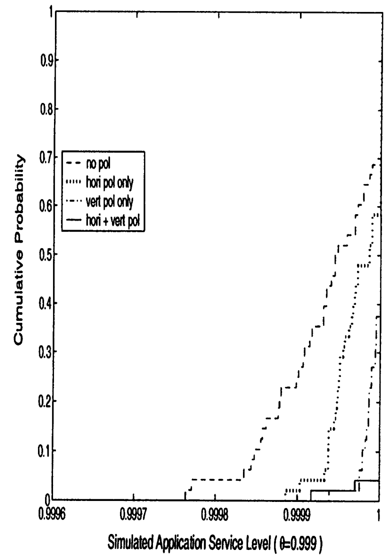


(a) $\theta = 0.999$, PBE(0.99) Bad behaviour

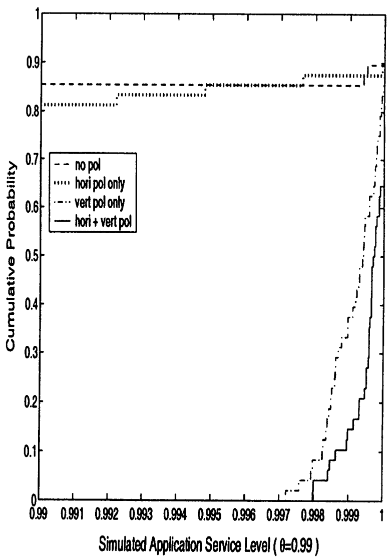


(b) $\theta = 0.99$, PBE(0.99) Bad behaviour

Figure 3. Impact of bad behaviour and policing on utility service levels

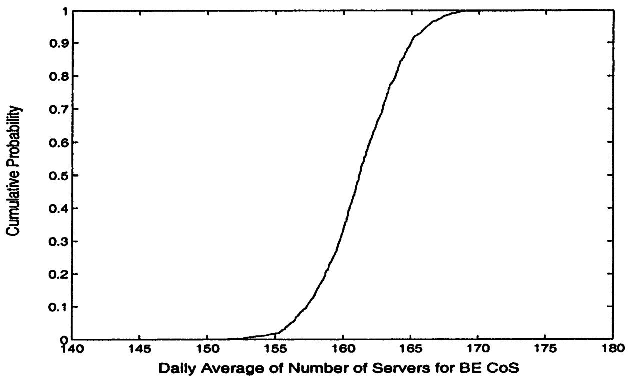


(a) $\theta = 0.999$, PBE(0.99) Bad behaviour

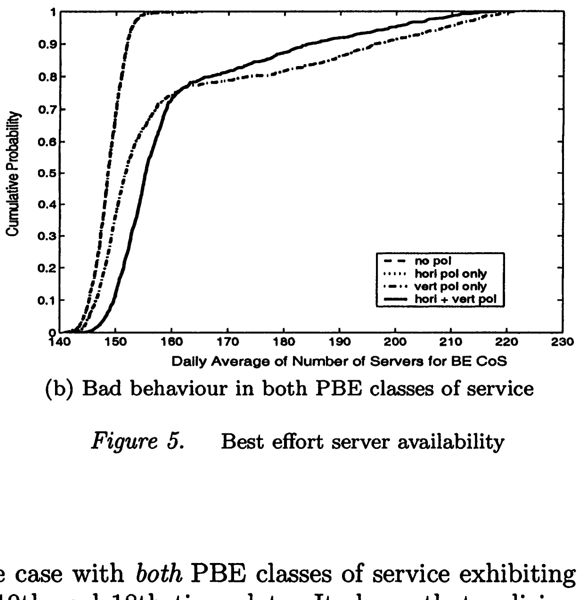


(b) $\theta = 0.99$, PBE(0.99) Bad behaviour

Figure 4. Impact of bad behaviour and policing on application service levels



(a) No bad behaviour



(b) Bad behaviour in both PBE classes of service

Figure 5. Best effort server availability

results for the case with *both* PBE classes of service exhibiting bad behaviour between the 10th and 18th time slots. It shows that policing limits demand thereby increasing the number of unused servers. From the figure, we note that the no policing and horizontal policing only scenarios overlap while the combination of vertical and horizontal policing is the most effective.

5. Summary

This paper presents a Resource Access Management (RAM) framework for enterprise applications that access resource utilities. The framework supports admission control, CoS, and policing. A case study used measurements from 48 servers in a data center to demonstrate the effectiveness of our approach for a hypothetical utility.

Based on our case study we find that the hypothetical utility is able to offer differentiated services including predictable best effort with multiple levels of assurance. Applications in these classes of service receive service in proportion to the service levels of their class. Our credit-based policing mechanism reduces service level violations in the presence of bad behaviour from applications. Policing improves the service levels yet maintains the property of service differentiation between the two predictable best effort classes. Finally, even with resource sharing and statistical assurances there remain significant opportunities for making resources available as part of a best effort service.

We find that statistical demand profiles and entitlement profiles are complementary in their support of RAM. The SDPs help to size resource pools. An EP specifies the time scales and manner in which an application must conform to its SDP. Correspondingly for a PBE CoS, a utility must also provide a time scale over which it provides its assurance level.

Last, we fully expect that from time to time applications will require more resources than expected. Our RAM framework provides the basis to decide when its possible to support such requests. Our future work includes exploring the use of multiple time scales with vertical policing, dealing with long term growth in demands (trending), exploiting EPs to look ahead at resource entitlements when considering resource allocation issues, supporting multiple resource types, exploring issues of time scale and statistical assurances, and providing quantitative support for models that take into account pricing and penalties.

References

[1] HP Openview Performance Agent. `www.openview.hp.com/products/performance`.

[2] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, and M. Kalantar. Oceano – SLA based management of a computing utility. In *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management*, May 2001.

[3] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle. Managing energy and server resources in hosting centers. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles (SOSP)*, October 2001.

[4] Think Dynamics. `www.thinkdynamics.com`.

[5] Ejasent. Utility computing white paper, November 2001. `www.ejasent.com`.

[6] I. Foster and C. Kesselman. The grid: Blueprint for a new computing infrastructure, July 1998. ISBN 1-55860-475-8.

[7] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration, www.globus.org, January 2002.

[8] R. Garg, R Singh Randhawa, H. Saran, and M. Singh. A sla framework for qos provisioning and dynamic capacity allocation. In *Proceedings of IWQoS 2002*, pages 129–137, Miami, USA, May 2002.

[9] J. Hellerstein, F. Zhang, and P. Shahabuddin. A statistical approach to predictive detection. *Computer Networks*, January 2000.

[10] Hewlett-Packard. HP utility data center architecture. `www.hp.com/solutions1/infrastructure/solutions/utilitydata/architecture`.

[11] M. Litzkow, M. Livny, and M. Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference on Distributed Computing Systems*, pages 104–111, June 1988.

[12] A. Natrajan, M. Humphrey, and A. Grimshaw. Grids: Harnessing geographically-separated resources in a multi-organisational context. In *Proceedings of High Performance Computing Systems*, June 2001.

[13] J. Rexford, F. Bonomi, A. Greenberg, and A. Wong. A scalable architecture for fair leaky-bucket shaping. In *IEEE INFOCOM (3)*, pages 1054–1062, 1997.

[14] J. Rolia, S. Singhal, and R. Friedrich. Adaptive Internet Data Centers. In *Proceedings of SSGRR'00*, L'Aquila, Italy, `www.ssgrr.it/en/ssgrr2000/papers/053.pdf`, July 2000.

[15] J. Rolia, X. Zhu, and M. Arlitt. Resource access management for a utility hosting enterprise applications. Technical Report HPL-2002-346, HP Labs, Palo Alto, CA, Dec 2002.

[16] J. Rolia, X. Zhu, M. Arlitt, and A. Andrzejak. Statistical service assurances for applications in utility grid environments. In *IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 247–256, Fort Worth, TX, October 2002.

[17] D. Shen and J. Hellerstein. Predictive models for proactive network management: Application to a production web server. In *IEEE NOMs*, 2000.

[18] Sychron. Sychron Enterprise Manager, 2001. `www.sychron.com`.

[19] TerraSpring. `www.terraspring.com`.

[20] B. Urgaonkar, P. Shenoy, and T. Roscoe. Overbooking and application profiling in shared hosteing platforms. In *Fifth Symposium on Operating Systems Design and Implementation (ODSI)*, Dec. 2002.

[21] D. Xu, K. Nahrstedt, and D. Wichadakul. Qos and contention-aware multi-resource reservation. *Cluster Computing*, 4(2):95–107, 2001.

[22] S. Zhou. Lsf: Load sharing in large-scale heterogeneous distributed systems. In *Workshop on Cluster Computing*, 1992.

SLA-DRIVEN MANAGEMENT OF DISTRIBUTED SYSTEMS USING THE COMMON INFORMATION MODEL

Markus Debusmann[1]*, Alexander Keller[2]
[1]*FH Wiesbaden - University of Applied Sciences, Department of Computer Science*
Kurt-Schumacher-Ring 18, 65197 Wiesbaden, Germany
m.debusmann@computer.org
[2]*IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, USA*
alexk@us.ibm.com

Abstract: We present a novel approach of using CIM for the SLA-driven management of distributed systems and discuss our implementation experiences. Supported by the growing acceptance of the Web Services Architecture, an emerging trend in application service delivery is to move away from tightly coupled systems towards structures of loosely coupled, dynamically bound systems to support both long and short term business relationships across different service provider boundaries. Such dynamic structures will only be successful if the obligations of different providers with respect to the quality of the offered services can be unambiguously specified and enforced by means of dynamic Service Level Agreements (SLAs). In other words, the management of SLAs needs to become as dynamic as the underlying infrastructure for which they are defined.

Our previous work has shown that Web Services, as a typical example for a service-oriented architecture, can be extended in a straightforward way for defining and monitoring SLAs. However, SLAs defined for a Web Services environment need to take into account the underlying managed resources whose management interfaces are defined based on traditional management architectures, such as SNMP-based management or the Common Information Model (CIM). As a solution to this problem, the approach presented in this paper addresses the integration problem of how to transform a Web Services SLA so that it can be understood and enforced by a service provider whose management system is based on a traditional management architecture, such as CIM.

Keywords: SLA, Web Services, Common Information Model, Inter-Domain Management

1. Introduction and Problem Statement

Over the last year, emerging component based service architectures built on top of Web Services [12], such as the *Open Grid Services Architecture (OGSA)* [7, 20], have been gaining increasing acceptance beyond computing-intense scientific and commercial applications: It appears highly likely that the next generation of e-Business systems will consist of an interconnection of services, each provided by a possibly different service provider, that are put together "on demand" to offer an end-to-end service to a customer. Such an environment – referred to as 'Computing Grid' [6] – will be administered and managed according to dynamically negotiated Service Level Agreements (SLA) between service providers and customers [13, 21]. Consequently, systems ma-

*Work done while author was an intern at IBM T.J. Watson Research Center.

nagement will increasingly become SLA-driven and needs to address challenges such as dynamically determining whether enough spare capacity is available to accomodate additional SLAs, the negotiation of SLA terms and conditions, the continuous monitoring of a multitude of agreed-upon SLA parameters and the troubleshooting of systems, based on their importance for achieving business objectives. A key prerequisite for meeting these goals is to understand the relationship between 'high-level' SLA parameters (e.g., Availability, Throughput, Response Time) and 'low-level' resource metrics, such as counters and gauges. However, mapping SLA parameters onto metrics that are retrieved from managed resources is a difficult problem [14].

This paper presents our approach for mapping SLAs, defined using the *Web Service Level Agreement (WSLA)* framework (described in section 2), which is based on the Web Services Architecture, onto the Common Information Model (CIM) [2]. Thus, the work described in this paper can be regarded as a precursor to future work on integrating emerging service architectures with traditional enterprise management frameworks. The novelty of our approach lies in the way we address the following key questions; these questions also reflect the structure of this paper:

1 **How can SLA parameters be mapped onto resource metrics?**
At the core of our approach to this problem is the WSLA language that allows a party involved in the establishment of an SLA to define what is actually meant by an SLA parameter. Instead of merely assigning thresholds to pre-defined SLA parameters whose semantics vary greatly [1], the WSLA framework (presented in section 2) allows the precise definition of how SLA parameters are supposed to be computed and aggregated from resource metrics.

2 **What SLA monitoring components ought to be implemented as Web Services? For which components is CIM the better answer?**
Based on an inter-domain SLA management scenario, section 2 breaks down the SLA monitoring process into a set of elementary services needed to enable the management of an SLA throughout the various phases of its lifecycle. Since we are dealing with a service architecture and a resource management architecture, every service may be implemented either as a Web Service or based on CIM. An analysis and an evaluation of the various options is given in section 3.

3 **Which parts of the SLA should be modeled in CIM and how does a suitable model look like?**
While this question is closely related to the previous one, there are a few additional implications a suitable CIM model for SLAs needs to take into account: In particular, the CIM model needs to provide a means for keeping data that relates to the definition aspects of the SLA while being able to measure and store the actual SLA paremeter and metric values at runtime. Stated differently, the measured values need to be tied back to their definitions and to the SLA in which they are defined. Our solution to this problem, based on the CIM Metrics Model, is described in the second part of section 3.

4 **How can one delegate management functions to an agent in a WBEM/CIM environment?**
Traditionally, the purpose of CIM subagents (termed "providers") is to make the instrumentation of managed resources accessible to a CIM Object Manager (CIMOM). Providers respond to incoming requests, retrieve the requested management information and return the results to a CIMOM. Thus, they play a passive role in the management process by reacting to requests coming from a CIM client.

Since an SLA is usually associated with a schedule that indicates precisely when and how often the measurements are supposed to be taken, a CIM provider needs to take an active role when carrying out its measurements. Our approach to this classical problem of (statically, in our case) delegating measurement functionality to agents [22], with a specific focus on SLAs and CIM, is described in section 4.

5 **Finally, how can one achieve Interoperability between the Web Services Architecture and CIM?**
This is obviously a very broad question, for which a generic approach is likely to be as complex as the well-known approaches for achieving interoperability between traditional management architectures (for an in-depth discussion of this subject, see [17, 19]). Nevertheless, we have designed and implemented a mechanism for deploying SLAs from a Web Services environment into a CIMOM and a way to deliver measurements from a CIMOM back to a Web Service. Our experiences with the proof-of-concept implementation are described in section 5. Our work can be regarded as a precursor to future work dealing with the development of generic mechanisms for integrating Web Services based management with existing management infrastructures, such as CIM.

2. Web Service Level Agreements (WSLA)

This section describes our work towards a flexible SLA monitoring framework, primarily targeted at Web Services, but applicable to any kind of managed resource in a distributed management environment as well. In order to address the requirements of facilitating the SLA definition and the automated configuration of an SLA monitoring infrastructure in inter-domain environments, we have specified and implemented the *Web Service Level Agreement (WSLA)* framework [10]. In [9], we have described the concepts behind WSLA. Although it is not the purpose of this paper to describe WSLA in detail, we need to provide a brief overview of WSLA and its principles to set the stage for our CIM based SLA model detailed in section 3 and the architecture of our solution described in section 4.

Our approach to enable SLA-driven Management of distributed and highly dynamic systems, WSLA, consists of a flexible and extensible language [15] based on the XML schema and a runtime architecture based on several SLA monitoring services, which may be outsourced, either in full or in parts, to third parties to ensure a maximum of accuracy [18]. WSLA enables service customers and providers to unambiguously define a wide variety of SLAs, specify the SLA parameters and the way how they are measured, and tie them to managed resource instrumentations. In addition, the various WSLA monitoring services, described below in further detail, can be configured automatically according to the terms and conditions specified in the SLA. A Java-based prototype implementation of the WSLA framework, termed *SLA Compliance Monitor*, is included in the current version 3.2 of the publicly available IBM Web Services Toolkit [8].

2.1 SLA Lifecycle in the WSLA Architecture

Figure 1 depicts the typical lifecycle of an SLA in a multi-provider environment. The lifecycle consists of the following straightforward phases: SLA creation, SLA deployment, SLA execution, and SLA termination. For the sake of brevity, the latter is not depicted in the figure.

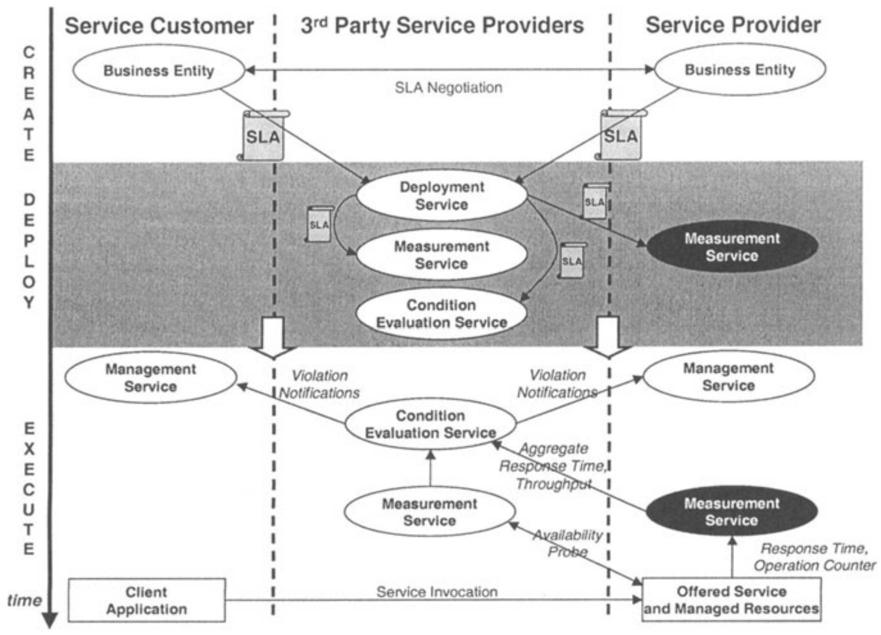


Figure 1. Lifecycle of a Service Level Agreement in a Multi-Provider Environment

The SLA creation process involves the negotiation and signing of an SLA by both a service provider and service customer. During this process, a customer retrieves the metrics offered by a provider, aggregates and combines them into various SLA parameters, defines service levels for every SLA parameter, and submits the SLA to the service provider for approval. On the side of every *signatory party* (a party that signs an SLA) a **Business Entity** carries out the negotiation: It embodies the business knowledge, goals and policies of a party. Such knowledge enables the Business Entity to decide which service levels should be specified in the SLA to ensure compliance with its business goals. A typical example for such a decision on the service customer side is to define thresholds for response times or throughput, depending on the price he is willing to pay. On the provider side, typical business actions are to decide if the SLA is acceptable as a whole or whether the customer-specified thresholds are too restrictive. Once agreement on the main elements of the SLA is reached, customer and provider may define third parties (which we call *supporting parties* in the WSLA context), to which SLA monitoring tasks may be delegated. Supporting parties come into play when either a function needs to be carried out that neither service provider nor customer wants to do, or if these signatory parties do not trust their counterparts to perform a function correctly. Keynote Systems, Inc. [11] and Matrix NetSystems, Inc. [16] are real-life examples of such third-party monitoring service providers.

Once the SLA is finalized, both service provider and service customer make the SLA document available for deployment. The **Deployment Service** is responsible for checking the validity of the SLA and distributing it either in full or in appropriate parts to the supporting parties (gray shaded area in the middle of Figure 1). The latter is needed to ensure that a supporting service receives only the amount of information it needs to carry out its tasks.

In our scenario, we assume that a part of the SLA monitoring and supervision activities is delegated to third party service providers. Their runtime interactions are depicted in the lower part of Figure 1. Typical services that may be outsourced to third parties fall into two categories:

Measurement Service: The Measurement Service maintains information on the current system configuration, and runtime information on the metrics that are part of the SLA. It measures SLA parameters such as availability or response time either from inside, by retrieving raw metrics directly from managed resources, or outside the service provider's domain, e.g., by probing or intercepting client invocations. A Measurement Service may measure all or a subset of the SLA parameters. Multiple Measurement Services may simultaneously measure the same metrics, e.g., a Measurement Service may be located within the provider's domain while another Measurement Service probes the service offered by the provider across the Internet from various locations. For our discussion, we call metrics that are retrieved directly from managed resources **Raw Metrics**. **Composite Metrics**, in contrast, are created by aggregating several raw (or other composite) metrics according to a specific algorithm, such as averaging one or more metrics over a specific amount of time or by breaking them down according to specific criteria (e.g., top 5%, minimum, maximum, mean, median etc.). This is usually being done by a Measurement Service within a service provider's domain (depicted in Figure 1 as the oval having a black background), but can be outsourced to a third-party Measurement Service as well (Measurement Service with white background). In sections 4 and 5, we will describe our approach to designing and implementing an internal Measurement Service (black oval in the Figure) in CIM and how it accesses managed resource instrumentation.

Condition Evaluation Service: This service is responsible for monitoring compliance of the SLA parameters at runtime with the agreed-upon Service Level Objective (SLO) by comparing measured parameters against the thresholds defined in the SLA and notifying the Management Services of the service customer and provider. It obtains measured values of SLA parameters from the Measurement Service(s) and tests them against the guarantees given in the SLA. This can be done each time a new value is available, or periodically.

Finally, both service customer and provider have a **Management Service**: Upon receipt of a notification, the Management Service (usually implemented as part of a traditional management platform) will take appropriate actions to correct the problem, as specified in the SLA. The main purpose of the Management Service is to execute corrective actions on behalf of the managed environment if a Condition Evaluation Service discovers that a term of an SLA has been violated.

2.2 Expressing SLAs in the WSLA Language

In this section, we provide a brief overview over the parts of the WSLA language that relate to the definition of SLA parameters and the way they are monitored. For a detailed discussion of the entire WSLA language, the reader is referred to [10].

The **Parties** section identifies the contractual parties and contains the technical properties of a party, i.e., their addresses and interface definitions (e.g., the ports for receiving notifications).

The **Service Description** section of the SLA specifies the characteristics of the service and its observable parameters as follows: For every **Service Operation**, one or more **Bindings**, i.e., the transport encoding for the messages to be exchanged, may be specified. In addition, one or more **SLA Parameters** of the service may be specified. Examples of such SLA parameters are *service availability*, *throughput*, or *response time*. Every SLA parameter refers to one **Metric**, which, in turn, may aggregate one or more other (composite or raw) metrics, according to a measurement directive or a function. Examples of composite metrics are *maximum response time of a service*, *average availability of a service*, or *minimum throughput of a service*. Examples of raw metrics are: *system uptime*, *service outage period*, *number of service invocations*. **Measurement Directives** specify how an individual raw metric can be obtained from a managed resource or from a system acting as a proxy for the resource. Typical examples of measurement directives are the uniform resource identifier of a hosted computer program, a protocol message (e.g., an SNMP GET message), the command for invoking scripts or compiled programs, or a query statement issued against a database or data warehouse. **Functions** are the measurement algorithm that specifies how a composite metric is computed. Examples of functions are formulas of arbitrary length containing mean, median, sum, minimum, maximum, and various other arithmetic operators, or time series constructors. For every function, a **Schedule** is specified. It defines the time intervals during which the functions are executed to retrieve and compute the metrics. These time intervals are specified by means of *start time*, *duration*, and *frequency*. Examples of the latter are *weekly*, *daily*, *hourly*, or *every minute*.

Obligations, the last section of an SLA, define the SLOs, guarantees and constraints that may be imposed on the SLA parameters. For the sake of brevity, we have omitted their description here; further details and usage examples can be found in [10].

3. Integrating the WSLA and CIM Environments

Considering an SLA management environment as shown in Figure 1 raises the question how the five services can be integrated in the most efficient way. Today, the Business Entity is usally a human being. The Management Service represents the management platform run by the service provider and customer. Thus, the key question for integrating WSLA and CIM is: Which of the remaining services (Deployment, Measurement, Condition Evalution) are best implemented as a Web Service and which services ought to be implemented with CIM? Three alternatives can be considered:

1 The first approach is to implement the services entirely in a Web Services environment, as demonstrated by the WSLA Compliance Monitor of the Web Services Toolkit [8]. This obviously simplifies the delegation of services to third party providers; however, the integration with a management platform and today's managed resources is challenging as none of them have a management interface based on Web Services. Therefore, a pure Web Services based solution is highly unlikely.

2 Implementing all services on a CIM basis is the other extreme. This simplifies the integration with managed resources. However, if certain tasks are to be delegated to third party providers, this solution makes assumptions about their management infrastructure and thus limits the flexibility of the overall SLA management system.

3 For maximum flexibility, it is crucial to find the right balance between those two extremes. In our solution, we chose the Measurement Service to be CIM based (depicted as a black oval in Figure 1), which facilitates the ties between high-level SLA parameters and low-level resource metrics as well as the integration with managed

resources. Making the Condition Evaluation Service a Web Service yields the flexibility of delegating its tasks to third party providers. The Deployment Service is the gateway between both worlds by having a Web Services interface. Its backend acts as a CIM Client and is thus able to communicate with the CIM based Measurement Service for setting up the measurements defined in the SLA.

3.1 Representing SLA Definitions in CIM

Obviously, the straightforward way of implementing an SLA model in CIM is to reuse existing classes of the CIM Core and Common Schemas. However, CIM does not yet provide explicit classes for defining SLAs and SLOs. The CIM Policy Schema consists of various classes representing policies (`CIM_Policy`), which aggregate conditions (`CIM_PolicyCondition`) and actions (`CIM_PolicyAction`). Consequently, these classes could serve as a basis for defining the CIM equivalent of WSLA Obligations. However, obligations are evaluated by the Condition Evaluation Service, which we decided to keep outside the scope of our CIM implementation. Furthermore, the CIM Policy Schema does not provide an explicit representation of conditions and actions, which is the case of WSLA. On the other hand, we were able to reuse the class `CIM_PolicyTimePeriodCondition`, which maps to the WSLA Schedule concept and therefore serves as base class of `IBM_Schedule`. It should be noted that recent work in the Policy Working Group aims at extending the Policy Schema by a mechanism to express SLAs in CIM.

Information relating to the definition of SLA parameters and their associated metrics is found in another CIM Common Schema: The CIM Metrics Schema defines, among other, two classes that allow the representation of metric definitions (`CIM_BaseMetricDefinition`) and – once a metric instance is created – its value (`CIM_BaseMetricValue`). This matches the WSLA philosophy very well, since the metric definitions (and the way they are computed) are part of an SLA and therefore – in a first step – deployed to the Measurement Service in charge of computing them. Once the monitoring process starts, a Measurement Service obtains and computes values for these metric definitions and thus creates new instances of the class `CIM_BaseMetricValue` (or more specific subclasses). We were therefore able to fully take advantage of the CIM Metrics Model. Distinguishing between metric definitions and values has the following advantages:

1 keeping both definitions and values together and thus linking information from the deployment and runtime stages,

2 leveraging the power of CIM queries for SLA retrieval, e.g., retrieve all SLA parameters for a given SLA,

3 enabling the service provider to develop a collection of common-off-the-shelf metric definitions that can be reused for different customers.

Figure 2 depicts the CIM SLA model, which is equivalent to WSLA in terms of its expressiveness. As a result of the decision to implement the Measurement Service in CIM, the model reflects only those parts of an SLA that define the relationships between SLA parameters and metrics, i.e., the aggregation functions and the schedule for their retrieval. The central element of the model is the `IBM_SLA` class, which ties together all the other elements comprising the SLA. It is derived from `CIM_ManagedElement`. Several instances of `IBM_SLA` can exist in parallel, thereby representing SLAs of one individual or several different customers.

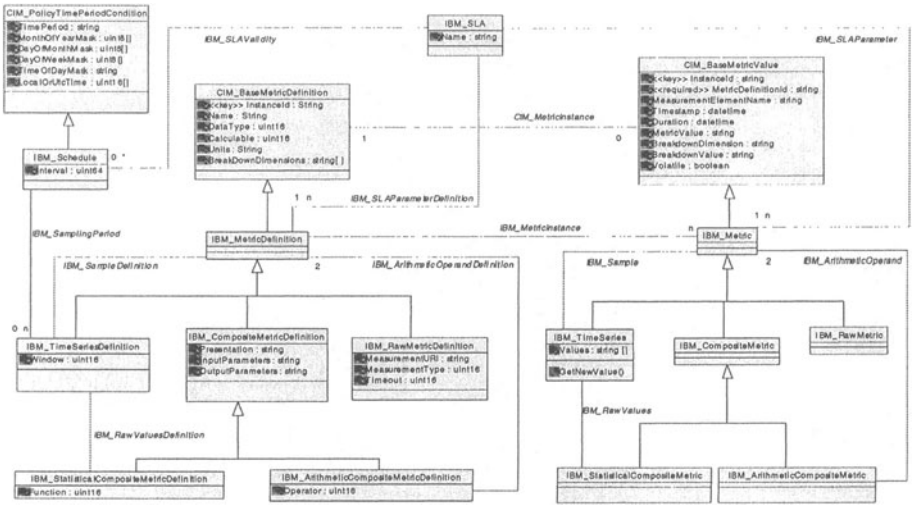


Figure 2. CIM Model of Service Level Agreement definitions and values

IBM_MetricDefinition is the base class of the various SLA metric definitions; it inherits from the CIM_BaseMetricDefinition class and is refined as follows: As discussed in section 2.1, WSLA distinguishes between raw metrics, composite metrics, and time series. Consequently, the model comprises three corresponding classes IBM_RawMetricDefinition, IBM_CompositeMetricDefinition and IBM_TimeSeries-Definition that hold the definitions for these metric types. Composite metric definitions are further subclassed because they represent complex metrics that are computed by the Measurement Service. The IBM_ArithmeticCompositeMetricDefinition class represents the arithmetic operator (e.g., +,-,*,/) which aggregates two IBM Metric-Definitions by following the association IBM_ArithmeticOperandDefinition. The IBM_StatisticalCompositeMetricDefinition class captures the definitions of statistical functions that apply to times series. The latter are stored using the IBM_Time-SeriesDefinition class, which holds metric values sampled in regular intervals, e.g., according to an IBM_Schedule. IBM_TimeSeries instances may be used as input for any number of statistical composite metrics. This reduces redundancy and ensures the integrity of measurement data by providing a shared basis for statistical calculations. The intervals during which metrics are collected and placed into a time series (cf. the association IBM_SamplingPeriod) are represented by the class IBM_Schedule, which extends the class CIM_PolicyTimePeriodCondition by a property 'Interval'.

The three metric types, along with the function definitions and their schedules allow the definition of arbitrarily complex SLA parameters, such as the average utilization of network interfaces or the maximum reponse time within the last hour. Note that all the classes discussed until now – depicted in the left part of Figure 2 – are used to represent the definitions within an SLA — and not the actual measurement values. They are instantiated whenever a new SLA is deployed to the CIM based Measurement Service. Once the definition classes are instantiated, the Measurement Service uses these definitions to retrieve and compute the actual values.

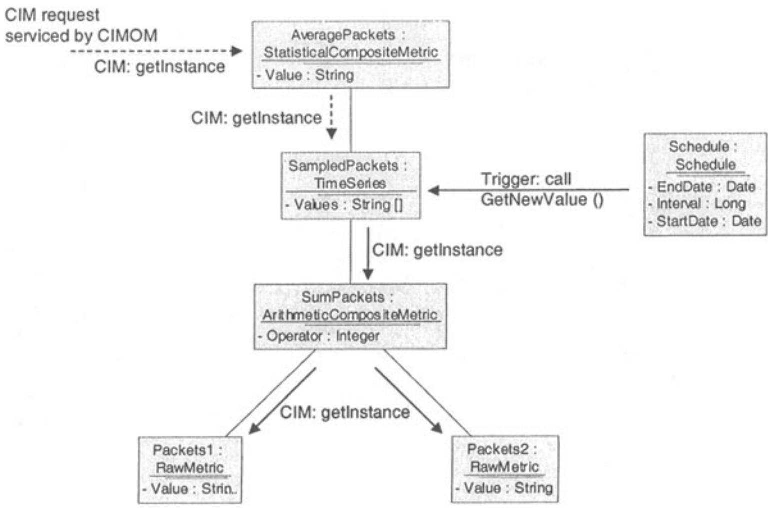


Figure 3. Aggregating the 3 metric types: raw metrics, composite metrics and time series

3.2 Computation and Aggregation of Metric Values

The computation of SLA parameters requires the automatic retrieval of metric values by the Measurement Service. During runtime, instances of `IBM_Schedule` are used to trigger the retrieval of current metric values and to perform metric computation and aggregation. The input metrics and the (intermediate or final) results are represented by the classes we will discuss below; they are depicted in the right half of Figure 2.

The runtime relationships between metric value instances comprising a simple SLA are shown in Figure 3. There are two separate activation mechanisms for calculating the metrics: timer-triggered (solid arrows) and request-triggered (dashed arrows). In regular time intervals, an `IBM_Schedule` instance initiates the collection of a new metric value for an `IBM_TimeSeries` object by invoking its `GetNewValue()` CIM method. This causes the collection of the `IBM_ArithmeticCompositeMetric` associated with the `IBM_TimeSeries`, which is done by means of the CIM operation `getInstance`, defined in [4]. Before the `IBM_ArithmeticCompositeMetric` instance can be calculated, its associated `IBM_RawMetric`s have to be retrieved. After the calculation is done the result is given back and finally stored within the `IBM_TimeSeries` object.

The second possible activation mechanism is a CIM request from a CIM client. In our example, a request for the `IBM_StatisticalCompositeMetric` is handled, thus the associated `IBM_TimeSeries` instance has to be retrieved. After that, the average value is calculated based on the values retrieved from the `IBM_TimeSeries` object.

4. CIM based SLA Measurement Service

Figure 4 depicts the architecture of our CIM based SLA Measurement Service. Since the CIMOM is the central component responsible for realizing the Measurement Service, the SLA structure has to be mapped first onto a CIM model, as described in section 3.1. This model has to be loaded once into the CIMOM (1) and appropriate

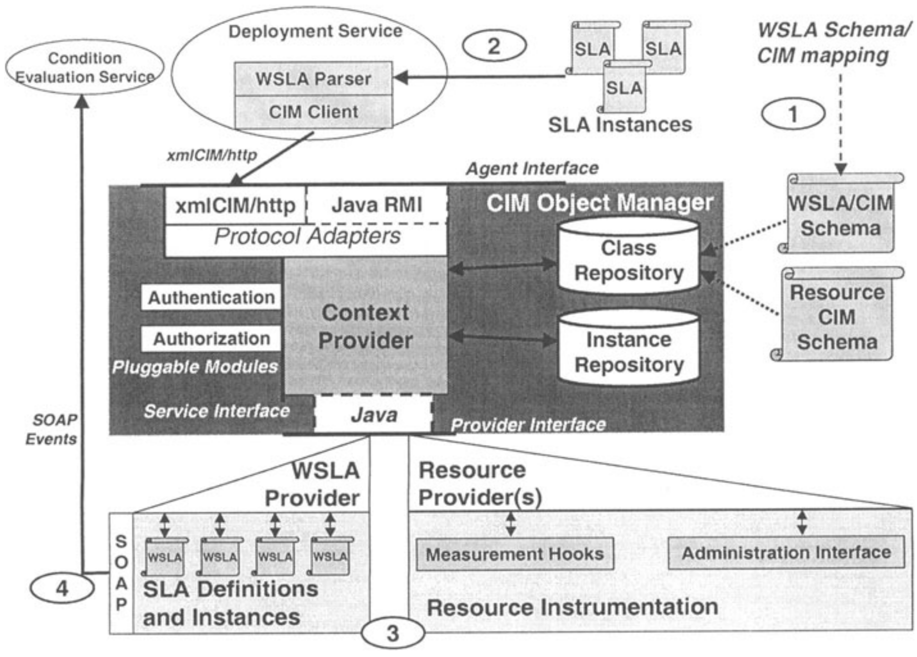


Figure 4. Architecture of the CIM based SLA Measurement Service

providers implementing this information model have to be developed. Since all the SLAs share a common structure, this CIM SLA model is stored in the class repository.

After this WSLA/CIM schema is loaded, SLAs of different customers can be deployed (2). In our implementation, the deployment of SLAs is realized as a custom-based solution that fits the WSLA environment (cf. section 3). Therefore, the Deployment Service offers a Web Services interface for receiving new WSLA documents, i.e., agreed-upon SLAs signed by the signatory parties. After receiving an SLA, the Web Service backend uses an XML parser to analyse and process the SLA. If the document is valid, the backend sends a series of CIM requests to the CIMOM to create the CIM instances and associations representing the content of the SLA. This leads to the instantiation of all the definition classes, which were described in section 3.1.

We distinguish between two types of providers: A WSLA provider, implementing the SLA model (maintaining the SLA definition and carrying out the computations) and one or more resource providers. The resource providers are responsible for exposing metrics from managed resources. In the next step (3), the classes of the WSLA provider that relate to metric values at runtime (described in section 3.2) calculate the SLA parameters based on the raw metrics obtained from one or more resource providers. The computation is either triggered by `IBM_Schedule` instances or CIM requests issued by an external CIM client, respectively. Finally, in step (4), the computed SLA parameters are forwarded as SOAP events to the Condition Evaluation Service (implemented as Web Service) through a SOAP library embedded in the WSLA provider. This mechanism is detailed in section 4.3. Note that since CIM providers are implemented on a per-class basis, the term *WSLA provider* is used as a shorthand for *the set of CIM instance and association providers implementing the SLA model.*

4.1 Active CIM Providers

One of the major novelties of our approach is the use of active CIM providers. While active management agents are known in the network management community for some time (e.g., the OSI Event Report Management Function or the Summarization Function), CIM providers are, until now, stateless resource providers. They are passive and only surface information from managed resources without providing advanced processing capabilities. Normally, the retrieval of information is initiated by a management system acting as a CIM client. Stateless providers may cause a considerable overhead by requiring periodic polling for new values.

In our implementation, the WSLA provider actively monitors the SLA by autonomously retrieving metric values from managed resources and calculating SLA parameters without being triggered by an external client. Instead, the retrieval of new metrics is automatically requested by the provider implementing the `IBM_Schedule` class of the SLA model (see section 3.2 for details). Having this delegated management functionality executed autonomously eliminates the need for polling and thus reduces the overhead significantly.

4.2 Recovery Mechanism

SLA management requires the continuous monitoring of SLA parameters and metrics. Since the CIM providers are not loaded automatically, but rather on demand, there is the potential problem that providers are not reactivated after a restart of the Measurement Service. Since the monitoring is triggered by the provider implementing the `IBM_Schedule` class, one needs to make sure this provider is properly reactivated.

Two different cases have to be considered: First, the deployment of a new SLA and, second, the restart of the CIMOM after a failure. When a new SLA is deployed to the CIMOM, the `IBM_Schedule` provider is loaded automatically, since every SLA contains one or more instances of the `IBM_Schedule` class. Thus, the first case is not a cause for concern. However, in the CIMOM restart case, providers are not loaded automatically, but only when a request for their data comes in. Consequently, a recovery procedure needs to be in place to guarantee that the `IBM_Schedule` provider is always loaded, since it issues requests to the providers of other classes (e.g., `IBM_TimeSeries`) of the SLA model, which forces the CIMOM to load them if they are not activated. In other words, the `IBM_Schedule` provider is used to bootstrap the other providers of the SLA model. This can be achieved by having a simple CIM client enumerate the `IBM_Schedule` instances once the CIMOM has started. Such a command can be included in the startup script of the CIMOM.

4.3 Event Forwarding

In addition to collecting resource metrics and computing SLA parameters, the Measurement Service needs to forward newly computed SLA parameters by means of asynchronous events to the WSLA Condition Evaluation Service (cf. Figure 4).

CIM defines an event model [3] that enables CIM clients to subscribe to events published as CIM objects by the CIMOM and its providers, respectively. The CIM event model distinguishes between `ClassIndication` (creation/deletion/modification of CIM classes), `InstIndication` (creation/deletion/modification/read of CIM instances as well as the invocation of methods), and `ProcessIndication` (not CIM-specific alerts caused by managed resources, e.g., SNMP traps). The client can apply filters to limit the number of events being generated. Events are delivered by a handler and are currently limited to the XML-encoded CIM operations. Other handler types are intended for the future, but not specified yet. In this approach, a CIM client would need to act as a gateway to forward CIM events into a non-CIM environment.

Since a generic CIM/Web Services interoperability solution would require the definition of mappings between the different information models and communication protocols (a non-trivial task comparable to generic management gateways, as described in [17]), we chose a more pragmatic approach that is not based on the CIM event model. Instead, we enable our WSLA provider to emit events directly to a WSLA Condition Evaluation Service. This implies bypassing the CIMOM and, thus, the CIM event model. Our task is facilitated by the fact that the provider is already aware of the SLA parameters that are to be sent to each party, because – in WSLA – this event subscription information is already part of the SLA. By doing so, we are able to augment the provider with a SOAP library, which allows the sending of SOAP notifications to the WSLA Condition Evaluation Service. This principle can be applied to notification mechanisms of other architectures (CORBA events, SNMP traps, etc.) as well.

5. Prototype Implementation

The prototype has been implemented using the SNIA (Storage Networking Industry Association) CIMOM v0.7 and the Java Development Kit (JDK) v1.3.1. In principle, every CIM class – as well as the associations – is implemented by a separate (instance or association) provider. This facilitates the adaptability of a model if it is subject to changes and reduces the complexity of individual providers.

However, the principle of implementing every CIM provider on a per-class basis could not be held up for the implementation of the `IBM_ArithmeticCompositeMetric`, `IBM_StatisticalCompositeMetric`, and `IBM_TimeSeries` classes because they have cyclic dependencies (cf. Figure 2). Consider, e.g., the case when the CIMOM is restarted after a failure and when persistent instances have to be reloaded (no persistent storage mechanism of CIM instances is standardized yet, thus every CIM implementation handles this differently) in order to monitor the deployed SLA(s). Each CIM instance of these metric classes is associated with a calculation object that performs the retrieval of raw metric values and the metric calculation. These calculation objects have to be initialized with the references to their associated objects. Assuming an `IBM_TimeSeries` instance has to be initialized, its associated class can be either of type `IBM_RawMetric`, `IBM_ArithmeticCompositeMetric`, or `IBM_StatisticalCompositeMetric`. If an operand is either an `IBM_ArithmeticCompositeMetric` or a `IBM_StatisticalCompositeMetric`, the attempt to resolve the reference to this instance would result in a request to the corresponding provider without having finished the initialization of the `IBM_TimeSeries` instances (cf. Figure 5). During the initialization of the other providers, a single reference to an `IBM_TimeSeries` instance would entail an attempt to initialize the `IBM_TimeSeries` provider again, thus resulting in a loop.

Combining the initially three separate providers into a single provider – responsible for handling all three classes – solves this deadlock situation because all metric instances are jointly restored from persistent storage. By doing so, the provider has all information available internally to resolve the references without having to rely on

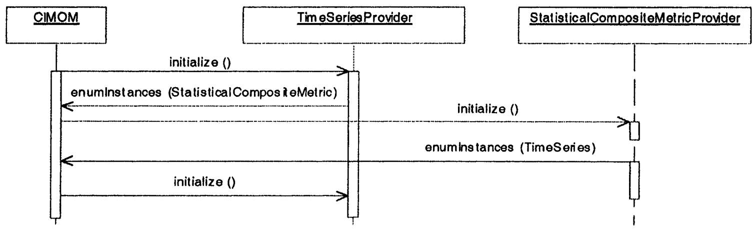

CIMOM
TimeSeriesProvider
StatisticalCompositeMetricProvider
initialize ()
enumInstances (StatisticalCompositeMetric)
initialize ()
enumInstances (TimeSeries)
initialize ()

Figure 5. Deadlock between different CIM provider classes

other CIM providers. In case of deploying a new SLA, this problem does not occur because the Deployment Service creates the instances in the correct order.

The CIM based Measurement Service and the WSLA Condition Evaluation and Deployment Services are new extensions to an existing prototype for discovering quantitative models for service level management (see [5] in this book for more details). The CIM based Measurement Service, described in this paper, is used to carry out SLA measurement tasks and sends its results through the WSLA Condition Evaluation Service both to the AutoTune Manager and a Managed Probe. The Measurement Service obtains its data from the IBM DB2 UDB database management system, which has been extended by a CIM based management instrumentation.

6. Conclusions and Outlook

We have presented a novel approach for SLA-driven management of distributed systems using CIM. It uses the WSLA framework to define and formally represent SLAs as XML documents. To access existing managed resource instrumentations, these documents need to be transformed into a representation compatible with typical management architectures. Here, we assume a CIM based management environment.

Our approach to this problem consists of developing a CIM based model for SLAs that preserves the expressiveness of WSLA, and an SLA Measurement Service, implemented as a CIM provider. The model is populated by a Deployment Service whenever a new SLA needs to be monitored by our system. The Deployment Service is implemented as a Web Service. It acts as a CIM client and is able to perform the mapping of the SLA into a CIM based environment. During deployment, the Measurement Service instantiates CIM objects that represent the SLA definitions. During the runtime phase, new metric data is collected from managed resources and subsequently aggregated into SLA parameters and forwarded to one or more Condition Evaluation Services, according to the schedule defined in the SLA. This implies that the data collection must be triggered from within the CIM Object Manager by means of so-called active CIM providers. This concept, available e.g., in the Summarization Function of the OSI/TMN management framework, goes beyond the capabilities of today's CIM providers, which are passive and need to be triggered by external management applications. The move from traditional stateless CIM providers to active CIM providers required some work, since typical CIMOM implementations are not geared towards active providers. In particular, we had to address the problem of ensuring a recovery after a CIMOM failure. Other problems were caused by the interdependencies between different CIM providers, for which we were able to find a solution.

While our work shows that it is possible to implement more advanced functionality – such as data collection capabilities – with CIM Object Managers, a more general approach to CIM provider initialization is needed to avoid potential deadlocks. In addition, the customized mapping of WSLA documents into the CIM schema needs to be expanded to support the full interoperability of CIM with a Web Services environment.

Acknowledgments

The authors express their gratitude to Bob Moore and Lee Rafalow of IBM for helpful discussions and ongoing advice and for their help to making this work succeed.

References

[1] ASP Industry Consortium. *White Paper on Service Level Agreements*, 2000.

[2] Common Information Model (CIM) Version 2.2. Specification, Distributed Management Task Force, June 1999. http://www.dmtf.org/standards/cim_spec_v22/.

[3] CIM Event Model White Paper. Version 2.6, Distributed Management Task Force, March 2002.

[4] Specification for CIM Operations over HTTP, Version 1.1. Specification, Distributed Management Task Force, May 2002. http://www.dmtf.org/standards/documents/WBEM/DSP200.html.

[5] Y. Diao, F. Eskesen, S. Froehlich, J.L. Hellerstein, A. Keller, L.F. Spainhower, and M. Surendra. Generic On-Line Discovery of Quantitative Models for Service Level Management. In G.S. Goldszmidt and J. Schönwälder, editors, *Proceedings of the 8th IFIP/IEEE International Symposium on Integrated Network Management*. Kluwer Academic Publishers, March 2003.

[6] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.

[7] I. Foster, C. Kesselman, J.M. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Service Architecture for Distributed Systems Integration. Draft, Globus Project, July 2002.

[8] IBM Corporation. *Web Services Toolkit version 3.2*, August 2002. Available at: http://www.alphaworks.ibm.com/tech/webservicestoolkit.

[9] A. Keller, G. Kar, H. Ludwig, A. Dan, and J.L. Hellerstein. Managing Dynamic Services: A Contract based Approach to a Conceptual Architecture. In R. Stadler and M. Ulema, editors, *Proceedings of the 8th IEEE/IFIP Network Operations and Management Symposium (NOMS'2002)*, pages 513–528, Florence, Italy, April 2002. IEEE Press.

[10] A. Keller and H. Ludwig. The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. *Journal of Network and Systems Management, Special Issue on E-Business Management*, 11(1), March 2003.

[11] *Keynote – The Internet Performance Authority*. http://www.keynote.com.

[12] H. Kreger. *Web Services Conceptual Architecture 1.0*. IBM Software Group, May 2001.

[13] L. Lewis. *Managing Business and Service Networks*. Kluwer Academic Publishers, 2001.

[14] L. Lewis and P. Ray. On the Migration from Enterprise Management to Integrated Service Level Management. *IEEE Network*, 16(1):8–14, January 2002.

[15] H. Ludwig, A. Keller, A. Dan, R.P. King, and R. Franck. *Web Service Level Agreement (WSLA) Language Specification*. IBM Corporation, November 2002.

[16] Matrix NetSystems, Inc. *Beyond Mere Latency*, 2002. http://www.matrixnetsystems.com.

[17] S. Mazumdar. Inter-Domain Management between CORBA and SNMP. In *Proceedings of the IFIP/IEEE International Workshop on Distributed Systems: Operations & Management*, L'Aquila, Italy, October 1996.

[18] C. Overton. On the Theory and Practice of Internet SLAs. *Journal of Computer Resource Measurement*, 106:32–45, April 2002. Computer Measurement Group.

[19] N. Soukouti and U. Hollberg. Joint Inter-Domain Management: CORBA, CMIP and SNMP. In A. A. Lazar and R. Saracco, editors, *Proceedings of the 5th International IFIP/IEEE Symposium on Integrated Management (IM)*, pages 153–164, San Diego, USA, May 1997.

[20] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, and C. Kesselman. Grid Service Specification. Draft 3, Global Grid Forum, July 2002.

[21] D. Verma. *Supporting Service Level Agreements on IP Networks*. Macmillan Publishing, 1999.

[22] Y. Yemini, G. Goldszmidt, and S. Yemini. Network Management by Delegation. In I. Krishnan and W. Zimmer, editors, *Proceedings of the Second International Symposium on Integrated Network Management*, pages 95–107. Elsevier Science Publishers B. V. (North Holland), April 1991.

SESSION 11

Management System Design

Chair: Olivier Festor
LORIA-INRIA, France

A MANAGEMENT-AWARE SOFTWARE DEVELOPMENT PROCESS USING DESIGN PATTERNS

Oliver Mehl, Mike Becker, Andreas Köppel, Partho Paul, Daniel Zimmermann and Sebastian Abeck
{oliver.mehl, mike.becker, andreas.koeppel, partho.paul, daniel.zimmermann, sebastian. abeck}@cooperation-management.de
Cooperation & Management
Institute of Telematics, University of Karlsruhe
Zirkel 2, D-76128 Karlsruhe, Germany

Abstract: The provision of quality-assured IT services through a service provider requires that all IT components involved in these services can be managed in an efficient and effective way. The necessary management infrastructure must be adapted to these IT components and must be standard-based to allow its integration into an overall management environment. The broad spectrum of applications differing in functionality and architecture together with the need for a deep correlation of the management infrastructure with the internal structure and processes of an application make it difficult to use pre-defined application management solutions off-the-shelf.

The development process described in this paper addresses the problem by integrating the development of the management infrastructure into the software development process. The integration assures that the management infrastructure is adapted to the application to be managed. The infrastructure, including the management model as its core component, is based on the Common Information Model (CIM) standard. To support the development of the management model, a management design pattern catalog is introduced, that provides CIM-based patterns for the definition of standardized management models. The use of this catalog is demonstrated by an extension to the management model for the distributed Enterprise Resource Planning System R/3 of SAP AG.

Key words: application management, software development process, management infrastructure, management model, design pattern, CIM

1. INTRODUCTION

Modern software systems have to be able to provide IT services to customers in a quality-assured way. Therefore, every component that forms an application - including network-, system- and application components - must provide a management infrastructure including management models and corresponding instrumentation mechanisms.

Due to the broad structural variety and individual functionality of application architectures, the process to create an adequate and sound management infrastructure is a complex task. Additionally, in contrast to network or systems management, where management models can be built on dedicated base models, in most cases application management models must be built from scratch. An in-depth look into the application and its internal processes is necessary to ensure that the generated model reflects the application system appropriately. This puts the task of creating such a management model into the hands of the software developer. Being the person most familiar with the application, he is predestined to deal with the provisioning of the necessary management infrastructure.

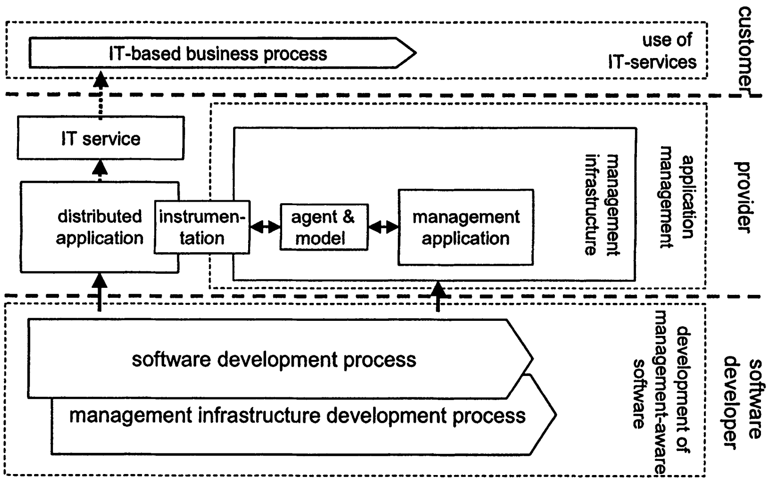

IT-based business process
use of IT-services
customer
IT service
distributed application
instrumentation
agent & model
management application
management infrastructure
application management
provider
software development process
management infrastructure development process
development of management-aware software
software developer

Figure 1. Development of management-aware software

To reduce the complexity of this task, an integrated process for the development of management-aware software systems as depicted in Figure 1 is necessary. A tight integration of the software and management infrastructure development processes avoids the problems of a too late alignment and harmonization of the application and its management infrastructure. To handle the additional complexity of this approach an integrated approach must provide guidelines and tools for application developers to ensure that management aspects are considered in all phases of the process.

Within this context the paper focuses on the development of the management model as part of the management infrastructure and demonstrates how the use of a design pattern catalog for management models supports the process. The design patterns presented describe recurring parts of management models for application systems, modeled on a level of abstraction suited for management needs using a standardized management modeling language to ensure reusability and portability.

2. STATE OF THE ART

2.1 Management of applications

The complexity of application management [1] is not only driven by its various functional aspects but also by the broad spectrum of applications as managed objects. An important requirement for a successful management of applications is an adequate information model, which describes the applications' functional components and relations in the context of the business processes. Regarding the way to develop these models two basic approaches can be identified.

The most frequently used approach at present is a disjoint development of the software system and the corresponding management model. In this case, the management model and instrumentation code [2] need to be developed and integrated after the implementation of the application. If integration is impossible, the management must be based on information already available from the software, complemented by information gathered from an outside view of the application (e.g. system management information). This approach bears the risk of significant differences between the representation of the application in the management model and the actual software structure. The quality of the management achieved in this approach depends heavily on the degree to that additional instrumentation code can be integrated into the existing application system.

The second approach is to integrate the development of the management model into the software development process. This approach reduces the danger of inconsistent views on the application through a tight coordination between the development processes of both models. Due to their different aims and viewpoints a total match of the models is unlikely. Nevertheless, an integrated approach usually results in a well-adapted management view on the managed application that directly influences the quality of the overall management solution. The management-aware development process presented in this paper follows this integrated approach as described in detail in section 3.

2.2 Management infrastructure

In today's management scenarios required information for managing an application is often provided trough a specific management infrastructure in a non-standard way. To be integrated into a broader context such as service management management information must be accessible not only by application-specific management tools, that are tightly coupled to the managed application, but also by external management applications or platform such as Tivoli [3], BMC [4] or HP Openview [5] to enable correlation and aggregation of the management information.

A standard for modeling management information is the Common Information Model (CIM) [6] specified by the Distributed Management Task Force (DMTF). In contrast to management information languages such as SMI-GDMO (Structure of Management Information, Guidelines for the Definition of Managed Objects) [7], DMI-MIF (Desktop Management Interface, Management Information Format) [8] or SNMP-SMI (Simple Network Management Protocol, Structure of Management Information) [9] that are devoted to particular management domains CIM is an implementation and architecture-independent modeling language that allows to describe overall management information in a networked / enterprise environment [10]. It is flexible and extensible and therefore provides a wide range of applications. The rules for building CIM-compliant management models are defined in the CIM meta schema [6]. A set of basic elements to model management information is provided through the CIM base schema. CIM makes extensive use of the concepts of classes and instances known from object-oriented modeling and uses UML class diagrams to describe the models. This fact supports the idea to use CIM in an integrated development process in which the software developer must carry out the development of the management model as he is usually already familiar with these concepts and the notation.

The CIM compliant management infrastructure can be used to guarantee standardized access to an application's management information and functionality. To provide a manageable application that supports this approach, a developer has to provide the CIM-based management model, additional instrumentation code as well as a CIM provider for the application that can be registered with the CIM object manager (CIMOM) to make the CIM model, management data and functionality available to management applications.

3. AN INTEGRATED MANAGEMENT DEVELOPMENT PROCESS

The idea to integrate the management development process into existing software development processes assumes the identification of a generic core process. This process can then be extended to incorporate management aspects taking into account the integrated development of a standard-based management infrastructure for a management-aware software system. The integrated process can then be mapped to the original development processes to make them management-aware. Obviously the integrated development approach increases the overall complexity of the software development process that must be handled by the software developers. Even though a separate development team that is specialized in management details might transparently develop the complete management infrastructure, an intense synchronization between both processes is inevitable and results in additional costs. On the other hand the integrated process avoids the costly and error-prone process of adding management components after the implementation of the core functionality of a software system. The consideration of management aspects right form the start of the application development improves the application's manageability in terms of the overall quality of its services.

When taking a closer look at established software development models such as the waterfall model, the V model or the spiral model [11] all of them share three main phases: a system analysis phase, a software design phase and an implementation and test phase. The integrated management development process depicted in Figure 2 is based on a software development process (SWD process) that consisting of these three shared phases. The management infrastructure development process (MID process) is separated into corresponding phases. The integration of both processes is done for each phase separately to ensure a consistent completion of each phase and by that to ease the integration of each of the phases into existing development processes.

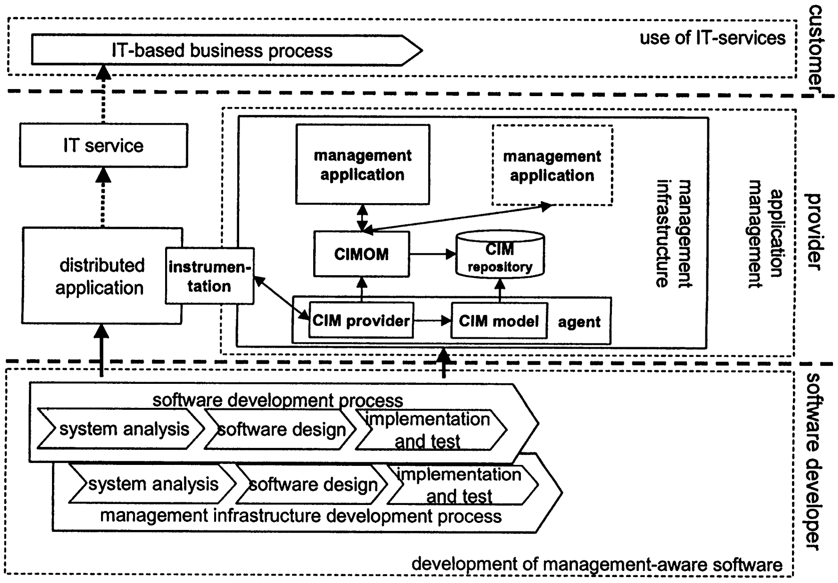


Figure 2. An integrated, CIM-based development process for management-aware software

To ensure a standardized access to the management information and functionality of the application the management infrastructure developed in this processes is supposed to be CIM-compliant. It deals with the development of the management model, the provider, the instrumentation code as well as the management application.

The following sections describe the first two phases of this development process. The last phase is left out as it concerns only implementation details.

3.1 The system analysis phase

The main focus of the system analysis phase is to determine and describe the requirements for an application system defined by the end user or customer. The requirements are documented in a product concept catalog and are later transferred into a customer requirement specification. Both documents describe the scope of the services of the developed application. In both cases the specification of the management requirements is realized after the requirements for the software have been defined as most management aspects are linked to or based on details of the software requirements analysis. To support the definition of management requirements a management catalog is used. The catalog provides a very high level classification of generic management information organized according to the five functional areas FCAPS of the OSI management [12]. It can be used in interviews to capture and define the customer's management requirements by instantiation and adaptation of the generic management aspects to the developed product.

After the requirements are defined, a product model is specified. Based on this model prototypes of the graphical user interfaces (GUI) are constructed that provide both, the software engineer and the customer, with a first vision of the application. In addition, a first version of a user manual is produced that is completed during the other two phases of the development process. If the development of a management application is necessary, the GUI for this application is specified in this phase following the same process but on a more general level. At first the management data provided by the management application is described in a broad outline through the definition of a first set of relevant management information and access functions for the application. This process is also supported by the management catalog that allows the definition of this data based on input from the requirement analysis. Additionally control mechanisms for the application are specified. Management data and management functions are then implemented in the management GUI.

The system analysis phase ends with the review of all the documents and specifications that have been created during this phase to guarantee that the specifications meet the requirements. In case of inconsistencies, changes or extensions to the existing specifications must be carried out.

3.2 The software design phase

The software design phase depicted in Figure 3 deals with the transformation of the specified requirements into a software architecture. During this process constraints and limiting conditions are refined and extended based on the outcomes of the system analysis phase. In addition, operation conditions are defined.

The software architecture describes the structure of the software system by defining the system components and their relationships. A system component is a closed part of a software system dealing with a specific (functional) aspect of the application. Depending on the level of detail the components are organized into layers or tiers whereas logical layers are mapped onto physical layers. The mapping of the components can be done following predefined schemes.

After the system components and the architecture have been defined, the components are specified in detail. This includes the definition and documentation of their interfaces as well as their internal behavior in a formal or semi-formal way. Tools supporting this process are e.g. UML CASE tools to create static (class diagrams) or dynamic (activity or sequence diagrams) diagrams of the components. Due to its complexity, the process of defining the software architecture and its components is carried out iteratively, refining the results of the preceding phase. If necessary, results must also be fed back into prior phases of the design process to change or extend earlier specifications or constraints.

The design of the management model can be initiated after the system components and the overall architecture have been defined. The management model can include various kinds of management information from the different management areas depending on the focus of the management. Constraints and decisions taken in the design of the application architecture influence the development of the management model. They must be taken into account to ensure a consistent and sound management view of the application. To ease the development of the management model, a management design pattern catalog is used. The purpose of this catalog, its structure and usage is described in section 4 in detail.

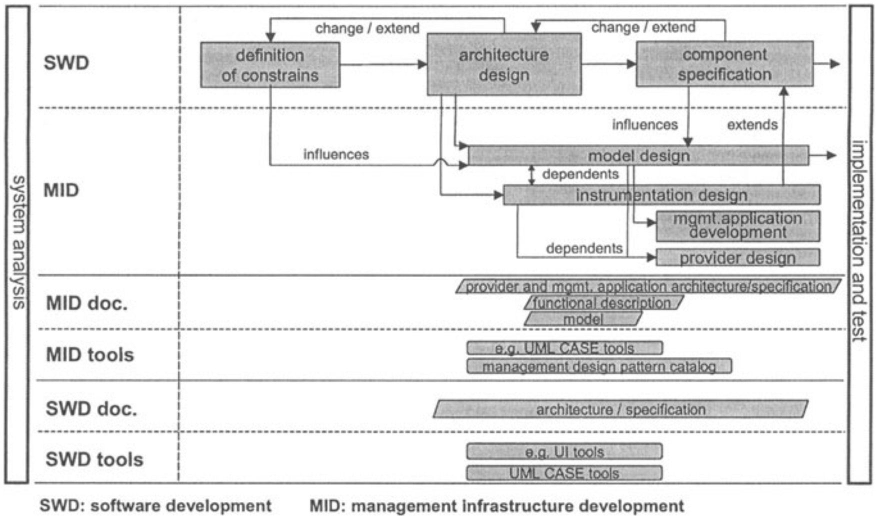


Figure 3. Management-aware software design phase

The management information and functionality is provided by the application components through the instrumentation. Therefore the instrumentation requirements extend the component specification as they define additional functions and parameters that need to be integrated into the components. The instrumentation also influences the design of the management model: only aspects that can be covered by the instrumentation can be instantiated in the management model.

After the management model and instrumentation design become more stable, the design of the management provider and, if necessary, the management application development is initiated. The provider is responsible to transfer the management data received from the instrumentation to the management application (via the CIMOM) and must therefore implement the management model. The management application is designed and refined according to the requirements and the initial GUI design defined in the system analysis phase. Based on the information represented in the management model, details about management data presentation and possible data manipulation operations are specified.

4. A MANAGEMENT DESIGN PATTERN CATALOG FOR THE SOFTWARE DESIGN PHASE

The component- or object-oriented way to design the CIM-compliant management model in the integrated development process fosters the reuse of design models in the form of patterns that solve recurring design problems. In software engineering design pattern consist of a combination of components or classes, their interfaces and relationships that reflect important aspects of a solution to a specific problem. The reuse of these patterns is only possible if they provide a sufficient level of abstraction from both the underlying problem and the solution. While software design patterns usually handle details of the system design, patterns used to build management models operate on a higher level of abstraction relying on component- and function-oriented view of an application system. Following this approach a management design pattern catalog can be used to assist in the development of management models. Using such a catalog provides several benefits: The developer of the management model gets a set of building blocks that allow an efficient implementation of the model. The patterns also support the development of the application system instrumentation as they specify operations and attributes that must be provided by the application to enable its management. Both aspects reduce the effort to develop an appropriate management infrastructure for an application as only additional application-specific characteristic that are not covered by the patterns must be added to the model. Additionally, the use of management design patterns stimulates the development of comparable management models. By basing the representation of application system components in the management model on well-defined building blocks the interoperability with other models providing similar attributes and inheritance hierarchies is supported.

4.1 Management design pattern catalog

In contrast to application software design the management model design focuses on the functional units and services of an application on a reasonable level for management tasks. When analyzing the functional aspects of the application various indicators help the developer to discover such entities. Examples are components supporting asynchronous data processing or boundaries between software components that are embedded in separate executables or communicate remotely.

The management design pattern catalog supports the developer in identifying such entities as well as in the selection of the parts of these entity that are relevant to be include in the management model. The resulting requirements for the instrumentation of the application system can then be fed back into the application design process. The level of abstraction used for the patterns is located in between the rather abstract concept of architecture styles (see [13]) and the fine-grained software design patterns used in the software design process. The patterns focus on the description of logical processing elements such as services and sub-systems. The patterns are described following the approach used in [14] to ease their selection and use by software developers. Therefore each pattern in the catalog is described by its name, a short overview as well as examples for its use. Synonyms and related terms are mentioned to ease navigation and search for an appropriate pattern. While this information is intended to support the selection of a pattern from the catalog, each pattern is also described in detail and visualized by a CIM model. Preconditions and limitations for the use of the pattern are mentioned.

The initial set of design patterns for the management design pattern catalog were identified during the design of a management infrastructure for the Enterprise Resource Planning System (ERP System) R/3 of SAP AG. A strong effort during the analysis was put on keeping the identified patterns independent from the specific characteristics of the SAP software to guarantee their reusability and to define them on a suitable level of abstraction to keep them applicable for other scenarios.

Two high-level patterns were identified as starting points for further analysis:

- The basic structure of an application system as a class model is presented in the *distributed application system* pattern.
- Systems featuring logical segments with private data and configuration sections may be modeled using the *logical system* pattern. A web server providing services for different virtual sites on the same hard- and software platform can be taken as an example for such a system.

Design patterns for single services that were identified in the functional analysis include:

- An *application service* represents a service that provides a set of related functions in an application system.
- A *distributed service* is provided by services of different component systems. A load balancing service can be taken as an example for this pattern.
- A *specialized service* represents a service implemented from specialized components or component systems within a distributed application system, playing a vital role as a single point of failure for the system.
- A *queued service* uses data queues or asynchronous request processing to provide its service. An example is a request processing service of a print server.

In addition, the following patterns can be used to describe more general component relationships.

- The pattern *client-server relationship* represents the relationships between two components that are part of a service provide/service user relationship. The

communication relation ship between business tier components and a database can be described using this pattern.
To create a model describing the properties of an external component that is used by an application system the pattern *external view* may be used.

4.2 Pattern examples in detail

In the following sections the design patterns *distributed service* and *specialized service* are described in detail using the catalog structure introduced in section 4.1.

4.2.1 Application Service

Overview: An *application service* combines a set of functions of a technical service. All processing elements of an application can be mapped to such a service. Criteria for this kind of services are a fixed interface and permanent service availability.
Example: A spool service for print jobs or a web server module such as a SSL module for communication encryption can be modeled as application service.
Related terms: functional group, processing element, module

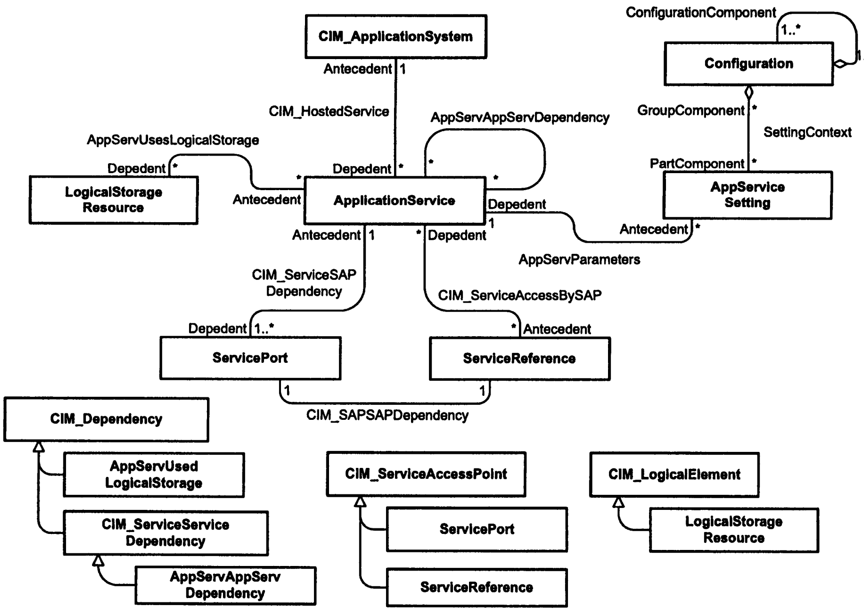


Figure 4. CIM model of the management design pattern *application service*

Motivation: The pattern allows to model the management aspects of any relevant components of an application system from a functional point of view.
modeling details: The *ApplicationService* class is linked to the *CIM_Application-System* class that implements the service functionality. It can use other application

services to implement its functions or provide its own functionality to other services. This is done by the use of a *ServiceReference* (to use functions of other services) or a *ServicePort* (to provide functionality) that are derived from the class *CIM_Service-AccessPoint.* The use of data storage systems is modeled through the class *Logical-StorageResource.* Configuration options are mapped to the *AppServiceSetting* class derived from *CIM_Configuration* that represents a part of the overall application settings.

Preconditions and limitations: The key problem to use this pattern is the identification of the relevant elements of an application that should be modeled as application services. Therefore an adequate level of abstraction of the software component model must be chosen before this pattern can be used.

Visualization: See Figure 4

4.2.2 Specialized Service

Overview: The pattern *specialized service* represents a service that is implemented only by specific components and is of central importance for the overall system.

Example: The payment transaction component of an e-commerce application can be seen as a specialized service. It is the single interface of the application providing access to the bank to carry out financial transactions and it is used by all other components of the e-commerce application requiring this functionality.

Related terms: Dispatcher, master, single point of failure

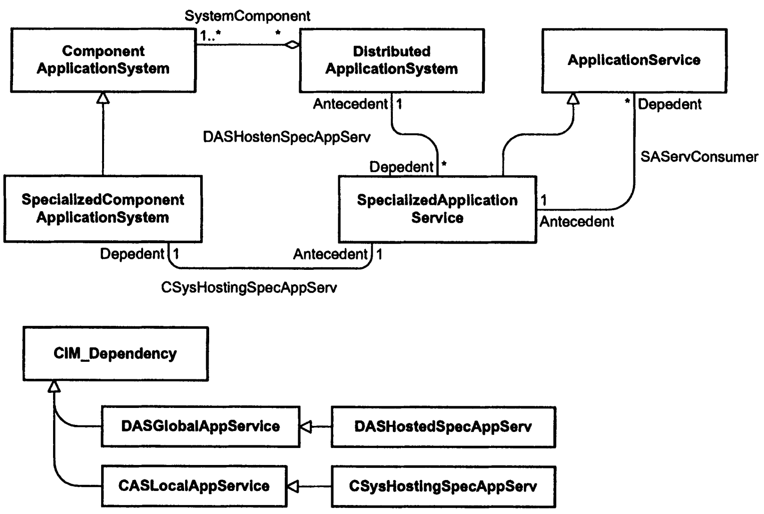


Figure 5. CIM model of the management design pattern *specialized service*

Motivation: Specialized services are of special interest from a management point of view regarding fault or performance management. A failure on their part has major implications on the overall system availability. Their performance influences the overall productivity of the system.

Modeling details: A specialized application service is modeled by the *Specialized-ApplicationService* class derived from the *ApplicationService* class. This class is associated with a component system *SpecializedComponentApplicationSystem* that provides the service. The specialization of this system does not necessarily involve special software components but can also be achieved through specific configuration options. The service is used by other application services indicated by the *SAServConsumer* association to the *ApplicationService* class.
Preconditions and limitations: The overall application system must be built from different component systems. The services provided by the application system must be provided through component-based services. The relationships between these services must be known to allow the identification of specialized services. The pattern does not represent the functional aspects of the relationships.
Visualization: See Figure 5

5. USE OF MANAGEMENT DESIGN PATTERNS IN THE CONTEXT OF THE ERP SYSTEM SAP R/3

The overall architecture of the ERP system SAP R/3 follows a distributed three-tier architecture. In this scenario the database tier is a centralized external relational database providing its services to the business tier. In the business tier so called work processes use this central database to provide their functionality in the context of various business processes. The work processes are grouped to application servers that are located on different physical systems. Each application server consists of exactly one dispatcher process and an optional gateway service to handle the distribution of requests between the local work processes and the communication between the applications servers of a single SAP system.

The Internet Communication Manager (ICMan) component enables an application server to communicate using IP-based protocols such as HTTP or SMTP. This is achieved by extending the application server by a separate ICMan process that can be addressed by the work processes through the dispatcher process.

In the following the use of the management pattern catalog is demonstrated in the development of a management model for the ICMan.

5.1 Management model for the ICMan component

Based on the functional and architectural aspects of the ICMan component the management pattern *specialized service* was identified as best choice to develop the management model. The service provided by the ICMan can be characterized as specialized service, because its service is provided by not more than one component instance but it is accessed by several other components to provide their services. Even though the ICMan component can't be seen as central in the overall application context it is central in the context of its associated application server.

Figure 6 depicts the CIM model of the ICMan component as it was integrated into the management model of the SAP base system. According to the design pattern *specialized service* the ICMan component was modeled as a CIM class *SAP_BCInternetCommMgrService* derived from the abstract class *SAP_Application-Service* that corresponds to the class *ApplicationService* in the design pattern. The association *CsysHostingSpecialService* is instantiated as *SAP_BCKernelICMan-Implementation* and is linked to the *SpecializedComponentApplicationSystem* that implements the component system including the ICMan, which in this case is represented by *SAP_BCKernel.* As the service of the ICMan is used by the work processes of the application server the dependency *SAServConsumer* is modeled by the association *SAP_BCICManProvidesServiceToWP* between the *SAP_BCInternet-CommMgrService* class and the *SAP_BCWorkProcess* that correspond to the *ApplicationService* class in the pattern.

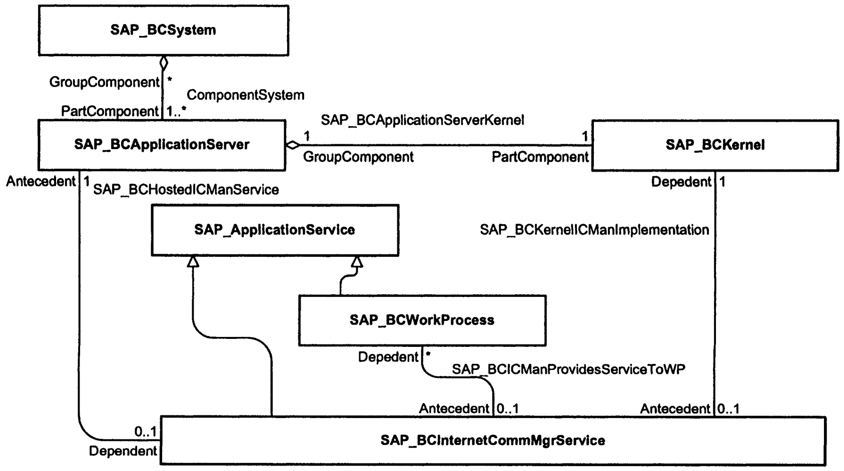


Figure 6. CIM model of the ICMan component

6. CONCLUSION

The management-aware development process depicted in this paper presents an approach for the integrated development of applications and their management infrastructure by the inclusion of management aspects in all phases of the software development. It fosters the alignment and integration of the management infrastructure, weaving the management model with the application system in a standardized way. The design pattern catalog presented supports application developers to handle the additional complexity of the integrated development process. It provides as set of patterns to ease the construction of CIM-based management models as an important building block of the management infrastructure. The use of the pattern catalog was demonstrated in the development of the management model of a central component of an ERP system.

The integrated development process is currently being evaluated in the development of different management-aware components. The results are fed back into the ongoing improvement of the process and the supporting tool set. Actual activities focus on the specification of an extended schema for a product concept catalog and a customer requirement specification used in the system analysis phases as well as customized methods for using them efficiently. As one of the supporting tools the design pattern catalog is also continuously evaluate, revised and extended by modeling further components of the SAP ERP system to improve and verify the reusability of the design patterns defined so far.

REFERENCES

[1] R. Sturm and W. Bumpus, "Foundations of Application Management": John Wiley & Sons, 1999.

[2] M. Katchabaw, S. Howard, H. Lutfiyya, A. Marshall, and M. Bauer, "Making Distributed Applications Manageable Through Instrumentation", presented at 2nd International Workshop on Software Engineering for Parallel and Distributed Systems (PDSE'97), 1997.

[3] IBM, "Tivoli® Software", http://www.tivoli.com.

[4] B. Software, "Enterprise Applications Management", http://www.bmc.com.

[5] H.-P. Company, "HP OpenView", http://www.openview.hp.com.

[6] DMTF, "Common Information Model (CIM) Specification, Version 2.2", http://www.dmtf.org/standards/cim_spec_v22/.

[7] ISO, "International Organization for Standardization - Information Technology - Open Systems Interconnection - Structure of Management Information: Guidelines for the Definition of Managed Objects, ISO 10165-4", 1992.

[8] DMTF, "DMTF, Desktop Management Interface (DMI) Standards", http://www.dmtf.org/standards/standard_dmi.php, 1998.

[9] IETF, "RFC1155 - Structure and Identification of Management Information for TCP/IP-based Internets , Internet Engineering Taskforce", 1990.

[10] W. Bumpus, "Common Information Model: implementing the object model for enterprise management". New York, N.Y: Wiley, 2000.

[11] G. Kotonya and I. Sommerville, "Requirements Enginerring": John Wiley & Sons, 2000.

[12] ISO, "International Organization for Standardization - Information Technology - Open Systems Interconnection - System Management: Object Management Function, ISO 10164-1", 1993.

[13] D. Garlan and M. Shaw, "An Introduction to Software Architecture", vol. Volume I: World Scientific Publishing Company, New Jersey, 1994.

[14] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "Design Patterns: elements of reusable object-oriented software": Addison-Wesley Publishing Company, Reading, MA, 1995.

MANAGING VIRTUAL STORAGE SYSTEMS: AN APPROACH USING DEPENDENCY ANALYSIS

Andrzej Kochut *
Computer Science Department
University of Maryland
College Park, Maryland 20742, USA
kochut@cs.umd.edu

Gautam Kar
IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598, USA
gkar@us.ibm.com

Abstract:

We present an approach for managing the performance of virtual storage systems by experimentally identifying the dependencies that exist between various components that comprise the system. Specifically, we show how one may profile dependencies between each logical volume exported by a storage system and components that this volume uses. To do so the technique estimates the arrival rate and size of requests issued to the internal system component as a functions of arrival rate and size of requests issued to the logical volume. The complete dependency profile of the system consists of a set of such functions for READ and WRITE operations separately and for each pair: logical volume - internal system component. The empirical technique of obtaining such profiles for typical existing storage systems is presented. We propose the use of Common Information Model (CIM) as a way to express dependency and performance information in an architecture-independent manner. The dependencies between components are computed as a fraction of bandwidth that is passed on to the sub-components. We discuss how the dependency profile of the system may be used to perform root-cause analysis and early Service Level Agreement violation notification. We also demonstrate the use of the method by applying it to a Linux system using software RAID.

Keywords:

Dependency analysis, virtual storage systems, root-cause analysis, Service-Level Agreement, systems monitoring.

1. Introduction

Fast growth in demand for big, efficient and reliable storage systems has led to the development of very complex and heterogeneous architectures. With this increase in architectural complexity and size of storage, new challenges for design, monitoring and management have emerged. The share of management costs in the Total Cost of

*Work done while author was an intern at IBM T. J. Watson Research Center.

Ownership for storage systems has increased steadily, creating a need for storage management systems that are scalable and autonomous. Management problems associated with storage systems start with the design of the system itself. The tasks of optimal allocation of data objects to logical volumes, configuration of software and hardware components, determining root-causes of problems, and predicting the violation of Service Level Agreement (SLA) prove to be very difficult. There is substantial amount of work done in the area of virtual storage design. The suite of storage design algorithms presented in [1] and [2] may be used to automatically design and configure a set of RAID arrays, given performance and reliability requirements of the request streams. In [9] authors show a way to automatically configure Storage Area Network to suit the needs of predicted data transmission. Difficulty in the design of the system stems from the trade-off between the utilization of components of the storage system and the nature of guarantees in the SLA used by the users of the system. Moreover, the design of the storage system is an ongoing process, because needs/demands of the users change constantly, making it necessary to re-evaluate storage requirements and reprovision storage allocation. Since, today, storage systems, in the form of SANs, are shared between multiple servers, and hence multiple applications, such reprovisioning is necessary to honor SLAs between the storage service layer and the applications.

In this paper we concentrate on aspects of performance problem detection and resolution in distributed environments consisting of virtual storage systems, e.g. SAN. The special emphasis of this research has been to understand how problems identified at the storage layer can be related to problems manifested at the application layer. The approach we have taken is to develop methods to characterize and compute dependencies that exist between virtual storage entities that applications and file systems use, and physical storage entities, such as RAID drives, into which the virtual entities are mapped. We propose modeling the dependencies between components of the system as a set of functions representing the arrival rate and size of requests issued to the internal component as a function of arrival rate and size of requests issued to the logical volume exported by the system. These functions are numerical representations of the storage policies implemented by the system. Storage policy defines the way data is stored and retrieved, what communication media are used to transfer the data, and what additional operations are performed while executing the request. For example, RAID1 ([4], [6]) storage policy defines on which physical devices the data is replicated, from where the data is read (load balancing), how the system behaves when one of the devices fails, etc. Figure 1 depicts a virtual storage system. Logical volumes *LV1* and *LV2* use a fiber switch to transfer the data to and from hard drives and virtual storage systems, such as IBM's Enterprise Storage Server (ESS). Data objects (*DO1* through *DO5*) are mapped onto logical volumes. Streams of requests (*S1* through *S6*) access the data objects.

Although it is sometimes possible to obtain an analytical model of the storage policy (one of the examples may be found in [8]), in general storage systems it may be very difficult to obtain a model that is close to reality. Thus we propose an empirical technique for estimating the dependency profile by applying controlled, variable load to logical volumes and observing the impact it has on internal system components. Our approach requires instrumentation of the system that can supply the management application with data about the component's performance. Because of the heterogeneity of the storage systems, the data needs to be expressed in a uniform, standard way to enable interoperability between components supplied by different vendors. To achieve

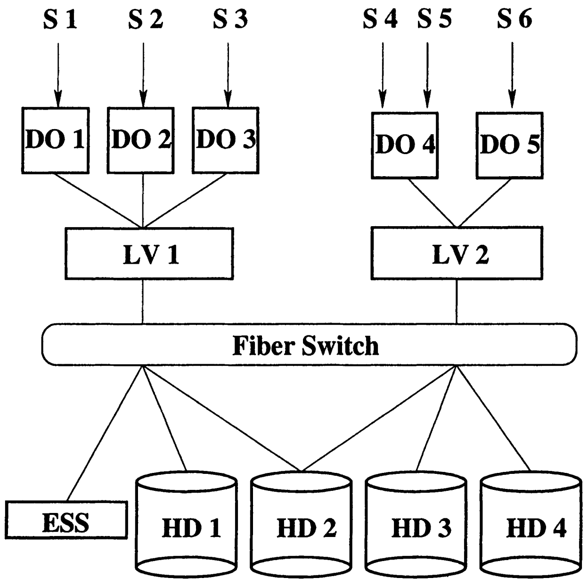


Figure 1. Example of a virtual storage system. Logical volumes *LV1* and *LV2* are mapped onto physical drives *HD1* through *HD4* and *ESS* using fiber switch. Data objects (*DO1* through *DO5*) are mapped onto logical volumes. Streams *S1* through *S6* access data objects.

this goal we propose the use of Common Information Model (CIM) as defined in [5]. CIM is a set of classes that may be used to describe system components as well as metrics associated with each component. By use of this standard, the management application may gather system-wide information about each component's performance and use it to prepare dependency profiles and later monitor the behavior of the system. Our method is similar to Active Dependency Discovery (ADD) technique presented in [3], where the authors perturb a system's components in order to estimate dependencies between them. However ADD is used for application-level dependencies while our method quantifies the performance dependencies between the elements of the storage subsystem.

The remainder of this article is organized as follows. Section 2 describes the way we model the system and use the model for computing dependencies. In Section 3 an example application of our method to the Linux Software RAID is presented. Finally, in Section 4 we discuss our future research plans.

2. Active Modeling Technique

Today's storage systems are usually heterogeneous, capable of storing terabytes of data and employing complex storage policies. A virtual storage system exports to its users a set of logical volumes that resemble physical devices. However, usually they are not mapped directly to the underlying physical devices in a simple one-to-one manner. Instead, the mapping is generally very complex. All these factors make a virtual storage system very difficult to design, model and monitor.

2.1 Model of the Virtual Storage System

We model the storage system as a set of independent components treated as "black-boxes" that execute requested actions by interacting with other components of the system. Each storage component presented with a request (i.e. READ or WRITE operation issued by another component or upper-level subsystem) performs internal computations and may issue requests to one or more other lower-level components. The request may be fulfilled in synchronous or asynchronous fashion. Moreover, we assume that each component may buffer requests and perform various optimizations on them before requesting processing from other components. For example, it is quite typical for storage systems to coalesce requests in order to increase sequentiality of physical disk access. In our example from Figure 1, the model of the system consists of eight components corresponding to two logical volumes, fiber switch, ESS and four hard drives. The way the components interact is decided by a set of policies that are defined for the system. For instance, policy of the logical volume *LV1* determines what actions are performed once the *LV1* is presented with the READ request. A simple scenario may involve issuing two READ requests to two hard drives and then transmitting the data using one of the channels of the fiber switch. However, in more realistic, real-world situations, the sequence of events may be considerably more complex. Various mirroring and striping techniques, multiple levels of caching, and other storage optimization strategies may make the process very tough to model and understand.

We believe that due to the complexity and heterogeneity of storage systems it is very important to have a modeling technique that could be applied with different levels of granularity. For example, we may want to model IBM's ESS either as a single storage element exporting storage space characterized by the size of the storage, its performance and reliability properties, or we may want to model it at a greater level of detail taking into consideration its internal structure. The choice between various levels of detail depends on available performance data as well as on the precision of predictions we want to make. Our model is flexible, making it possible to choose the level of detail that is suitable for a particular system. In a more elaborate scenario, instead of one component representing ESS, we can have many interacting components corresponding to elements of the drive arrays, cluster caches and other performance-related elements of the architecture.

The second part of our model consists of sets of metrics associated with each storage component. We identified three metrics as being crucial for understanding the behavior of a system's component: request arrival rate, size of requests, and request service time. An important metric that may be computed based on the above values is the utilization of a component. We use the standard definition of utilization as a multiplication of arrival rate and service time. The utilization indicates how close a given component is to becoming a bottleneck.

In order to make it possible to gather system-wide information in the heterogeneous virtual storage system we propose the use of Common Information Model (CIM) defined by Distributed Management Task Force. CIM is a common data model for describing management information. In particular, the standard contains a set of classes describing virtual storage subsystem as well as general framework for expressing metrics associated with system's elements. Each vendor (hardware or software) that contributed to the storage system that we want to monitor should provide the performance metrics described in the standard. The management application may access the performance data of all hardware and software elements constituting the virtual storage system. In the example from Figure 1, manufacturers of hard drives, fiber switch, and ESS should provide specific software that could populate performance information in the CIM schema. Similarly, software vendors should provide corresponding performance information about logical volumes. We believe that existing software drivers may be easily extended with the performance metrics we require. In order to use CIM with our method it is sufficient to add instances representing the performance metrics to the class *MetricDefinition*. Modeling of components and their dependencies may be done using classes from *CIMDevice* schema. A good example of modeling the storage system using CIM may be found in [7]. In Section 3 we present an example of application of our method to the Linux software RAID. We demonstrate how to extend the information exported by each device participating in the virtual storage subsystem with performance metrics required by our method. The extension task proves to be easy and non-intrusive.

2.2 Dependency Discovery Technique

We describe a method to dynamically gather information about the way components of a virtual storage system interact with each other. Our method consists of three major stages: instrumenting the system, applying controlled load to the collection of logical volumes, measuring the metrics (identified in the previous section) associated with the physical storage volumes, and finally, analyzing the obtained data and preparing the dependency profiles.

In the instrumentation stage, each component that may influence the performance should export the metrics described in Section 2.1. The use of CIM makes it possible to have multi-vendor systems be monitored using a common data model. In practice, each vendor would supply its specific CIM provider (piece of code publishing the performance data in the common CIM data repository). We have implemented an experimental CIM provider for our Linux system described in Section 3.

The second stage is devoted to inter-component dependency discovery and quantification. In order to do so, we propose estimating the impact of each of the logical volumes on the physical system components. Given a logical volume V and a component C of the storage system, we want to measure the way a stream of requests applied to V affects the system component C. We model this dependency as a set of functions computing arrival rates and sizes of requests issued to component C in terms of arrival rates and sizes of requests issued to logical volume V. We call this set of functions a **dependency profile** of component C on logical volume V. In the general case, each type of operation issued to logical volume may cause an arbitrary operation on any other component of the system. However, in most of the storage systems the situation is simpler. The WRITE operation causes only WRITE operations to be issued to other components, and similarly the READ operation causes only READ operations to be issued to other components. However, it may be the case that the WRITE request issued to a logical volume causes a READ request to be issued to one of the components. For example, if the storage system uses indexing, the index may be read in order to obtain information needed to perform the WRITE operation. Thus the full *dependency profile* of a component C on logical volume V consists of four functions:

- $READ\_RATE(V, C, type, rate, size)$ - average rate of arrival of READ requests to the component *C*, given that requests of type *type* arrive at *V* with average rate *rate* and are of average size *size* and all remaining logical volumes remain idle.
- $READ\_SIZE(V, C, type, rate, size)$ - average size of arrival of READ requests to the component *C*, given that requests of type *type* arrive at *V* with average rate *rate* and are of average size *size* and all remaining logical volumes remain idle.
- $WRITE\_RATE(V, C, type, rate, size)$ - average rate of arrival of WRITE requests to the component *C*, given that requests of type *type* arrive at *V* with average rate *rate* and are of average size *size* and all remaining logical volumes remain idle.
- $WRITE\_SIZE(V, C, type, rate, size)$ - average size of arrival of WRITE requests to the component *C*, given that requests of type *type* arrive at *V* with average rate *rate* and are of average size *size* and all remaining logical volumes remain idle.

All of the above functions characterize the impact of requests issued to the logical volume *V* on the internal system component *C*. This interaction is determined by a number of factors, the most important of which is the storage policy of the storage system. This is a set of algorithms governing the way the data is stored and retrieved. For instance, in the example from Figure 1 the storage policy determines what operations are performed once the READ request is issued to the logical volume *LV1*. It decides from which physical drives the data should be read as well as what communication channel in the fiber switch should be used to transmit the data. Some of these choices may be random. For example, if *LV1* implements RAID 1 algorithm and uses drives *HD1* and *HD2*, then the system may randomly choose the drive from which the data should be read. It is quite common to have load balancing techniques that use randomization to maximize the parallelism of data access.

Complexities of the virtual storage systems make it very difficult to compute the above functions analytically. We propose estimating them by applying controlled variable load to components of the system. Assuming that all performance-critical elements were instrumented and the data is readily available for the management application using the CIM standard, we can observe the impact of request streams issued to the logical volume on components of the system. In this way, by varying the type (i.e. READ or WRITE), arrival rate, and size of requests issued to the logical volume, we may obtain sample values of the functions constituting the *dependency profile*. During the measurements we increase the load until the utilization of one of the components approaches 1. At that point we know that the maximum capability of the subsystem was reached, and sampling may stop. In order to obtain service time profile of the component, we apply varying load to this component and observe the variation of its performance metrics. By varying arrival request rate and size of requests, we may obtain samples of the service time function. Similarly, as in the case of computation of the *dependency profiles* we stop increasing load once one of the components becomes fully utilized.

After obtaining *dependency profiles* for each pair (logical volume - component) as well as service time functions for each component of the system, it is possible to determine the dependencies between elements of the storage system. We propose using effective bandwidth as a measure of dependency. More precisely, for each logical volume *V* we identify a set of components that *V* depends upon. This set may be identified by inspecting *dependency profiles*. If the size and rate functions for a volume-component pair are non zero, it means there is a dependency between these components. For each component *C* on which *V* depends, we compute the bandwidth demand put on component *C* by traffic coming into component *V* as

$$BW(V, C, type, rate, size) = WRITE\_RATE(V, C, type, rate, size) *$$
$$WRITE\_SIZE(V, C, type, rate, size) + READ\_RATE(V, C, type, rate, size) *$$
$$READ\_SIZE(V, C, type, rate, size)$$

As a measure of the **dependency strength** between logical volume *V* and component *C* we elected to use the coefficients obtained by first order linear regression applied to pairs: bandwidth incoming to *V*, bandwidth demand put upon component *C*. We believe, that the distribution of the bandwidth depends primarily upon the storage policy, and not the size and arrival rates of requests. Experiments that we have conducted with Linux Software RAID support this assumption. However, in a general case, it may be possible that the dependency between components changes with the load applied to the system (i.e. changes in the arrival rate or average request size of the stream accessing logical volume). In that case we have a dynamic dependency quantification updated each time the traffic pattern changes. However we believe that this situation is not likely in the real-world storage systems. Thus, in the article we assume that the dependency strength may be modeled using linear regression on bandwidth distribution.

2.3 Applications of the Method

Dependency information, as collected in the previous section, may be used to build a dependency graph. While monitoring a set of applications, e.g. a file system, if a performance deterioration is observed, it would be possible to narrow down the set of probable causes by traversing the graph. The root-cause of the problem may then be determined by analyzing the set of suspected components resulting from the traversal. In our interpretation, for pair of components *V* and *C*, a dependency strength of 1 indicates that full bandwidth is passed on to component *C*. Values higher than 1 suggest that additional traffic is generated in order to execute requests. Value of zero denotes lack of any dependence between *V* and *C*. Values higher than zero, but smaller than one imply partial dependence between components *V* and *C*. These interpretations may be used as hints while traversing the graph in search of the root-cause of the problem. Figure 2 depicts example dependency graph for system from Figure 1 for READ operation. Arrows denote dependency between components of the system. Values associated with arrows in the graph denote strengths of dependencies. For example, data object *DO2* depends on logical volume *LV1* with strength 1. *LV1* depends upon *ESS* with strength equal to 0.8, which means that 0.8 of the bandwidth of READ operations issued to *LV 1* is "handed over" to *ESS*.

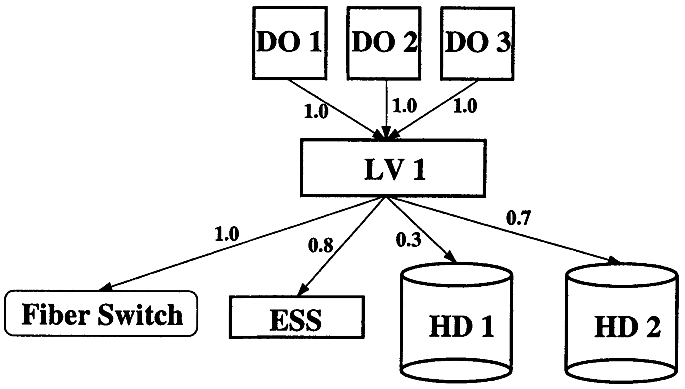


Figure 2. Example of the dependency graph for READ operation for the storage system from Figure 1.

Another application of the model is SLA violation prevention. Assuming that for a given system all of the performance profiles have been obtained, we can monitor each component of the system during its operation and alert the users (i.e. the operating system that uses the storage subsystem) about the possibility of SLA violation. Performance related SLA requirements are typically expressed as constraints on the maximum time spent by the system on processing a request, given that the arrival rate of requests and their sizes remain within the declared bounds. To forecast an SLA violation we propose monitoring the utilization of each component. Whenever the utilization of a component approaches one, the probability of queuing delays on this component increases. Using the dependency information it is possible to determine which data objects and requests streams may be affected by the delay and issue an alert to owners/issuers. The notification may be used by the users to throttle down the request rate or to take other preventive actions.

The third possible use of our model is data objects placement. Given an existing virtual storage system, the act of assigning data objects to logical volumes is a difficult one. The obtained *dependency profile* may be used to predict utilization of internal components given characteristics of request streams accessing the data objects and placement of data objects. Thus we may use our profiles as input to a verification stage in the optimization algorithm, searching the exponential space of possible placements. However, issues related to this application are beyond the scope of this article.

3. Method application to Linux Software RAID

To demonstrate the use of our method we present its application to an example virtual storage system. Our experimental system consists of Linux RedHat 7.1 running kernel 2.4.9. The virtual storage system consists of two hard drives and two logical volumes: RAID 0 and RAID 1. Figure 3 depicts this configuration.

Two physical hard drives *HDA* and *HDC* were partitioned into *HDA1*, *HDA2*, *HDA3*, and *HDC1*, *HDC2*, respectively. The first logical volume implemented software RAID 0 algorithm and used three partitions *HDA1*, *HDA2*, and *HDC1*. The second volume, implementing software RAID 1 algorithm, spanned two partitions:

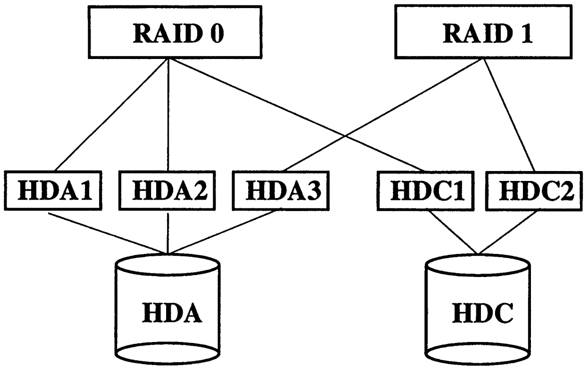


Figure 3. Linux software RAID architecture used for the experiment. Logical volume implementing RAID 0 algorithm spans partitions *HDA1*, *HDA2*, and *HDC1*. RAID 1 uses partitions *HDA3* and *HDC2*.

HDA3, and *HDC2*. To instrument the system we extended the standard Linux kernel statistics with metrics defined in Section 2.1. For each block device we obtained: average request arrival rate, average service time, and average size of the READ and WRITE requests (averaged over short intervals). The instrumentation we added is non-intrusive and relies on the data structures already present in the kernel. We believe that obtaining these metrics is quite easy for most of the storage architectures.

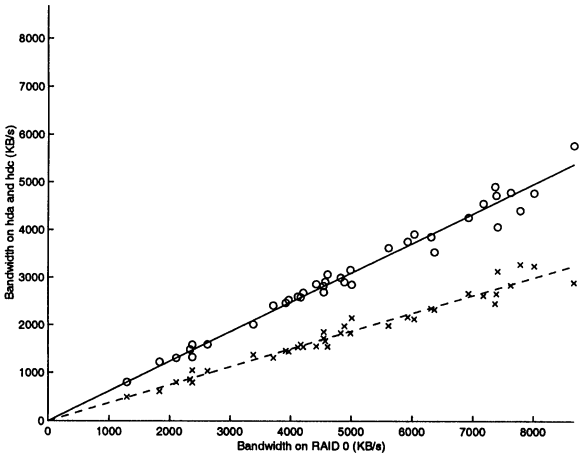


Figure 4. Dependency between logical volume RAID 0 and components HDA and HDC for READ operation. X axis represents bandwidth observed on logical volume RAID 0. Y axis denotes bandwidth observed on HDA (circles), and bandwidth observed on HDC (crosses). The solid line represents first-order linear regression model of the dependency between RAID 0 and HDA. Dashed line represents first-order linear regression model of the dependency between RAID 0 and HDC.

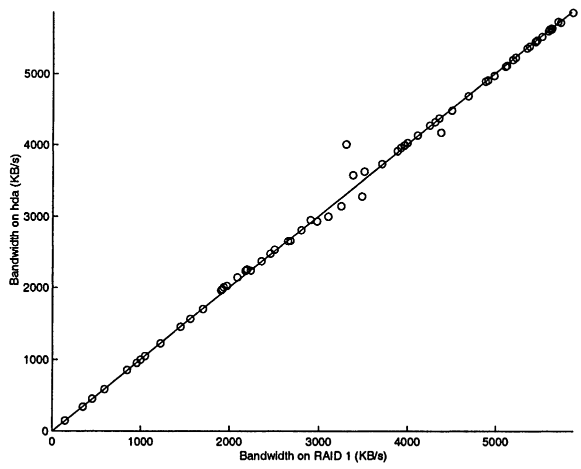


Figure 5. Dependency between logical volume RAID 1 and component HDA for WRITE operation. X axis represents bandwidth observed on logical volume RAID 1. Y axis denotes bandwidth observed on HDA (circles). The solid line represents first-order linear regression model of the dependency between RAID 1 and HDA.

Internal component	Logical Volume	
	RAID 0	RAID 1
HDA	0.62	0.65
HDA1	0.31	0.00
HDA2	0.31	0.00
HDA3	0.00	0.65
HDC	0.37	0.34
HDC1	0.37	0.00
HDC2	0.00	0.34

(a)

Internal component	Logical Volume	
	RAID 0	RAID 1
HDA	0.67	1.00
HDA1	0.33	0.00
HDA2	0.33	0.00
HDA3	0.00	1.00
HDC	0.33	0.99
HDC1	0.33	0.00
HDC2	0.00	0.99

(b)

Table 1. Estimation of dependency strengths for READ operation (a) and WRITE operation (b) obtained using our method.

The process of gathering the dependency information consisted of applying a controlled varying load to components of the system (one component at a time with remaining components being idle). Requests were randomly scattered over the volumes to minimize the effect of OS level caching. We developed an application that applied increasing load as long as all components of the system were under-utilized. First, we applied load to the RAID 0 logical volume. As a result we obtained samples of the rate and size dependency functions for various sizes and request rates arriving to RAID 0. The arrival rates and sizes of requests issued to all components as a result of the experiment were gathered. We used Matlab statistical toolbox to compute the first-order linear regression for the sample points representing bandwidth applied to logical volume and internal component. The resulting dependencies between RAID 0 and components *HDA* and *HDC* are depicted in Figure 3. Figure 4 shows the dependency between RAID 1 and component *HDA* for WRITE operations.

Similar sampling and regression modeling was performed for RAID 1 volume. The results are summarized in Table 1 (a). Values in the table denote the fraction of the bandwidth applied to the logical volume (columns) that were "passed on" to the internal components (rows of the table). For example, 0.37 of the amount of bandwidth observed on RAID 0 appeared on *HDC*. It may be seen, that the method precisely identified nearly equal split of work between three equal-sized partitions (*HDA1*, *HDA2*, and *HDC1*) caused by striping. Workload relayed to *HDA* was equally split among *HDA1* and *HDA2*. The load-balancing algorithm of Linux software RAID 1 showed bias toward the drive *HDA*. Results for WRITE requests and both logical volumes are presented in Table 1 (b). Again we can observe equal split of workload among all partitions participating in the RAID 0. As expected in the case of mirroring, full incoming workload of the RAID 1 volume was relayed to both partitions participating in the mirror. Presented results show that the method properly identified dependencies between components of the Linux software RAID.

4. Conclusions and Future work

This paper presents an empirical method for detecting and quantifying performance dependencies between components of the virtual storage system. We have proposed a flexible model and a set of metrics that may be used to quantify these dependencies. Use of the CIM standard enables our solution to be used in heterogeneous environment. We have described an empirical technique for quantifying the inter-component dependencies by applying controlled load to logical volumes exported by the system and monitoring the effect it has on the internal system components. We have proposed effective bandwidth as a measure of dependency and have showed an application of it in a Linux environment using software RAID.

The results reported in this paper are based on experiments conducted on a simple virtual storage system. One of the areas we are pursuing as future research work is to relate the dependency knowledge obtained at the storage layer to performance metrics measured at the application layer. In addition, our future research plans also involve investigation of the correspondence between workload applied to a logical volume V and utilization of internal system component C. It is possible to use the *dependency profiles* defined in this article and knowledge about service time of C to predict the utilization of C caused by the request stream applied to the logical volume V. Moreover, the total predicted utilization of the component C may be estimated as a sum of utilizations of all volumes that use C. This information will help a systems designer in the allocation of data objects with associated quality of service, such as data base table spaces, to virtual storage entities such a logical units.

References

[1] G. A. Alvarez, E. Borowsky, S. Go, T. H. Romer, R. Becker-Szendy, R. Golding, A. Merchant, M. Spasojevic, A. Veitch, and J. Wilkes. Minerva: An automated resource provisioning tool for large-scale storage systems. *ACM Transactions on Computer Systems*, 19(4):483–518, 2001.

[2] E. Anderson, M. Kallahalla, S. Spence, R. Swaminathan, and Q. Wang. Ergastulum: quickly finding near-optimal storage system designs. *HP Technical Report*, 2001.

[3] A. Brown, G. Kar, and A. Keller. An active approach to characterizing dynamic dependencies for problem determination in a distributed environment. *International Symposium on Itegrated Network Management*, May 2001.

[4] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson. Raid: High-performance, reliable secondary storage. *ACM Computing Surveys, Vol. 26 No. 2, pp. 145-185*, 1994.

[5] DMTF. Common information model specification. *http://www.dmtf.org*, June 1999.

[6] D. A. Patterson, G. A. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks (raid). *International Conference on Management of Data (SIGMOD)*, pages 109–116, 1988.

[7] J. Schott. Modeling storage. *DMTF System/Device Working Group, http://www.dmtf.org/*, 2001.

[8] M. Uysal, G. A. Alvarez, and A. Merchant. A modular, analytical throughput model for modern disk arrays. *In Proc. of the Ninth International Symposium on Modeling, Analysis and Simulation on Computer and Telecommunication Systems (MASCOTS)*, August 2001.

[9] J. Ward, M. O'Sullivan, T. Shahoumian, and J. Wilkes. Appia: Automatic storage area network fabric design. *In Conference on File and Storage Technologies (FAST), Monterey, CA.*, January 2002.

DESIGN AND IMPLEMENTATION OF A GENERIC SOFTWARE ARCHITECTURE FOR THE MANAGEMENT OF NEXT-GENERATION RESIDENTIAL SERVICES

Filip De Turck*, Stefaan Vanhastel, Koert Vlaeminck,
Bart Dhoedt, Piet Demeester
Department of Information Technology, Ghent University - IMEC
Sint-Pietersnieuwstraat 41, B-9000 Gent, Belgium.
Tel.: +32 9 267 35 87, Fax: +32 9 267 35 99
filip.deturck@intec.rug.ac.be

Filip Vandermeulen, Frederik De Backer, Francis Depuydt
Belgacom ANS/NIS Strategy Department
Koning Albert II-laan 27, B-1030 Brussel, Belgium.
Tel.: +32 2 202 10 80, Fax: +32 2 202 71 35
filip.vandermeulen@belgacom.be

Abstract: In this paper, we address the design of a generic architecture for the management of residential services. The architecture consists of components both at the customers' side as well as at the service provider's side. The key features of the architecture are service modularity, the concept of service sessions, service packaging and subscription. The architecture allows service providers and telecom operators to rapidly provide new integrated value-added services to their customers. Layer-based design ensures that the architecture is independent of the particular service and service realization technology. The architecture provides generic access session management, service session mangement, subscription management and billing. Its implementation is based on J2EE (Java 2 Enterprise Edition). The various components of the architecture will be discussed, together with the implementation issues.

Keywords: Service Management, Multimedia Home Services, Subscription Management, OSGi, J2EE

1. Introduction

Recently, there has been a growing interest shift from the provisioning of commodity services towards value-added residential services. Commodity services are mainly connectivity services such as internet access, leased lines and VPNs. The management of this type of services has been studied in extensive detail [1, 2] and various management platform solutions exist. However, the revenue for telecommunication network operators and network service providers from these commodity services is stagnating. As a consequence, residential services are gaining more and more interest.

*Postdoctoral Fellow of the Fund of Scientific Research - Flanders, Belgium (F.W.O.-V.)

Popular examples include Video on Demand (VoD), video-conferencing, e-gaming, virtual home environments and domotics systems. Due to the customer interaction, management of such residential services is more complex and critical than the management of commodity services. The design of efficient management solutions for such services has not yet been studied in enough detail. At the time of writing, different standards and systems are emerging for the delivery of residential services to customers. The following areas and industry segments in the service provisioning market can be distinguished:

(i) *Multi-party Multi-media Conferencing Systems and Applications*: traditional telecommunications equipment vendors are introducing the concept of soft switches which take over the signalling and transport functionality of traditional TDM-based PSTN networks. Soft-switches control the voice gateways at the edge of an IP based packet transport network, and translate traditional PSTN signalling into other signalling protocols (SIP [3], H323 [4], MGCP [5], Megaco [6], etc.) which are more aligned with the IP network. Soft-switches can also support these new forms of signalling directly to the consumer's appliance endpoints, such as Intelligent Access Devices (IADs) or IP telephones (either standalone or software based IP telephony clients on a PC platform such as Microsoft NetMeeting). By means of standardized and Java based APIs such as those defined by Sun's JAIN framework [7], these protocols can be controlled and handled from within J2EE-based [8] application servers.

(ii) *The Connected Home and Home Gateways*: the concept of the connected home implies a household, which is connected to the outside world via a broadband gateway terminal (i.e. a home gateway). The home gateway acts as some kind of internal driver and application management system, which is able to perform driver life-cycle management and configuration management of in-house devices and appliances. In addition, the home gateway can act as repository for content downloaded from the Internet or from network server-side service and content delivery platforms. This temporary locally-proxied content can be consumed within the house from different rooms and consumer devices (such as TV sets, PCs, PDAs, etc.), by connecting them to a multi-room multi-source system.

(iii) *Home Theatre Systems*: home theatre systems are currently entering the residential home at an increasing speed, as high-end plasma screens and projector-based systems are gradually becoming affordable for households. In combination however with network side Video-on-Demand (VoD) server platforms, home theatre systems could be used to view content streamed in real-time (e.g. via RTP). Another area where the home theatre system could be used as multimedia endpoint consumption system is online gaming.

(iv) *The Intelligent Home*: Domotics is yet another emerging industry area with increasing potential to bring added value to the home. Domotics systems allow to control systems such as lighting, heating, home surveillance, etc. More and more vendors of domotics systems provide touch-panel !ike controls on their system, but some of them go even further by providing IP modules that connects to the bus or the central computer of the system. Via the home gateway, the domotics system can be connected to the outside world. Through the IP modules, the domotics system can be controlled and interfaced from anywhere. For example, events in the home could trigger the setup of a video connection from an in-house video camera to a PDA.

In each of these areas, different vendors come up with different systems and platforms that in most cases are implemented without any adherence to a standard – in so far any standard exists. None of the above areas and applications will unlock massive revenue streams in the residential market, since a key element is clearly missing in the above description. That missing element is the enabling platform that allows rendering the set of isolated potential solutions and platforms into one integrated package.
This paper describes the design and implementation of a platform which supports (i) integration of several existing stand-alone service components, (ii) generic subscription management to all services, (iii) generic access session management, and (iv) related billing. The platform consists of components both at the service provider's side as well as at the customers' side. The platform implementation takes benefit of the most recent enabling software and middleware technologies: J2EE (Java 2 Enterprise Edition) based application servers, EJBs (Enterprise Java Beans) in the backend layer, servlets in the control layer, and JSPs (Java Server Pages) in the view layer. As such, the Model-View-Control (MVC) design pattern is exploited in its most generic way. XML-RPC [9] and RMI/IIOP [10] are used as enabling middleware protocols for communication between components in the subscribers' end-terminals and the platform components. The use of J2EE for building service provisioning systems has already been reported upon in [11]. However, service subscription, access session and interaction with software components at the customers' side were not addressed in [11]. The IETF OPES (Open Pluggable Edge Services: [12]) Working group also aims at designing architectures to allow for service provisioning, but doesn't provide service providers with a platform to address all their needs.
The remainder of this paper is structured as follows: section 2 describes the key features of the platform. The platform's software architecture is introduced in section 3, whereas the μ-flow concept is described in section 4. The main components of the architecture are described in section 5. The scenario of a service access session is detailed in section 6. The database model is covered in section 7 and the gateway components are addressed in section 8. Finally, section 9 sums up the conclusions that can be drawn from this study and the important issues for further work.

2. Key Features of the Architecture

The main characteristics of the implemented service management architecture are: (i) service modularity, (ii) the concept of service sessions and (iii) service packaging and subscription. Each of these characteristics is described in more detail below:

(i) *Service Modularity*: The platform allows the subscription to, and consumption of value added services according to a session driven model. In order to cope with the heterogeneous base of multimedia and entertainment delivery and processing platforms, the architecture is built around the concept of Pluggable Service Driver Modules (PSDMs). A PSDM is a plug-in component, which controls, manages and interfaces underlying delivery platforms. Examples include a VoD delivery platform consisting of video servers spread over the country, or an e-gaming service delivery platform. Another example is a network of soft-switches, which can handle the setup and delivery of IP-Telephony calls or Video-Telephony calls. For each of these service technologies and in particular for each vendor specific implementation of a service, a PSDM can be provided which plugs into the service architecture. These PSDMs export well-known control interfaces within the platform and act as drivers to the underlying service delivery platforms.

(ii) *Service Session Concept*: Every kind of service consumption takes place within the context of sessions. When a user accesses the application server implementing the service architecture, he/she starts and enters the context of an access session. This access session establishes a secure context with the service architecture. Within the access session, the end-user is authenticated. From within this access session, which is materialized by a server side access session user agent component, the end-user can start a service session. Within this service session the end-user can execute the logic of the service. The service session is materialized by a PSDM session instance within the application server.

(iii) *Service Packaging and Subscription*: The platform enables the easy packaging of services, applications and features into product packages to which a customer or a subscriber can subscribe online. Even for services for which new appliances are needed, the stakeholder who exploits the platform (e.g. the telecommunication operator or service provider) will deliver and install these appliances as a result of an online subscription. Subscription is at the heart of unlocking revenue streams. Combined with the concept of customer zero-concern with getting the right hardware and software environment at home for consuming certain services, subscription triggers certain company processes in order to install the customer (download software to the home gateway in a secure way or ship certain hardware appliances which the customer connects and which get configured remotely).
When an end-user starts a certain service, the service architecture checks whether the user is authorized to launch the service session according to his actual subscription profile. If this is the case, the service is launched via the appropriate PSDM module. As a result of this service consumption session, a trigger/message can be sent to the billing systems to start billing the service on whatever basis: time interval during which the service is used, type of service, exchanged traffic volume, etc.

The next section provides an overview of the service management platform.

3. Platform Overview

The platform is implemented on a J2EE based application server. The implementation consists of four levels, which are positioned according to the fragments of the Model-View-Control (MVC) design pattern:

(1) A *backend resource layer*: This layer consists of the database(s) and any underlying service delivery platform, which is under control of the platform.

(2) An *EJB backend layer*: This layer consists of a number of deployed Enterprise Java Beans (EJBs) that abstract the resources from the underlying backend resource layer. These EJBs are also referred to as the Pluggable Service Driver Modules (PSDMs). Amongst others, the following EJBs have been deployed:

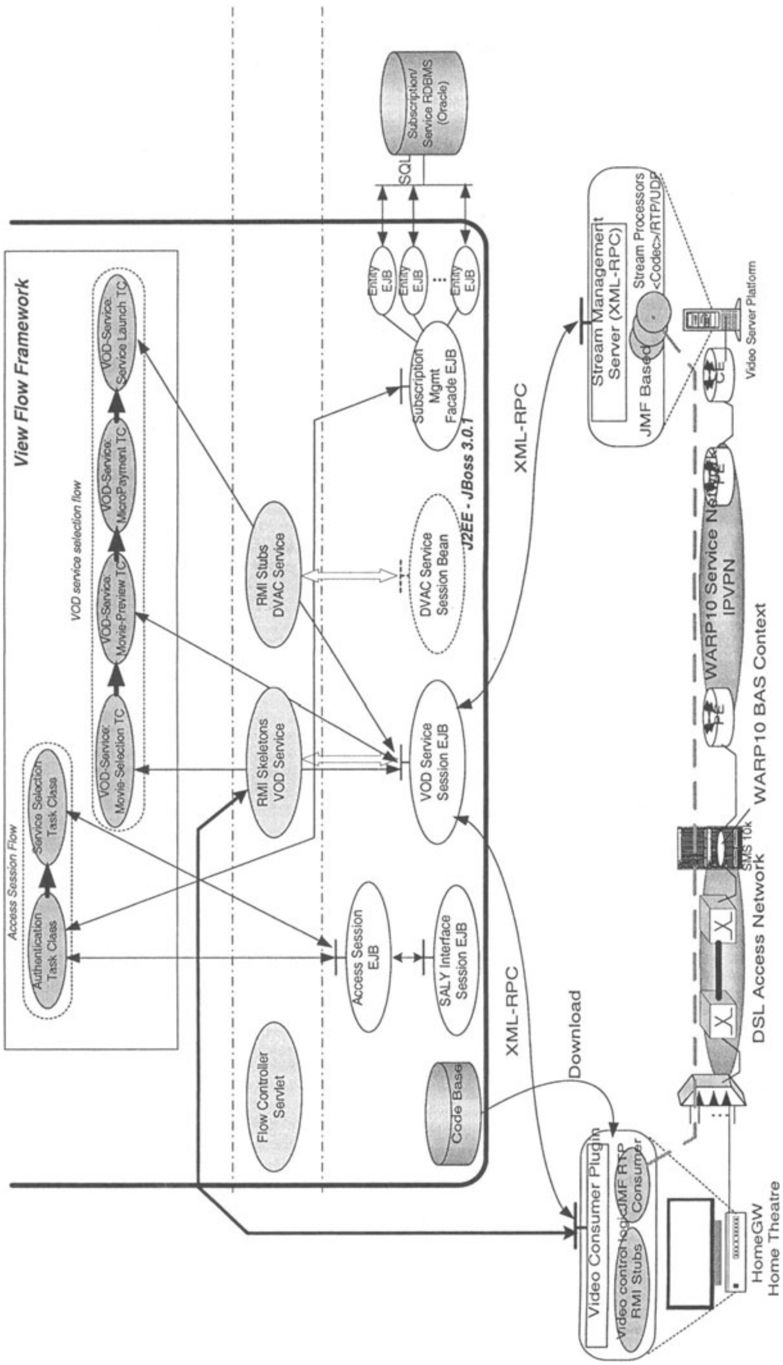

View Flow Framework
Access Session Flow
Authentication Task Class
Service Selection Task Class
VOD service selection flow
VOD-Service: Movie-Selection TC
VOD-Service: Movie-Preview TC
VOD-Service: MicroPayment TC
VOD-Service: Service Launch TC
Flow Controller Servlet
RMI Skeletons VOD Service
RMI Stubs DVAC Service
Access Session EJB
SALY Interface Session EJB
VOD Service Session EJB
DVAC Service Session Bean
Subscription Mgmt Facade EJB
Entity EJB
Entity EJB
Entity EJB
J2EE - JBoss 3.0.1
Code Base
SQL
Subscription/ Service RDBMS (Oracle)
XML-RPC
XML-RPC
Download
Stream Management Server (XML-RPC)
JMF Based
Stream Processors <Codec>/RTP/UDP
Video Server Platform
CE
PE
PE
WARP10 Service Network IPVPN
WARP10 BAS Context
SMS 10k
DSL Access Network
Video Consumer Plugin
Video control logic RMI Stubs
JMF RTP Consumer
HomeGW
Home Theatre

Figure 1. High-level computational decomposition of the implementation: setup of a VoD service session.

(a) An Access Session Bean;

(b) A Subscription Management Facade Session Bean together with a set of subscription level RDMS entity beans;

(c) A Service Control Bean;

(d) A VoD Service Session Bean;

(e) An on-Line Gaming Session Bean;

(f) A Digital Video and Audio Conferencing Bean (DVAC Bean);

Each of these beans will be described in the next section.

(3) The *control layer*: This layer contains a number of servlets which execute service management control by invoking business functions exported by the EJBs of the underlying EJB backend layer.

(4) The *View framework layer*: At this layer, the actual components that materialize the application views to the end-users are located. The view framework is developed around the paradigm of a μ-flow-processing framework. Every logical functionality a user can invoke via the service portal is implemented through and supported by a μ-flow. This μ-flow consists of a number of steps, each implemented in an action task class. An action task class implements an atomic piece of functionality in a generic and reusable way (reusable in other μ-flows). The action class is the place where actual calls are made to backend EJBs. A flow description is read and parsed by a controller servlet which subsequently instantiates the required set of action classes. Each action class is linked to a JSP page which requests the end-user at the browser front-end for the required input, and which returns the appropriate outputs. Every JSP page contains a link to the logically next page, which invokes an operation on the next task class according to the logic of the flow. For every logical action on the service portal front-end, a μ-flow is defined: a flow for setting up an access session, a flow for starting a VoD service session, a flow for taking a new service subscription, etc.

Figure 1 shows the High-level computational decomposition of the platform for the setup of a sample Video on Demand (VoD) service. The authentication and service selection flow is shown, which are implemented as Java Server Pages. These invoke operations on the underlying servlets and Enterprise Java Beans (EJBs). The VoD Service Session EJB is responsible for the configuration of the involved Video Servers and video consumer plugins. For a video session with garantueed bandwidth, an dedicated connection can be realized in the underlying DSL/ATM network (via automatic invocation of the connection management platform).

4. μ-flow Model

The view framework uses the concept of modular configurable μ-flows, which implement the logic of a stepwise execution process with a wizard like presentation style in the browser for the end user. The view framework consists of the following components:

(1) *Description of a μ-flow in XML format*: a μ-flow consists of a number of steps. Within each step, there is indication of (i) the name of a Java task class, which

```
<?xml version="1.0" encoding="UTF-8" ?>
<statemachine>
  <initialize>
    <startflow>1</startflow>
    <startstate>1</startstate>
    <security>1</security>
  </initialize>
  <flows>
   <flow id="1" type="task">
    <states>
     <state>
      <id>1</id>
      <view>/tasks/LoginTask.jsp</view>
      <template>0</template>
      <controllerclass>com.belgacom.warpportal.
                      task.LoginTask</controllerclass>
      <errorstate>1</errorstate>
      <events>
       <event>
        <id>1</id>
         <actionflow>1</actionflow>
         <actionstate>2</actionstate>
           <actiontype>forwardunload</actiontype>
       </event>
      </events>
     </state>
```

```
    <state>
     <id>2</id>
      <view>/tasks/ServiceSelect.jsp</view>
      <template>0</template>
       <controllerclass>com.belgacom.warpportal.
              task.ServiceSelect</controllerclass>
      <errorstate>1</errorstate>
     <events>
       <event>
         <id>1</id>
          <actionflow>1</actionflow>
          <actionstate>3</actionstate>
          <actiontype>forwardunload</actiontype>
        </event>
       <event>
          <id>2</id>
          <actionflow>1</actionflow>
          <actionstate>4</actionstate>
          <actiontype>forwardunload</actiontype>
        </event>

                 ...
   </flow>
  </flows>
</statemachine>
```

Figure 2. Sample XML fragment describing a μ-flow.

executes the logic of the step, (ii) the JSP page to be used for querying the input and presenting the output of the step, and (iii) the next steps/states to be taken in the flow based on possible outcomes of the executed logic in the step. It is possible to refer within a particular flow description to another type of flow. In other words, it is possible to jump from a step in a flow of type X to a step described in a flow of type Y. As such, flows described as a demarcation of a number of steps belonging together in a logical way can be reused within other flows. A sample μ-flow description is shown in the XML fragment in Figure 2.

(2) *A μ-flow controller servlet*: The controller servlet parses on system startup the XML code of the different flows to be supported, and instantiates within a logical scope of each flow a tree of task classes pertaining to each step/state in the flow.

(3) *Task classes*: Each step in a flow is materialized by a task class which implements the TaskHandler Java interface. The most important method of a task class is the execute() method. This method pushes the appropriate JSP/HTML page to the web front-end in order to query for any user input. Next the task performs the required logic by interfacing with the system backend layer, and returns the appropriate output back to the front-end.

(4) *Memory Handler class*: Each flow instance disposes of a memory handler object which maintains the data belonging to and determining intermediate states within a flow. Within the same flow instance all that state data is maintained on the memory handler object which acts as some kind of data clipboard associated to the servlet session. By default, the memory object is cleared when a jump is made from a state in one flow to a state in another flow, however, this default behavior can be changed by configuration in the flow's XML file.

5. Component Description

Access Session Bean. The Access Session Bean is invoked by classes in the access session μ-flow. The first method invoked on the Access Session Bean is `loginOnAccessSession`. Subsequently, the Access Session Bean invokes a method in the Subscription Utility Class (SU) in order to check the subscriber's existence. Directly after authentication, the IP address assigned to the end terminal or home gateway of the end-user is determined. It is initially supposed that the subscriber connects from his home environment via the DSL access network. When the subscriber has been successfully authenticated, the access session flow can take a number of directions depending on the user's preference. The following functionalities can be chosen: (i) service selection and setup, (ii) on-line subscription management, and (iii) identification management. Each of these functionalities will be detailed in section 6.
Figure 3 shows the generic EJB interface operations of the Access Session Bean.

Subscription Management Facade Session Bean. The Subscription Management Facade Session Bean allows the control of the subscription management. A distinction is made between a `customer`, a `subscriber` and a `subscribergroup`. Each `subscribergroup` has an associated Service Access Group (SAG), which contains the different services, the `subscribergroup` is subscribed to. The internal database model will be further detailed in section 7.
Figure 4 shows the generic EJB interface of the Subscription Management Control EJB. Only the creation operations are shown. Obviously, the interface also offers deletion and modification operations.

Service Control Bean. This bean allows the generic addition, modification and deletion of services and the associated service parameters. The EJB Service Control interface is shown in Figure 5.

```
interface AccessSession extends EJBObject {

        public String loginOnAccessSession(
                String loginName,
                String loginPinCode,
                String password
        ) throws AccessSessionLoginError,
                        RemoteException;

        public Collection getAllServiceTypes(
        ) throws AccessSessionInfoError,
                RemoteException;

        public Collection getSubscribedServices(
        ) throws AccessSessionInfoError,
                RemoteException;

        public EJBObject setupServiceSession(
                String serviceName
        ) throws
                AccessSessionActionError,
                   RemoteException;

        public SubscriptionMgmtControl
                setupSubscriptionMgmtSession(
        ) throws AccessSessionActionError,
                RemoteException;

        public void changeLoginCredentials(
                String loginName,
                String password
        ) throws AccessSessionInfoError,
                RemoteException;

        public void changeIdentification(
                SubscriberValObj subscriberInfo
        ) throws AccessSessionInfoError,
                RemoteException;
}
```

Figure 3. The Access Service Session EJB interface.

```
public interface SubscriptionMgmtControl extends
EJBObject {
        // Customer operations
        public Long createCustomer(
                CustomerValObj customer)
        throws CustomerAlreadyExists,
               InvalidObjectAttributes,
               ManagementError,
               RemoteException;

        // Subscriber group operations
        public Long createSubscriberGroup(
            String customerRefNr,
            SubscriberGroupValObj subscriberGroup)
        throws CustomerNotInExistance,
               SubscriberGroupAlreadyExists,
               InvalidObjectAttributes,
               ManagementError,
               RemoteException;

        // Subscriber operations
        public Long createSubscriber(
                String customerRefNr,
                SubscriberValObj subscriber)
        throws CustomerNotInExistance,
               SubscriberAlreadyExists,
               InvalidObjectAttributes,
               ManagementError,
               RemoteException;

        // Service SAG operations
        public Long createSag(
                String customerRefNr,
                String groupName,
                SagValObj sag)
        throws CustomerNotInExistance,
               SubscriberGroupNotInExistance,
               SagAlreadyAssigned,
               InvalidObjectAttributes,
               ManagementError,
               RemoteException;

        // Service SAG operations
        public Long addServiceToSag(
               String customerRefNr,
               String groupName,
               String sagName,
               SagServiceValObj sagService,
               Collection sagServiceParams)
        throws CustomerNotInExistance,
               SubscriberGroupNotInExistance,
               SagNotInExistance,
               InvalidObjectAttributes,
               ServiceAlreadyInSag,
               InvalidObjectAttributes,
               ManagementError,
               RemoteException;
}
```

Figure 4. The Subscription Management Control EJB interface.

Service Session Bean. Different specific Service Session Beans have been developed: (i) a VoD service session bean, (ii) an on-line gaming session bean and (iii) a digital video and audioconferencing bean. Other types of beans can easily be integrated with the described platform.

6. Access Session Scenario

When an end-user logs in to the service portal, the access session μ-flow's first task class instantiates an Access Session Bean, of which the reference is put on the HTTP session context object of the user's HTTP session (created and managed by the view framework controller servlet). After login, the IP address of the end terminal or home gateway of the end-user is determined and the following choices are presented to the customer:

- **Choice 1: Service selection and setup** The user is guided via a number of subsequent task classes through the selection and setup of a service.
- **Choice 2: On-Line subscription management** The user enters subscription management mode and can take a new subscription to a set of services and service packages, or can change and adapt the detailed characteristics of an existing subscription. Note that in this case the subscriber should have the authorisation to adapt subscription assignment groups and services within subscription assignment groups.

```
interface ServiceControl extends EJBObject {

    public Long addService(
            ServiceValObj service,
    Collection serviceParameters)
    throws ServiceAlreadyExists,
   InvalidObjectAttributes,
        ManagementError,
        RemoteException;

    public void modifyService(
            String currentServiceName,
            ServiceValObj service)
    throws ServiceAlreadyExists,
            ServiceNotInExistence,
            InvalidObjectAttributes,
            ManagementError,
            RemoteException;

    public void deleteService(
            String serviceName)
    throws ServiceNotInExistence,
        LinkedObjectsError,
        ManagementError,
        RemoteException;

    public Collection addParametersToService(
            String serviceName,
            Collection serviceParameters)
    throws ServiceNotInExistence,
        ParameterInExistenceError,
        ManagementError,
        RemoteException;

    public void modifyServiceParameters(
            String serviceName,
            Collection serviceParameters)
    throws ServiceNotInExistence,
        ParameterInExistenceError,
        ManagementError,
        RemoteException;

    public void deleteServiceParameter(
            String serviceName,
            String parameterName)
    throws ServiceNotInExistence,
        ParameterInExistenceError,
        LinkedObjectsError,
        ManagementError,
        RemoteException;
}
```

Figure 5. The Service Control EJB interface.

- **Choice 3: Identification management** The end-user enters a mode where personal identification parameters can be altered, e.g. : login names, passwords, address information, phone numbers, etc.

Each of these choices is detailed in the subsequent paragraphs.

Choice 1: Service Selection and Setup.

(1) In case of service selection, the flow passes via a service selection task class, which calls the `getSubscribedServices` method of the Access Session Bean. This method returns all the service types and corresponding service parameter ranges for which the subscriber has the right to launch service session instances.

(2) Subsequently, the subscriber selects a service and indicates he/she wants to start a service session instance. For this purpose, the access session flow will invoke for this purpose the `setupServiceSession` operation and passes the name of the service type. As a result, the access session bean will start up a service session bean of the right type and places the obtained reference on the HTTP Context Session of the subscriber's HTTP/servlet session. The access session flow stops here, and hands over the control to a service specific flow, which then handles the steps specific to the setup of the selected service.

Choice 2: On-Line Subscription Management.

(1) In case of on-line subscription, the access session flow delegates further control to the subscription management flow, which starts the subscription management feature selection class. This class starts a subscription management session through the operation `setupSubscriptionMgmtSession`. This operation

instantiates a subscription management session bean and returns its reference. This reference is put on the HTTP/Servlet Session Context object.

(2) In a subsequent flow task class, the subscriber can select one of the basic features of subscription management which are:

   (a) Addition/Deletion of service subscriptions to the portfolio of one of the customer's subscription assignment groups;
   (b) Modification of a service subscription within a subscriber group's SAG; Amongst others:
      - Change of subscription periods and intervals;
      - Change of service quality degrees and profiles;
      - Adaptation of basic service parameters;
   (c) Addition of new subscribers to one or more subscriber groups;
   (d) Modification of properties pertaining to the customer identity or pertaining to the subscriber group(s) created within the customer's scope;
   (e) Granting on-line subscription management authorisation at different levels to subscribers belonging to the customer's scope.

(3) Once the desired subscription management feature has been selected, the subscription management flow passes via the actual feature execution task, which invokes the subscription management bean created in the previous step. Its logic is executed through aid of the subscription management utility class.

Choice 3: Identification Management.

(1) In case of on-line identification management, the access session flow passes the action task class allowing identification change management feature selection. Two basic features are exported:

   - Modification of login credentials: login name and password. The login pincode has been assigned to the involved subscriber at the time of initial off-line subscription of the initial set of subscribers, or at on-line subscription when an authorized subscriber created the account for the involved subscriber. This pincode cannot be changed.
   - Modification of detailed affiliation data pertaining to the subscriber: address info, phone numbers, aliases, etc.

(2) After selection of the desired feature, the flow passes the feature execution task class, which on its turn invokes either `changeLoginCredentials` or `changeIdentification` dependently.

7. Database Model

The database's internal structure is detailed in Figure 6, which shows the scheme of the involved records. Every user of the system is described as a `subscriber`. If a user does not have a subscriber object configured in the system for which he knows the login name, password and pincode, that user cannot be authenticated to the service platform. There are two types of subscribers: `residential subscribers` and `corporate subscribers`. A subscriber object can only be of one type. A residential subscriber is acting independently and has full control over the services with the associated parameters he/she subscribes to. A corporate subscriber is always linked to a larger affiliation, which we represent in the inventory as a `customer`. Under the authority of a customer, a number of `subscriber groups` can be created. These subscriber groups contain one or more corporate subscribers. A subscriber group represents a logical group of users who have the same subscription contract, i.e. who have the same set of services (and service customizations) to which they are subscribed. A corporate subscriber is made member of a subscriber group through a `subscriber membership` object. As such a corporate subscriber can be part of multiple subscriber groups. A subscription to one or more services is always represented by a `subscription assignment group` or a `SAG`. A SAG contains one or multiple `sagservice` objects. A `sagservice` is a customization object of a corresponding singleton service object. This service object has a number of associated service parameter objects. These service parameters represent the customizable properties of a service together with maximum/minimum values or a set of allowed values or strings. For example, a video conferencing service has as property the number of parties that can be connected together in a session. The minimum value will be 0 and the maximum value could be 10. A sagservice object is a customization of a service object. A SAG is connected to either a subscriber group or directly to a residential subscriber. In case it is connected to a subscriber group, all corporate subscribers belonging to the group have the subscription as described by the SAG. In case it is connected to a residential subscriber, the SAG describes the subscription contract of only that residential subscriber. Finally, there are a number of auxiliary tables which are service specific. In the figure, the `VODSERVER` and `MOVIE` tables are specific for the VoD service implementation.

8. Home Gateway

The terminal architecture, also referred to as the home gateway, from which end-users consume multimedia and domestic services, is designed on a Linux based platform. The home gateway is designed according to emerging OSGi (Open Service Gateway initiative: [13]) concepts. A JVM based OSGi control kernel and corresponding bootstrapping software will be automatically downloaded from the platform on initial powering and sign-on of the gateway. The service management platform takes all responsibilities to download, remotely configure, and manage the Java based service consumption bundles on top of the JVM based OSGi execution kernel. Remote version management of these software bundles is under control of the platform.

9. Conclusion and Further Work

A generic architecture has been designed for the integration of stand alone service components, be it off-the shelf or developed in-house, through Pluggable Service Driver Modules (PSDMs). PSDMs act as service abstracting computational components that wrap and abstract the technology and vendor specific service logic within the generic service delivery architecture. The architecture has been implemented by making use of J2EE technology.
Examples of service technologies integrated in the platform are video conferencing control systems (e.g.soft-switches enabling SIP based video telephony, VoD platforms,

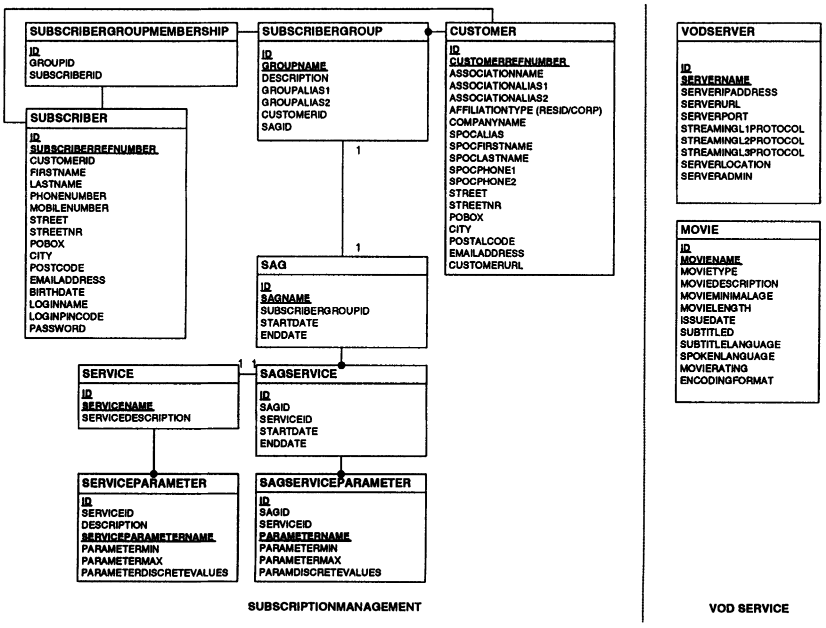


Figure 6. The internal database structure.

e-gaming control platforms, domotics systems at home, etc.) By means of an extensible internal RDBMS architecture, the platform will allow extensible on-line subscription management of customers, customer subscriber groups, and subscribers. The whole process of setting up an access session from the home with the platform, selecting a next generation service, consuming it and billing it within the controlled and managed scope of a single/multi-party, multimedia session, is handled according to a flow-driven model in a transactional execution context.

Important issues for further work include: (i) a thorough performance evaluation of the platform, (ii) application of load balancing techniques for the distribution of the platform load and (iii) development and thorough testing of several other types of service control beans, especially for the control of services delivered to wireless devices and (iv) designing Terminal QoS matching algorithms for deciding the best terminal parameters for conversational services between users with heterogeneous terminal equipment. The design of these QoS matching algorithms will be based on the work, presented in [14].

Acknowledgments

The first author wishes to thank his colleagues at the IBCN (INTEC Broadband Communication Networks) research group, Ghent University for the stimulating discussions.

References

[1] F. De Turck, F. Vandermeulen, P. Demeester, *On the design and implementation of a hierarchical, generic, scalable open architecture for the network management layer*, IEEE/IFIP Network Operations and Management Symposium, pp.745-758, Honolulu, April 2000.

[2] F. De Turck, S. Vanhastel, F. Vandermeulen, P. Demeester, *Design and implementation of a Generic Connection Management and Service Level Agreement Monitoring Platform Supporting the Virtual Private Network Service*, IFIP/IEEE IM 2001 conference, Seattle, May 2001, pp. 153-166.

[3] M. Handley, H. Schulzrinne, E. Schooler, J. Rosenberg, *SIP: Session Initiation Protocol*, IETF RFC 2543, March 1999.

[4] ITU H.323 Standard: *Packet-based multimedia communications systems*

[5] M. Arango, A. Dugan, I. Elliott, C. Huitema, S. Pickett, *Media gateway control protocol (MGCP)*, IETF RFC 2705, October 1999.

[6] Media Gateway Control (megaco) Charter, *www.ietf.org/html.charters/megaco-charter.html*

[7] Java API for Intelligent Networks (JAIN), *http://java.sun.com/products/jain/api_specs.html.*

[8] Java(TM) 2 Platform, Enterprise Edition (J2EE), *java.sun.com/j2ee/*

[9] XML-RPC Specification, *http://www.xmlrpc.com/spec*

[10] SUN Microsystems, *Java Remote Method Invocation Specification*, 1998, *http://java.sun.com/products/jdk/1.1/docs/guide/rmi/spec/rmiTOC.doc.html*

[11] S. Sengal, J.W. Gish, J.F. Tremlett, *Building A Service Provisioning System using the Entreprise Java Bean Framework*, Network Operations and Management Symposium, 2000, pp. 367-380.

[12] IETF Open Pluggable Edge Services (opes) Working Group, *http://www.ietf.org/html.charters/opes-charter.html*

[13] Open Service Gateway initiative (OSGi) - Specification Overview, *http://www.osgi.org/resources/spec_overview.asp*

[14] F. Vandermeulen *et al*, "*A Multimedia Terminal Architecture for Dynamically Configurable Protocol Stacks*", ICME 2000 conference, New York, August 2000.

SESSION 12

Fault Management

Chair: Joseph Hellerstein
IBM Research, USA

USING NEURAL NETWORKS TO IDENTIFY CONTROL AND MANAGEMENT PLANE POISON MESSAGES

XIAOJIANG DU, MARK A. SHAYMAN, RONALD SKOOG
Department of Electrical and Computer Eng. University of Maryland, Telcordia Technologies

Abstract: Poison message failure propagation is a mechanism that has been responsible for large scale failures in both telecommunications and IP networks: Some or all of the network elements have a software or protocol 'bug' that is activated on receipt of a certain network control/management message (the poison message). This activated 'bug' will cause the node to fail with some probability. If the network control or management is such that this message is persistently passed among the network nodes, and if the node failure probability is sufficiently high, large-scale instability can result. Identifying the responsible message type can permit filters to be configured to block poison message propagation, thereby preventing instability. Since message types have distinctive modes of propagation, the node failure pattern can provide valuable information to help identify the culprit message type. Through extensive simulations, we show that artificial neural networks are effective in isolating the responsible message type.

Key words: Poison message, neural network, node failure pattern, fault management

1. INTRODUCTION

There have been a number of incidents where commercial data and telecommunications networks have collapsed due to their entering an unstable mode of operation. The events were caused by unintentional triggers activating underlying system defects (e.g., software 'bugs,' design flaws, etc.) that create the propagation mechanism for instability. These system defects are generally not known to the network providers, and new defects are constantly introduced. More importantly, these points of vulnerability can be easily triggered through malicious attack.

Our goal is to develop a fault management framework that can protect networks from unstable behavior when the trigger mechanism and defect causing instability are unknown. This can be accomplished by identifying *generic failure propagation mechanisms* that permit failures to lead to network instability. Once these mechanisms are identified, control techniques can be designed to prevent or interrupt the failure propagation. There are several failure propagation mechanisms that can cause unstable network. Five generic propagation mechanisms have been identified thus far [4]:

- System failure propagation via routing updates;
- System failure propagation from management / control plane 'poison message';
- System failure propagation from data plane 'invalid message';
- Congestion propagation from congestion back pressure;
- Deadlocks created from overload and timeouts.

Here we focus on one of these that we call the 'poison message' failure propagation mechanism.

1.1 The Poison Message Failure Propagation Problem

A trigger event causes a particular network control/management message (the poison message) to be sent to other network elements. Some or all of the network elements have a software or protocol 'bug' that is activated on receipt of the poison message. This activated 'bug' will cause the node to fail with some probability. If the network control or management is such that this message is persistently passed among the network nodes, and if the node failure probability is sufficiently high, large-scale instability can result. Several such incidents have occurred in telecommunication and other networks, such as an AT&T telephone switching network incident in 1990 [8]. We are also aware of an incident in which malformed OSPF packets functioned as poison messages and caused failure of the routers in an entire routing area for an Internet Service Provider.

In the AT&T incident above, the trigger event is the normal maintenance event that caused the first switch to take itself out of service. The poison message is the ordinary message sent to neighboring switches informing them that it is temporarily going out of service. The poison message creates a state in the neighboring switches in which faulty code may be executed. In this case, an auxiliary event must occur for the faulty code to be executed causing the node to fail, namely the arrival of a pair of closely spaced call setup messages. The dependence on the auxiliary event makes the node failure probabilistic with the probability depending on network load (the rate of call setup messages). While in this particular example, the poison message itself is not flawed, in other examples such as the OSPF case referred to above, the poison message may be malformed or contain fields with abnormal values.

Obviously, the more challenging case is the one in which the message itself is completely normal and is 'poison' only because of a software defect in the router or switch.

There are some discussions about the poison message problem in [8]. One idea to fight the poison message problem is to limit the number of similar type Network Elements managed by the central control function and use a distributed architecture. But the above idea is not practical [8]. It might cause lots of changes to the existing systems. And it also causes software, hardware inconsistency in the system.

Our philosophy is to add some functions to the existing system rather than change it. We want to design a fault management framework that can identify the message type, or at least the protocol, carrying the poison message, and block the propagation of the poison message until the network is stabilized. We propose using *passive diagnosis* and *active diagnosis* to identify and block the corresponding protocol or message type.

1.2 The Problem Features

This problem has several differences from traditional network fault management problems. Typical network fault management deals with localized failure [5] [7]. For instance, when there is something wrong with a switch, what propagates is not the failure but the consequences of the failure on the data plane (e.g., congestion builds up at upstream nodes). Then multiple alarms are generated that need to be correlated to find the root cause [3] [9]. In our problem, the failure itself propagates, and propagation occurs through messages associated with particular control plane or management plane protocols. It is also different from worms or viruses in that worms and viruses propagate at the application layer.

A message type may have a characteristic pattern of propagation. For example, OSPF uses flooding so a poison message carried by OSPF link state advertisements is passed to all neighbors. In contrast, RSVP path messages follow shortest paths so a poison message carried by RSVP is passed to a sequence of routers along such a path. Consequently, we expect pattern recognition techniques to be useful in helping to infer the responsible message type.

We make the following three assumptions:

1) There is a central controller and a central observer in the network. I.e., we use centralized network management.

2) The recent communication history (messages exchanged) of each node in a communication network can be recorded.

3) Because the probability of two message types carrying poison messages at the same time is very small, we assume there is only one poison message type when such failure occurs.

In this problem, we use centralized network management scheme. This is due to the nature of the problem. The failure itself propagates in the whole network. In order to identify the responsible protocol, event correlation, failure pattern recognition and other techniques are used. They all need global information, correlation and coordination. We also suggest some distributed methods to deal with the poison message problem, such as the Finite State Machine (FSM) method. The FSM in our framework is a distributed method in the sense that it is applied separately to information collected at individual failed nodes, rather than across multiple failed nodes. The details are given in section 2.

There are several ways to record the communication history. Here we assume we can put a link box at each link of the network. The link box can be used to record messages exchanged recently in the link. The link box can also be configured to block certain message types or all messages belonging to a protocol. We refer to the blocking as message or protocol filtering. Filtering may be used to isolate the responsible protocol by blocking one or more message types and observing whether or not failure propagation continues. The filter settings may or may not be chosen to be the same throughout the network.

We suggest combining both passive diagnosis and active diagnosis to find out the poison message. Passive diagnosis includes analyzing protocol events at an individual failed node, correlating protocol events across multiple failed nodes, and classifying the observed pattern of failure propagation. Active diagnosis uses protocol or message type filtering.

2. PASSIVE DIAGNOSIS

Passive diagnosis includes the following methods.

1) Finite State Machine Method: This is a distributed method used at a single failed node. All communication protocols can be modeled as finite state machines [3] [11]. When a node fails, the neighbor of the failed node will retrieve messages belonging to the failed node. From the message sequence for each protocol, we can determine what state a protocol was in immediately prior to failure by checking the FSM model. We can also find out whether those messages match (are consistent with) the FSM. If there are one or more mismatches between the messages and the FSM, that probably means there is something wrong in the protocol.

2) Correlating Messages: Event correlation is an important technique in fault management. Recently exchanged messages are stored prior to node failure. Then we analyze the stored messages from multiple failed nodes. If multiple nodes are failed by the same poison message, there must be some common features in the stored messages. One can compare the stored messages from those failed nodes. If for a protocol, there are no common received messages among the failed nodes, then we can probably rule out this protocol--i.e., this protocol is not responsible for the poison message. On the other hand, if many failed nodes have the same final message in one protocol, we can use Bayes' Rule to calculate the probability of the final message being the poison one. We have reported the details of this technique in [1].

3) Using Node Failure Pattern: Different message types have different failure propagation patterns. One way to exploit the node failure pattern is to use a neural network classifier. The neural network is trained via simulation. A simulation testbed can be set up for a communication network. The testbed has the same topology and protocol configuration as the real network. Then for each message type used in the network, the poison message failure is simulated. And the simulation is run for the probability of a node failure taking on different values. After the neural network is trained, it is applied using the node failure sequence as input, and a pattern match score is the output. In addition, we can combine the neural network with the sequential decision problem. The details are discussed in Section 4.

The node failure pattern can also be combined with the network management and configuration information. For example, suppose BGP is the responsible protocol carrying the poison message. Since only the BGP speaker routers exchange BGP messages, when we get several failed nodes, we can check if all these nodes are BGP speakers. If any failed node is not a BGP speaker, then we can rule out BGP--i.e., BGP is not the poison protocol.

Generate Probability Distribution: The output of passive diagnosis will be a probability distribution that indicates for each protocol (or message type) an estimated probability that it is responsible for the poison message.

3. ACTIVE DIAGNOSIS

From passive diagnosis we have an estimated probability distribution over the possible poison protocols or message types. In active diagnosis, filters are dynamically configured to block suspect protocols or message types. Message filtering can be a valuable tool in helping to identify the culprit message type. For example, if a single message type is blocked and the failure propagation stops, this provides strong evidence that the blocked message type is the poison message. On the other hand, if the propagation continues, that message type can be ruled out.

In addition to its use as a diagnostic tool, filtering offers the possibility of interrupting failure propagation while the culprit message type is being identified. For example, all suspect message types can be initially blocked stopping failure propagation. Then message types can be turned on one-by-one until propagation resumes. While this approach may be attractive in preventing additional node failures during the diagnostic process, disabling a large number of control or management messages may result in unacceptable degradation of network performance. Consequently, the decision making for filter configuration must take into account tradeoffs involving the time to complete diagnosis, the degradation of network performance due to poison message propagation for each of the suspect message types, and the cost to network performance of disabling each of those message types. Each decision on filter configuration leads to further observations, which may call for changing the configuration of the filters. This suggests that policies for dynamic filter configuration may be obtained by formulating and solving a sequential decision problem [6] [12].

The Sequential Decision Problem

- At each stage, the state consists of a probability distribution vector with a component for each message type potentially carrying the poison message, and the recent history of the node failures.
- Based on the current state, a decision (action) is made as to how to configure filters.
- When new node failures are observed, the state is updated based on the current state, action and new observation.
- Actions are chosen according to a policy that is computed off-line based on optimizing an objective function taking into account:

– degree to which each action will help in isolating the responsible message type;

– urgency of diagnosis under each potential message type. E.g., if a suspect protocol sends its messages to a large number of nodes via flooding, the risk of network instability were it to be poison would be particularly great, and this risk provides additional impetus for filtering such a message type;

– impact of filtering action on network performance. A critical protocol or message type should be blocked only if there is a compelling need to do so.

There are three possible outcomes when message filtering is considered.

1) If message filtering is used and the propagation is stopped within a certain time, then either the poison message type is found (if only one message type is filtered) or the types that are not filtered are ruled out (two or more types are filtered).

2) If message filtering is used but the propagation is not stopped within a certain time, then we did not find the responsible message type this time. The filtered message types are removed from the possible suspect set. Collect more information, update the probability vector and reconfigure the filters.

3) If the current action is not to filter any message types, then we simply take another observation. Several other nodes will fail as the poison message propagates in the network. This information is used to update the probability vector. Based on the updated state, a new action is taken to configure the filters.

The sequential decision problem is modeled as a Markov Decision Process (MDP) in [2]. Also we proposed a heuristic policy and used a Q-factor approximation algorithm to obtain an improved policy for the MDP problem in [2].

We have proposed passive diagnosis and active diagnosis to identify the poison message. In this paper, we focus on one of the methods in passive diagnosis – *Using Node Failure Pattern*. We use neural networks to explore the node failure pattern of different poison messages. Also neural networks can be combined with the sequential decision problem in active diagnosis. Details are given in section 4.

4. NEURAL NETWORK SIMULATION RESULTS

We have implemented an OPNET testbed to simulate an MPLS network in which poison messages can be carried by BGP, LDP, or OSPF. The testbed has 14 routers of which 5 are Label Edge Routers and 9 are (non-edge) Label Switching Routers. The topology of the testbed network is shown below.

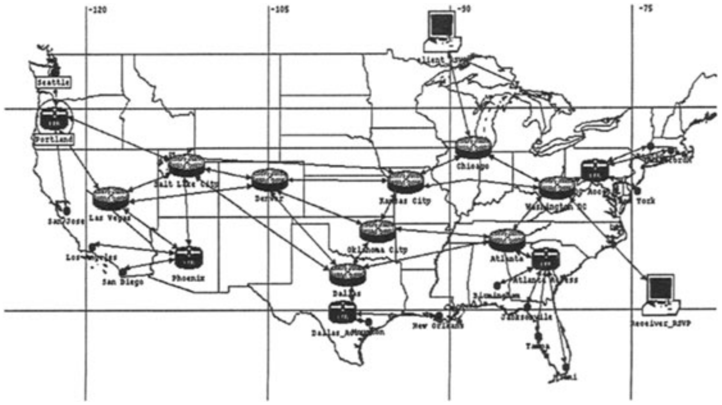


Figure 1. The Topology of OPNET Testbed

We use the Neural Network Toolbox in MATLAB to design, implement, and simulate neural networks. There are several different neural network architectures supported in MATLAB. We implemented two kinds of them in our simulation.

1) Feedforward backpropagation. Standard backpropagation is a gradient descent algorithm. This type of neural network is most commonly used for prediction, pattern recognition, and nonlinear function fitting.

2) Radial basis networks provide an alternative fast method for designing nonlinear feed-forward networks. Variations include generalized regression and probabilistic neural networks. Radial basis networks are particularly useful in classification problems.

Training and Learning Functions are mathematical procedures used to automatically adjust the network's weights and biases. The training function dictates a global algorithm that affects all the weights and biases of a given network. The learning function can be applied to individual weights and biases within a network. The training function we used is: trainb, which is a batch training with weight and bias learning rules.

4.1 Neural Network Structure and Training

We have implemented three feedforward backpropagation neural networks and one radial basis neural network. They have similar structure; all of them have three layers.

1) Input layer with 28 inputs. There are 14 nodes in the communication network. We use a vector to denote the node status in the communication network: $S_k = \left[s_k^1, s_k^2, \ldots, s_k^{14}\right]^T$, where k is the discrete time step, and $s_k^m = 0$ or 1. (0 means this node is normal, and 1 means this node is failed). The 28 inputs represent node status vectors at two consecutive time steps: S_{k-1}, S_k .

2) Hidden layer in the middle. In the middle of the three layers is the hidden layer. There is a transfer function in the hidden layer. A lot of transfer functions are implemented in MATLAB. In our simulation, we use three kinds of transfer functions in the feedforward backpropagation neural networks: 'tansig', 'logsig', and 'purelin'.

a. tansig - Hyperbolic tangent sigmoid transfer function.

b. purelin - Linear transfer function.

c. logsig - Log sigmoid transfer function.

3) Output layer with 12 outputs. The output is the probability distribution vector of the poison message. 12 outputs represent 12 message types in the OPNET testbed. Each output is the probability of the corresponding message being poison.

From the OPNET simulations, we record the node status vector S_k at time k. Then we use these data to train the neural networks. The input of the neural networks is a 28-element vector representing S_{k-1} & S_k. We use 32 sets of such input to train the neural networks. And the target of training (i.e., the output during training) is a 12-element vector representing the probability distribution vector of the poison message. Since we know what is the poison message during simulations, we know the target vectors. For example, a target vector of the 4th message being poison is [000100000000]. We set the training goal to be: error $<10^{-10}$. And the training epoch number set to be 50. One epoch means that the training data is used once. All neural networks meet the training goal.

4.2 Main Test Results

After training neural networks, we use some new data to test the neural networks. In our simulation we use 17 sets of 28-element vectors to test the four different neural networks. The result is very good. All of the neural networks can output a good probability distribution of the poison message for 14 out of 17 input data. Four sets of outputs are listed in Table 1.

In Table 1, "NN" means Neural Networks. And m1,m2,m3 & m4 represent four different poison messages. The numbers 1,2,3 and 4 in 1st row represent four different neural networks. Number 1,2 and 3 represent three feedforward backpropagation neural networks with transfer function being: 'tansig', 'purelin' in 1, 'tansig', 'tansig' in 2, and 'tansig', 'logsig' in 3. And 4 represents the radial basis neural network. The outputs of neural networks include both positive and negative numbers. We call the outputs the 'distribution scores'. And the final probability distribution of poison message is an origin shift and normalization from the distribution scores. I.e., $y=a(x+b)$, where x is the distribution score and y is the probability distribution, and a & b are parameters. Since during the neural network training, the output of non-poison message is set to zero, we set the transformation of the smallest (negative) number in the output vector to be zero. I.e., set b to be $-x_0$ where x_0 is the smallest number. And a can be determined from $\sum y=1$. The data in Table 1 is the probability distribution after transformation. The training data for the neural networks in Table 1 only included five different poison messages. That is why the outputs from neural networks 3 and 4 have several zeros.

Table 1. Output of Neural Networks

NN	1	2	3	4	NN	1	2	3	4
	0.0841	**0.1953**	**0.9992**	**0.8212**		0.0140	0.1302	0	0.0796
ml	0.1235	0.1273	0	0.1127	m2	**0.1791**	**0.1925**	0.0067	**0.3986**
	0.1146	0	0	0.0078		0.1148	0.0656	0	0.1599
	0.0427	0.0191	0.0006	0.0322		0	0	0	0.0032
	0	0.1035	0.0002	0.0262		0.1514	0.1919	0.9933	0.3586
	0.0892	0.0794	0	0		0.0937	0.0471	0	0
	0.0898	0.0870	0	0		0.1149	0.0793	0	0
	0.1331	0.0780	0	0		0.1237	0.0375	0	0
	0.1269	0.0805	0	0		0.1242	0.0844	0	0
	0.0446	0.0844	0	0		0.0022	0.0519	0	0
	0.1143	0.0743	0	0		0.0499	0.0488	0	0
	0.0373	0.0711	0	0		0.0320	0.0708	0	0
	0.0348	0.1247	0.0061	0.0475		0.0003	0.0885	0	0.0096
m3	0	0	0	0.0298	m4	0.0132	0	0	0.0032
	0.1677	**0.1290**	**0.7421**	**0.6973**		0.3187	0.0711	0	0.0032
	0.0413	0.1227	0.0000	0.0010		0.0777	**0.1551**	**0.9999**	**0.9735**
	0.1128	0.1281	0.2518	0.2244		0.0388	0.1549	0.0001	0.0105
	0.0745	0.0496	0	0		0.1162	0.0735	0	0
	0.0515	0.0876	0	0		0.1164	0.0819	0	0
	0.0984	0.0373	0	0		0.1452	0.0731	0	0
	0.0919	0.1057	0	0		0.0738	0.0812	0	0
	0.1172	0.0464	0	0		0.0653	0.0785	0	0
	0.1782	0.0646	0	0		0.0345	0.0676	0	0
	0.0317	0.1043	0	0		0	0.0746	0	0

The boldface numbers are the probabilities assigned to the actual poison messages. And the underlined numbers are instances where the neural networks assign largest probability to wrong message types. Thus, poison messages 1, 3 and 4 are correctly diagnosed by neural networks 2,3,4 but misdiagnosed by 1. Poison message 2 is correctly diagnosed by neural networks 1,2,4 but misdiagnosed by 3. From Table1, we can see that most of the time, the four neural networks can provide a good probability distribution about the poison message--i.e., the neural networks can identify the poison message. Combining all the test results, we find that the radial basis neural network performs the best. Also we find that different neural networks fail for different cases. This suggests that we can combine the outputs from two (or more) neural networks to get better results.

4.3 Serial Test

In the previous tests, we only use S_{k-1} & S_k as input--i.e., we only use the node status at two time steps. Another way to test the neural network is to input a series of node status, e.g., S_1, S_2; S_2, S_3; ... S_{k-1}, S_k. This means we want to input more information to the neural network in the hope of getting better results.

In particular we did the serial tests for the data sets that the neural network failed to correctly diagnose with the original input. The results are encouraging. The neural network can gradually identify the poison message for about 60%~70% of these data sets--i.e., the output gradually changed from a bad probability distribution to good probability distribution. One example of the serial test is given in Table 2. In the example, the neural network is a feedforward backpropagation neural network. We also used radial basis neural network for the serial test and have similar results. In this example, message No. 3 is the poison message. And the data in Table 2 is the probability distribution after origin shift and normalization.

Table 2. Serial Test Result

Input	S_1,S_2	S_2,S_3	S_3,S_4	S_4,S_5	S_5,S_6	S_6,S_7	Ave.
1	0.1296	0.0633	0.1558	0.0339	0.1231	0.0811	0.0865
2	0	0.0056	0.1032	0	0	0.1191	0.0380
3	0.0743	0.1267	0.1101	**0.1635**	**0.1726**	**0.1461**	**0.1415**
4	0.1231	0.1942	0.0725	0.0403	0.1205	0.0412	0.0986
5	0.1258	0.1217	0.1228	0.1100	0.1221	0	0.1004
6	0.0816	0.0888	0.0937	0.0727	0.0570	0.0860	0.0800
7	0.0975	0.1145	0.0774	0.0502	0.0664	0.0866	0.0821
8	0.0569	0.0185	0.0398	0.0960	0.0499	0.1283	0.0649
9	0.0983	0.0491	0	0.0896	0.0822	0.1224	0.0736
10	0.0642	0.1107	0.1337	0.1143	0.0473	0.0431	0.0855
11	0.0652	0	0.0142	0.0961	0.0684	0.1102	0.0590
12	0.0835	0.1070	0.0769	0.1333	0.0906	0.0359	0.0879

In Table 2, the 1st column is list of the 12 message types. And column 2 through column 7 are the probability distribution for different inputs. From Table 2, we can see that at the beginning, the neural network assigns message No. 1 the largest probability, so it does not find the poison message. When we input S_4, S_5, the neural network finds the poison message – message No. 3. And the neural network continues to find the poison message in the later tests. That shows the outputs of the neural network stabilized after input S_4, S_5. The last column is the average probabilities of the previous 6 columns. The average probabilities can be thought of as the combined result from a series of inputs. From Table 2, we can see that the combined result finds the poison message--i.e., it assigns the largest probability to the poison message.

4.4 Integration With the Sequential Decision Problem

The neural network can provide a good probability distribution about the poison message. But it cannot solve the problem completely since the neural network may assign a large probability to a wrong message type. We still need some actions to confirm that we find the poison message. One action is to use message filtering -- turn off the possible poison message, and see if the failure propagation stops in a certain time. If it stops, then the identity of the poison message is confirmed. On the other hand, if failures continue, then we have new data to input to the neural network. However, we need a neural network that takes into account the knowledge that a particular message type is blocked. Thus, it should output a probability distribution over the remaining message types.

We considered two approaches to combining neural networks with message filtering. In the first approach, we added another 12 inputs to the neural network. Each input represents one message type. When a message is turned off, the corresponding input is set to –1. Also the corresponding output in training is set to –1. The idea is that we use –1 to denote that this message is turned off. We trained the new neural network and tested its performance. But it did not work well for the test data.

In the second approach, we assumed that at most one message is turned off at any time. Then we created 12 additional neural networks, one corresponding to each message type that can be turned off. Each neural network has 28 inputs as before, but only 11 (rather than 12) outputs. If a message is turned off, we remove the output corresponding to that message. We trained these neural networks and the test results show that this method works well. One example of the integration with the sequential decision problem is given in Table 3. In the example, a feedforward backpropagation neural network is used.

In Table 3, the first four columns correspond to the test with 11-output neural networks. Each row is the probability of the corresponding message type being poison. And in the first row, k is the time step in the sequential decision problem. The poison message in this example is message No. 3. In the example, we use the policy of filtering the message type with the highest probability at each time step.

The bold number corresponds to the message being turned off. At time step 1, message No. 4 is turned off. Since it is not the poison message, the failure still propagates. We observe some nodes fail and input that node status to the corresponding neural network with 11 outputs. We list the outputs in column 3 (k=2). From Table 3 we can see that at time step 2, message No. 5 has the highest probability and it is turned off. The poison message is not found yet. So we observe some other node failure, and input the information to the neural network corresponding to message No. 5 turned off. Then at time step 3, we find the poison message -- message No. 3.

Table 3. Integration of Neural Networks With the Sequential Decision Problem

Step	k = 1	k = 2	k = 3	New	k = 1	k = 2	k = 3
p_1	0.1086	0.1475	0.1431	Test	0.1086	0.1669	0.1665
p_2	0	0	0		0	0.1440	0
p_3	0.1280	0.1459	**0.1480**		0.1280	0.1831	**0.1722**
p_4	**0.1650**	/	0.1407		**0.1650**	0	0
p_5	0.1633	**0.1478**	/		0.1633	**0.1881**	0
p_6	0.0386	0.0689	0.0569		0.0386	0.0451	0.0662
p_7	0.0541	0.0803	0.1004		0.0541	0	0.1169
p_8	0.0906	0.0604	0.0428		0.0906	0.0621	0.0498
p_9	0.0553	0.0995	0.1213		0.0553	0.0199	0.1411
p_{10}	0.1194	0.0572	0.0533		0.1194	0.0326	0.0620
p_{11}	0.0671	0.0828	0.0741		0.0671	0.1030	0.0862
p_{12}	0.0099	0.1097	0.1196		0.0099	0.0552	0.1392

Since the identity of the poison message does not change, it follows that if a message type has been filtered and propagation continues, that message type can be ruled out as the poison message. Thus, it would make sense to eliminate it as a possible output from the neural networks applied in subsequent time steps. In the example, when message type 5 is blocked, it would be desirable to use a neural network that had the outputs for both types 4 and 5 removed. However, to apply this approach in general would require training a neural network corresponding to each subset of the set of message types. The number of neural networks would be exponential in the number of message types, which is not practical. So we tried another approach that uses the original neural network with 12 outputs and computes the conditional probabilities given all message types that have been previously ruled out. For the message type that has been ruled out, its probability is set to zero, and the probability distribution is obtained by normalizing the rest of the outputs. The results of the new tests are listed in the last three columns in Table 3, and the underline zeros are those set manually. From table 3, we can see that the new approach works well, and it requires only one neural network.

If different message types have different filtering costs, the policy of blocking the message type with highest probability can be extended to the well known heuristic policy based on the ratios $E[C_j]/p_j$. I.e., the message type selected for blocking is the one that has the smallest ratio $E[C_j]/p_j$ where $E[C_j]$ is the expected cost (in terms of network performance) associated with blocking message type j for a time step, and p_j is the current estimate of the probability that message type j is the poison one. In our example, we are considering the special case where the costs are all equal.

5. SUMMARY

We have discussed a particular failure propagation mechanism--poison message failure propagation, and provided a framework to identify the responsible protocol or message type. We have proposed passive diagnosis, which includes the FSM method applied at individual failed nodes, correlating protocol events across multiple failed nodes and using node failure pattern recognition. If passive diagnosis cannot solve the problem by itself, it can be augmented by message type filtering, which is formulated as a sequential decision problem. In this paper, we focus on identifying node failure pattern using artificial neural networks. We have implemented and tested four different types of neural networks. Our tests show that neural networks can provide a good probability distribution for the poison message in most cases. We also performed the serial test that works for many of the data sets for which the original test failed. Furthermore, we have combined the neural networks with the sequential decision problem. In the sequential decision problem, the decision as to which message type(s) to filter is based on the current state consisting of the set of failed nodes together with the estimated probability that each message type is poison. The neural networks appear to be effective as a computational tool for updating these probabilities without requiring an explicit model for the transition probabilities in the underlying Markov chain.

ACKNOWLEDGEMENT

This research was partially supported by DARPA under contract N66001-00-C-8037.

REFERENCES

[1] X. Du, M.A. Shayman and R. Skoog, "Preventing Network Instability Caused by Control Plane Poison Messages" *IEEE MILCOM 2002*, Anaheim, CA, Oct. 2002.

[2] X. Du, M.A. Shayman and R. Skoog, "Markov Decision Based Filtering to Prevent Network Instability from Control Plane Poison Messages" submitted for publication.

[3] A. Bouloutas, et al, "Fault identification using a finite state machine model with unreliable partially observed data sequences," *IEEE Tran. Communications*, Vol.: 41 Issue: 7, pp: 1074–1083, July 1993.

[4] R. Skoog et al., "Network management and control mechanisms to prevent maliciously induced network instability," *Network Operations and Management Symposium*, Florence, Italy, April 2002.

[5] H. Li and J. S. Baras, "A framework for supporting intelligent fault and performance management for communication networks", *Technical Report, CSHCN TR 2001-13*, University of Maryland, 2001.

[6] M.A. Shayman and E. Fernandez-Gaucherand, "Fault management in communication networks: Test scheduling with a risk-sensitive criterion and precedence constraints," *Proceedings of the IEEE Conference on Decision and Control,* Sidney, Australia, Dec. 2000.

[7] I. Katzela; M. Schwartz, "Schemes for Fault Identification in Communication Networks", *Networking, IEEE/ACM Transactions on* , Vol.: 3. Issue: 6, pp: 753 – 764, Dec. 1995.

[8] D. J. Houck, K. S. Meier-Hellstern, and R. A. Skoog, "Failure and congestion propagation through signaling controls". *In Proc. 14th Intl. Teletraffic Congress*, Amsterdam: Elsevier, pp: 367–376, 1994.

[9] A. Bouloutas, S. Calo, and A. Finkel, "Alarm correlation and fault identification in communication networks", *Communications, IEEE Transactions on*, Vol.: 42, Issue: 2, pp: 523 -533, Feb-Apr 1994.

[10] D.A. Castanon, "Optimal search strategies in dynamic hypothesis testing", *IEEE Trans. Systems, Man and Cybernetics*, Vol.: 25 Issue: 7, July 1995

[11] A. Bouloutas, G.W. Hart and M. Schwartz, "Simple finite-state fault detection for communication networks," *IEEE Trans. Communications*, Vol. 40, Mar. 1992.

[12] J-F. Huard and A.A. Lazar. "Fault isolation based on decision-theoretic troubleshooting", *Technical Report 442-96-08, Center for Telecommunications Research*, Columbia University, New York, NY, 1996.

[13] R. Sutton and A. Barto. *Reinforcement Learning*: An Intro.. MIT Press 1998.

[14] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, Vols. I and II, Athena Scientific, 2000.

[15] D. Bertsekas and J. Tsitsiklis. *Neurodynamic Programming*, Athena Scientific, 1996.

PROBABILISTIC EVENT-DRIVEN FAULT DIAGNOSIS THROUGH INCREMENTAL HYPOTHESIS UPDATING

M. Steinder and A. S. Sethi
Computer and Information Sciences Department
University of Delaware, Newark, DE
{steinder,sethi}@cis.udel.edu

Abstract: A probabilistic event-driven fault localization technique is presented, which uses a symptom-fault map as a fault propagation model. The technique isolates the most probable set of faults through incremental updating of the symptom explanation hypothesis. At any time, it provides a set of alternative hypotheses, each of which is a complete explanation of the set of symptoms observed thus far. The hypotheses are ranked according to a measure of their goodness. The technique allows multiple simultaneous independent faults to be identified and incorporates both negative and positive symptoms in the analysis. As shown in a simulation study, the technique is resilient both to noise in the symptom data and to the inaccuracies of the probabilistic fault propagation model. [1]

1. Introduction

This paper presents a non-deterministic event-driven fault localization [9, 10, 17] technique, which uses a probabilistic symptom-fault map as a fault propagation model. While investigating fault localization techniques suitable for bipartite fault propagation models, this paper states the following objectives:

- Usage of probabilistic reasoning, which is necessary to diagnose Byzantine problems or when relationships among system events may not be determined with certainty, e.g., due to their dynamic nature [5, 6, 8, 10, 11].
- Ability to isolate multiple simultaneous faults even if their symptoms overlap [6, 10], which improves the technique's applicability to large systems.
- Event-driven diagnosis, which avoids the inflexibility of window-based tools [1], is not prone to inaccuracies resulting from an incorrect time-window specification, and allows fault localization to be interleaved with testing.
- Resilience to lost and spurious symptoms [5, 8, 17], which may dramatically reduce its accuracy if not taken into account by a fault localization algorithm.
- High accuracy and low-polynomial computational complexity.

In addition to providing the above features, the fault localization technique proposed in this paper is incremental, i.e., the interpretation of an observed symptom is incorporated in a solution resulting from the interpretation of the previously observed

[1]Prepared through collaborative participation in the Communications and Networks Consortium sponsored by the U. S. Army Research Laboratory under the Collaborative Technology Alliance Program, Cooperative Agreement DAAD19-01-2-0011. The U. S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

symptoms without re-analyzing them. Thanks to this feature, the algorithm continuously provides a system administrator with information about which faults are likely to exist in the system given symptoms observed thus far. In non-incremental techniques, such information is available on a periodic basis only [10, 17]. The technique proposed here produces a set of alternative hypotheses rather than just a single explanation. These hypotheses are ranked according to their measure of goodness. As a result, the system administrator obtains a better understanding of the system state. This feature also facilitates exchanging the hypotheses order as dictated by hypothesis ranking schemes that are not easy to express through a goodness function, e.g., those taking into account fault gravity, testing difficulty, or urgency of repair. Since an occasional inaccuracy of the most likely hypothesis may not be avoided, the ability to replace an incorrect hypothesis with its alternative without repeating the entire fault localization process improves the robustness of the fault management system.

While relationships between faults and symptoms in real-life systems are usually more complex than may be represented by a bipartite graph (in particular, they are frequently indirect), many fault localization techniques proposed in the literature [4, 10, 16, 17] use bipartite fault propagation models. The focus on this type of a model is justified by the following arguments: (1) Performing fault localization with more complex representations is difficult. (In general, the problem is NP-hard [10].) To avoid this complexity, more detailed models are frequently reduced to bipartite ones through a sequence of graph reduction operations [17]. (2) Building more complex models requires a profound knowledge of the underlying system, while symptom-fault maps may be obtained through external observation. In many real-life problems, only bipartite symptom-fault models are feasible [4]. (3) Some fault localization sub-problems may be accurately represented by bipartite symptom-fault maps [16], thereby necessitating fault localization algorithms suitable for bipartite fault propagation models. The distinguishing features of the approach presented in this paper, when compared to previous fault localization techniques suitable to use with a symptom-fault map as a fault propagation model, are as follows: the technique is more general by not assuming any particular problem domain or probabilistic model [4, 10, 17], resilient to lost and spurious symptoms [7, 10], event-driven [4, 7, 10, 17], incremental [4, 7, 10, 17, 15, 16], and more efficient [15, 16].

The paper is structured as follows. Section 2 describes the concept of probabilistic incremental hypothesis updating, which was originally introduced in [13]. In Section 3, incremental hypothesis updating is extended to include positive and lost symptoms in the analysis. Section 4 presents the methodology of dealing with spurious symptoms and discusses the necessary modifications to the original algorithm. In Section 5, the experimental study of the technique is described.

2. Incremental hypothesis updating

A symptom-fault map is a bipartite directed graph that, for every fault, encodes direct causal relationships between the fault and a set of symptoms observed when the fault occurs. We will use $\mathcal{F}$ and $\mathcal{S}$ to denote the sets of all possible faults and symptoms, respectively. In a non-deterministic model, with every fault $f_i \in \mathcal{F}$ a probability of its independent failure is associated, which is denoted by $p(f_i)$. The edge between $f_i \in \mathcal{F}$ and $s_j \in \mathcal{S}$ indicates that f_i may cause s_j. The edge is weighted with the probability of the causal implication, $p(s_j|f_i)$. A subset of symptoms observed by the management application is denoted by $\mathcal{S}_O$. The purpose of fault localization is to find $\mathcal{F}_d \subseteq \mathcal{F}$ that maximizes the probability that (1) all faults in $\mathcal{F}_d$ occur and (2) each symptom in $\mathcal{S}_O$ is explained by at least one fault from $\mathcal{F}_d$.

The technique we present in this section, which is called *incremental hypothesis updating* [13] (IHU), creates a set of most likely hypotheses, which may all be presented to the system administrator. Rather than waiting for a specific period of time before presenting a solution, the technique makes all these hypotheses available on a continuous basis, and constantly upgrades them with the information learned from the arriving symptoms. This allows the administrator to initiate recovery actions sooner, and it allows additional testing procedures to be performed. Each hypothesis is a subset of $\mathcal{F}$ that explains all symptoms in $\mathcal{S}_O$. We say that hypothesis $h_j \subseteq \mathcal{F}$ explains symptom $s_i \in \mathcal{S}_O$ if it contains at least one fault that explains s_i. The hypotheses are ranked using a belief metric, b. The algorithm proceeds in an event-driven and incremental fashion. The execution triggered by the i^{th} symptom, s_i, creates a set of hypotheses, $\mathcal{H}_i$, each explaining symptoms s_1 through s_i. Set $\mathcal{H}_i$ is created by updating $\mathcal{H}_{i-1}$ with an explanation of symptom s_i. We define H_{s_i} as a set $\{f_k \in \mathcal{F}\}$ such that f_k may cause s_i, i.e., the fault propagation model contains a directed edge from f_k to s_i. Using the notation from [10], H_{s_i} is the domain of symptom s_i.

After the i^{th} symptom is processed, belief metric b_i represents the probability that (1) all faults belonging to h_j have occurred, and (2) h_j explains every observed symptom $s_k \in \mathcal{S}_{O,i} = \{s_1, \ldots, s_i\}$. Formally, $b_i(h_j)$ is defined as follows:

$$b_i(h_j) = \Big(\prod_{f_k \in h_j} p(f_k)\Big) \prod_{s_l \in \mathcal{S}_{O,i}} \Big(1 - \prod_{f_k \in h_j} (1 - p(s_l|f_k))\Big) \qquad (1)$$

To incorporate an explanation of symptom s_i into the set of fault hypotheses, in the i^{th} iteration of the algorithm, we analyze each $h_j \in \mathcal{H}_{i-1}$. If h_j is able to explain symptom s_i, we put h_j into $\mathcal{H}_i$. Otherwise, h_j has to be extended by adding to it a fault from H_{s_i}. To avoid a very fast growth in the size of $\mathcal{H}_i$, the following heuristic is used. Fault $f_l \in H_{s_i}$ may be added to $h_j \in \mathcal{H}_{i-1}$ only if the size of h_j, $|h_j|$, is smaller than $\mu(f_l)$, the minimum size of a hypothesis in $\mathcal{H}_{i-1}$ that contains f_l and explains s_i. The usage of this heuristic is derived from the fact that the probability of multiple simultaneous faults is small. Therefore, of any two hypotheses containing f_l, the hypothesis that contains the fewest faults is the most likely to constitute the optimal symptom explanation. Thus, since it is not efficient to keep all possible hypotheses, we remove those that are bigger in size. While updating the set of hypothesis, $b_i(h_j)$ is approximated iteratively based on $b_{i-1}(h_j)$ using the following equations:

– If $h_j \in \mathcal{H}_{i-1}$ and h_j explains s_i

$$b_i(h_j) = b_{i-1}(h_j)\Big(1 - \prod_{f_l \in h_j \cap H_{s_i}} (1 - p(s_i|f_l))\Big) \qquad (2)$$

– Otherwise, if f_l explains s_i

$$b_i(h_j \cup \{f_l\}) = b_{i-1}(h_j)\, p(f_l)\, p(s_i|f_l) \qquad (4)$$

The upper bound on the worst case computational complexity of the resultant algorithm is $\mathcal{O}(|\mathcal{S}_O|k|\mathcal{F}|)$, where k is the maximum size of the set of hypotheses and k is $\mathcal{O}(|\mathcal{F}|)$ (in our study, $k = 2|\mathcal{F}|$). When $|\mathcal{H}_i| = k$, a new hypothesis may be added to $\mathcal{H}_i$ only after a hypothesis with the smallest $b_i()$ is removed.

3. The analysis of positive symptoms

The original version of IHU [13] formulates explanations of observed system disorder while not taking advantage of the fact that some possible indications of the disorder have not been observed. As many researchers point out [2, 17], the fact that many of its possible symptoms have not been observed should decrease our confidence in the fault's occurrence. In the realm of fault localization, an observation of network disorder is called a *negative symptom.* Both an opposite observation and the lack of any observation are considered *positive symptoms.* As it was shown in the study on fault localization with belief networks [15], the inclusion of positive symptoms into the fault localization process may significantly increase its accuracy.

To include the analysis if positive symptoms in IHU, the belief metric b_i^* associated with hypothesis $h_j \in \mathcal{H}_i$ needs to contain two components: a negative component b_i^n and a positive component b_i^p, where $b_i^*(h_j) = b_i^n(h_j)\, b_i^p(h_j)$ and $b_i^n(h_j) = b_i(h_j)$ of Equation (1). The positive component is defined as the probability that faults in h_j have not generated any of the symptoms in $\mathcal{S} - \mathcal{S}_{o,i}$. It decreases the value of the belief metric associated with hypothesis h_j if many of the symptoms that can occur as a result of faults in h_j have not been observed. The positive component of $b_i^*(h_j)$ is expressed through the following equation.

$$b_i^p(h_j) = \prod_{s_l \in \mathcal{S}-\mathcal{S}_{O,i}} \prod_{f_k \in h_j} (1 - p(s_l|f_k)) \tag{4}$$

When investigating a fault localization technique that takes advantage of positive symptoms, two properties of the managed system have to be taken into account: symptom observability ratio and symptom loss rate, which lead to refinements in the calculation of b_i^p presented in the following sections.

3.1 Symptom observability ratio

Frequently, an indication of an existing disorder may not be observed by the management system because the system configuration configuration excludes some conditions from being monitored, or filters out some of the symptoms before they reach the management application. If this fact is not taken into account, the reduction of $b_i^*(h_j)$ caused by the positive multiplier $b_i^p(h_j)$ may be excessive. Symptoms which may not be observed as a result of the management system configuration may be dealt with by not including them in the fault propagation model. An alternative solution, which preserves the model despite the management system configuration changes, associates a flag 1 or 0 with every symptom in the model to indicate that, in the current configuration, the symptom is observable or not observable, respectively. We will denote by $\mathcal{S}^o \subseteq \mathcal{S}$ the set of all symptoms which are observable in a current management system configuration. When symptom observability status is taken into account, the second product in Equation (4) is calculated over $s_l \in \mathcal{S}^o - \mathcal{S}_{O,i}$ rather than $s_l \in \mathcal{S} - \mathcal{S}_{O,i}$.

The ratio of the number of all observable symptoms to the number of all possible symptoms is called an observability ratio, and is denoted by $OR = |\mathcal{S}^o|/|\mathcal{S}|$ [15]. The observability ratio is an important parameter of the fault management system, which informs us of the extensiveness of the system instrumentation. It may be expected that a higher instrumentation level allows fault localization to be more accurate, but causes it to be less efficient as it requires the processing of more symptoms.

3.2 Symptom loss

In a real-life system, a symptom that has been triggered by faults in h_j may be lost before it reaches the management application as a result of using an unreliable communication mechanism to transfer alarms from their origin to the management node, as is the case with the SNMP protocol [3], or too liberal threshold values which prevent an existing problem from being reported. When a fault localization algorithm relies on positive information, a high rate of lost symptoms, if ignored by the algorithm, can reduce its accuracy. Thus, in the management system in which symptom delivery is not guaranteed, including positive symptoms into account necessitates the analysis of lost symptoms as well.

Let us denote by $p_{\text{loss}}(s_i)$ the probability that symptom $s_i \in \mathcal{S}$ is lost. The value of $p_{\text{loss}}(s_i)$ may be derived from a packet loss rate in the communication system, or from the confidence measure associated with the system baselining tool used to calculate the monitored threshold values. Symptom loss is included into the fault localization algorithm by modifying the definition of $b_i^p(h_j)$ (Equation (4)) as follows.

$$b_i^p(h_j) = \prod_{s_l \in \mathcal{S}^o - \mathcal{S}_{O,i}} \left(p_{\text{loss}}(s_l) + (1 - p_{\text{loss}}(s_l)) \prod_{f_k \in h_j} (1 - p(s_l|f_k)) \right) \tag{5}$$

3.3 Incremental calculation of b^p

IHU based on both positive and negative symptoms proceeds as follows. Initially, all observable alarms are considered positive symptoms. The only valid hypothesis is $\emptyset$, and $b_i^n(\emptyset) = b_i^p(\emptyset) = 1$. In the process of analyzing new symptoms, the value of belief metric $b_i^*(h_j)$ is calculated by multiplying $b_i^n(h_j)$ and $b_i^p(h_j)$, where $b_i^n(h_j)$ is computed incrementally using Equations (2)-(3). We obtain $b_i^p(h_j)$ as follows.

- If $h_j \in \mathcal{H}_{i-1}$ explains symptom s_i, then $b_i^p(h_j)$ may be approximated using the following formula.

$$b_i^p(h_j) = \frac{b_{i-1}^p(h_j)}{\prod_{f_l \in h_j} (p_{\text{loss}}(s_l) + (1 - p_{\text{loss}}(s_l))(1 - p(s_l|f_k)))} \tag{6}$$

- Otherwise, let $f_l \in H_{s_i}$ be a fault used to extend h_j. The value of $b_i^p(h_j \cup \{f_l\})$ is calculated as follows.

$$b_i^p(h_j \cup \{f_l\}) = b_{i-1}^p(h_j)\, b_i^p(\{f_l\}) \tag{7}$$

In Equation (7), $b_i^p(\{f_l\})$ denotes the positive component of a belief metric associated with a singleton hypothesis $\{f_l\}$ calculated given all symptoms observed thus far. The values of $b_i^p(\{f_l\})$ are pre-computed when the model is initialized. After every symptom observation, $b_i^p(\{f_l\})$ is incrementally updated using Equation (6).

4. Dealing with spurious symptoms

In real-life communication systems, an observation of a network state is frequently disturbed by the presence of spurious symptoms, which are caused by intermittent network faults or by overly restrictive threshold values. Spurious symptoms, if not taken into account by the fault localization process, may significantly deteriorate its accuracy. When a fault localization algorithm does not recognize that some symptoms may be spurious (as such they do not require an explanation), it strives to find the explanation of all the observed symptoms, thereby creating hypotheses which contain many non-existent faults [15]. In this section, we introduce an extended version of IHU, IHU+, which incorporates spurious symptoms in the analysis.

To deal with spurious symptoms IHU has to be modified as follows. Let s_i be the i^{th} observed symptom and let $p_s(s_i)$ denote the probability that symptom s_i is spuriously generated. While deciding whether hypothesis $h_j \in \mathcal{H}_{i-1}$ should be placed in $\mathcal{H}_i$ without modification or extended, the algorithm has to consider two possibilities: (1) that the symptom is valid and (2) that the symptom is spurious. When hypothesis h_j explains s_i, then regardless of these two possible interpretations of symptom s_i, hypothesis h_j can be added to $\mathcal{H}_i$ and the two choices are incorporated in the calculation of the belief metric for h_j. When hypothesis h_j does not explain s_i, then treating s_i as valid necessitates extending h_j, and treating s_i as spurious allows us to put h_j in $\mathcal{H}_i$ without extension. Since the first and second cases occur with probability $1 - p_s(s_i)$ and $p_s(s_i)$, these values are used as multipliers embedded in the calculation of the corresponding values of the belief metric. Recall from Section 2, that the original algorithm does not allow adding $h_j \in \mathcal{H}_{i-1}$ to $\mathcal{H}_i$ unless it explains or is extended to explain symptom s_i.

The inclusion of spurious symptoms into the analysis only affects the calculation of the negative component, $b_i^{+n}(h_j)$, of the belief metric, $b_i^{+}(h_j)$, while the positive component remains the same, i.e., $b_i^{+p}(h_j) = b_i^{p}(h_j)$ (Eqns. (6)-(7)). The modified negative component, $b_i^{+n}(h_j)$, is calculated iteratively as follows.

- If $h_j \in \mathcal{H}_{i-1}$ explains symptom s_i, then

$$b_i^{+n}(h_j) = b_{i-1}^{+n}(h_j)\Big(p_s(s_i) + (1 - p_s(s_i))\,(1 - \prod_{f_l \in h_j \cap H_{s_i}} (1 - p(s_i|f_l))\Big) \quad (8)$$

- Otherwise

$$b_i^{+n}(h_j) = b_{i-1}^{+n}(h_j)\,p_s(s_i) \quad (9)$$

In addition, for every fault $f_l \in H_{s_i}$ used to extend h_j

$$b_i^{+n}(h_j \cup \{f_l\}) = b_{i-1}^{+n}(h_j)\,p(f_l)\,p(s_i|f_l)\,(1 - p_s(s_i)) \quad (10)$$

Besides modifying the definition of the belief metric, the inclusion of the spurious symptoms' analysis in the fault localization process necessitates two additional changes in the original IHU. Recall from Section 2 that IHU takes advantage of two heuristics that allow us to limit the size of the set of hypotheses. The first heuristic forbids adding fault f_l to hypothesis $h_j \in \mathcal{H}_i$ if the size of the resultant hypothesis $h_j \cup \{f_l\}$ would be greater than $\mu(f_l)$. The second heuristic applied by Algorithm IHU limits the maximum size of the set of hypotheses to $k \in \mathcal{O}(|\mathcal{F}|)$ and removes the least probable hypotheses if this limit is exceeded. These two heuristics are modified in IHU+ as described in the following sections.

4.1 Calculating hypothesis size

In IHU, function $\mu(f_l)$ is defined as the minimum size of $h_k \in \mathcal{H}_{i-1}$ that contains f_l and explains symptom s_i, where the size of h_k is $|h_k|$. In IHU+, the size of h_j, $\alpha(h_j)$ is defined as the number of faults in h_j plus the number of symptoms observed so far that h_j considers spurious. This modification serves two purposes. It:

- Helps avoid the creation of duplicate hypotheses.
  Duplicate hypotheses introduce redundancy into the set of hypotheses, which may affect the accuracy of the technique. They increase the size of the set of hypotheses thereby making hypothesis removal due to the excessive set size more frequent. Thus, they increase the probability of removing a (currently) least likely hypothesis that may later turn out to be optimal. Although it is possible to unify duplicate hypotheses within the computational complexity bound of IHU+, the necessity to do so renders the implementation of the algorithm more difficult.
- Prevents hypotheses that contain fewer faults while not explaining many symptoms from being given unwarranted preference.
  When small hypotheses are unfairly favored over bigger hypotheses, it is difficult for the algorithm to extend a small hypothesis so that it provides an explanation to a bigger number of symptoms. As a result, the algorithm is likely not to provide an explanation to many observed symptoms.

4.2 Controlling hypotheses number

The second heuristic applied by Algorithm IHU limits the maximum size of the set of hypotheses to $k \in \mathcal{O}(|\mathcal{F}|)$. To add a new hypothesis to $\mathcal{H}_i$, when $|\mathcal{H}_i| = k$, a hypothesis h_l for which $b_i(h_l)$ is minimal must be first removed from $\mathcal{H}_i$. It is possible that symptoms to be received in the next iterations would increase the belief associated with h_l so that h_l would become the most probable hypothesis. If such h_l is removed at an earlier stage of the fault localization process, the algorithm will not propose the optimal solution. The phenomenon of removing a hypothesis that would become optimal at a later stage of fault localization, if it was kept in the set of hypotheses, will be referred to as the problem of *premature hypothesis removal.*

Although the problem of premature hypothesis removal exists regardless of including positive, lost, and spurious symptoms into the analysis, in most cases it may be ignored. A hypothesis removal due to the big size of $\mathcal{H}_i$ is a rare event, and it usually happens after many symptoms have been observed and analyzed. At this stage, the algorithm is already converging to the final solution, thus the removed hypothesis is not likely to become optimal in the future. However, when spurious symptoms are included in the analysis, the size of $\mathcal{H}_i$ grows much faster, and therefore the probability of prematurely removing an optimal hypothesis is high. The early removal of an optimal hypothesis is caused by the positive component of the belief metric, whose value may be very small if at this stage of fault localization, only a few symptoms related to the optimal hypothesis have been observed. The crux of the problem is that $b^{+p}(h_j)$ is calculated as if the current set of observed symptoms was the final one.

IHU+ avoids the problem of the premature hypothesis removal by using function rank_i rather than b^{+} to choose a hypothesis that has to be removed. Similar to the belief metric, function rank_i is composed of positive and negative components b^{+p} and b^{+n}, but the contribution of b_i^{+p} is weighted according to the number of symptoms observed so far. In the following definition of $\mathrm{rank}_i(h_j)$, $B_i^{+n}(h_j)$ and $B_i^{+p}(h_j)$ represent logarithmic-scale values of $b_i^{+n}(h_j)$ and $b_i^{+p}(h_j)$, respectively.

$$\mathrm{rank}_i(h_j) = B_i^{+n}(h_j) + \beta(i)B_i^{+p}(h_j) \tag{11}$$

Function $\beta(i)$ represents the contribution of the positive belief-metric component. In general, function $\beta(i)$ should assume a very small value when the number of symptoms observed so far, i, is small, and increase asymptotically to 1 as the value of i increases. In this study, we define $\beta(i)$ as follows.

$$\beta(i) = 1 - 2^{-SW(\frac{i-1}{EEF})^2} \tag{12}$$

In Equation (12), the expected evidence factor, EEF, and the average symptom weight, SW, are model-dependent. The expected evidence factor determines how quickly the value of $\beta(i)$ should converge to 1 in the absence of spurious symptoms. It is proportional to the average number of symptoms which may be observed per fault, i.e., EEF $= c\frac{|\mathcal{S}|\mathrm{OR}}{|\mathcal{F}|}$. In this study, we use $c = 4$. The average symptom weight accounts for the fact that some symptoms may be spurious, and, as such, should not increase the value of $\beta(i)$. This value should be equal to 1 when no spurious symptoms occur, and decrease as the spurious symptom probability increases. We define SW using the following formula.

$$\mathrm{SW} = 1 - \frac{\sum_{s_i \in \mathcal{S}} p_{\mathrm{s}}(s_i)}{\sum_{s_i \in \mathcal{S}} \sum_{f_l \in \mathcal{F}} p(s_i|f_l) + \sum_{s_i \in \mathcal{S}} p_{\mathrm{s}}(s_i)} \tag{13}$$

The values of EEF and SW are pre-computed at the model initialization phase, and remain constant during the process of fault localization, as long as the fault propagation model is not changed. Other definitions of function β are possible. For instance, we could incorporate a temporal aspect into function β by increasing its value with time. Such a definition could represent a property that, after a certain time since the fault localization process is started, all relevant symptoms should have been observed.

4.3 IHU+ algorithm

We are now ready to define an extended version of the incremental algorithm, IHU+, which incorporates positive, lost, and spurious symptoms in the analysis and is parametrized by observability ratio OR, symptom-loss probability function p_{loss}, and spurious-symptom probability function p_{s}.

Algorithm: Incremental hypothesis updating – IHU+(OR,$\mathbf{p_{loss}}$,$\mathbf{p_s}$)

```
let $\mathcal{H}_0 = \{\emptyset\}$, $b_0^{+n}(\emptyset) = b_0^{+p}(\emptyset) = 1$, and $\alpha(\emptyset) = 0$
for every observed symptom $s_i$:
    let $\mathcal{H}_i = \emptyset$, and for all $f_l \in \mathcal{F}$ let $\mu(f_l) = |\mathcal{F}| + |\mathcal{S}_O|$
    for all $h_j \in \mathcal{H}_{i-1}$ do
        for all $f_l \in h_j$ such that $f_l \in H_{s_i}$
            set $\mu(f_l) = \min(\mu(f_l), \alpha(h_j))$
            add $h_j$ to $\mathcal{H}_i$ and calculate $b_i^+(h_j)$
    for all $h_j \in \mathcal{H}_{i-1} \setminus \mathcal{H}_i$ do
        if $p_{\mathrm{s}}(s_i) > 0$
            add $h_j$ to $\mathcal{H}_i$, calculate $b_i^+(h_j)$, and set $\alpha(h_j) = \alpha(h_j) + 1$
        for all $f_l \in \mathcal{F} \cap H_{s_i}$ such that $\mu(f_l) > \alpha(h_j)$ do
            add $h_j \cup \{f_l\}$ to $\mathcal{H}_i$, compute $b_i^+(h_j \cup \{f_l\})$, and set $\alpha(h_j) = \alpha(h_j) + 1$
choose $h_j \in \mathcal{H}_{|\mathcal{S}_N|}$ such that $b_{|\mathcal{S}_N|}^+(h_j)$ is maximum
```

Observe that the worst-case computational complexity of the algorithm that takes positive, lost, and spurious symptoms into account is still $\mathcal{O}(|\mathcal{S}_O||\mathcal{F}|^2)$.

5. Simulation study

In this section, we describe a simulation study performed to evaluate the techniques presented in Sections 2, 3, and 4. As a real-life application domain we chose end-to-end service failure diagnosis [16], which deals with isolating faults responsible for a malfunctioning of end-to-end connectivity between systems. The first step toward diagnosing these problems is to isolate the responsible host-to-host services, where a host is an intermediate node used to provide the end-to-end connectivity. In the problem of end-to-end service-failure diagnosis, a fault propagation model is a bipartite causality graph with host-to-host and end-to-end service failures at the tails and at the heads of the edges, respectively.

The simulation study presented in this paper uses tree-shaped network topologies, which result, for example, from the usage of the Spanning Tree Protocol [12] as the data-link layer routing protocol. The usage of tree-shaped topologies greatly simplifies their random generation, while it does not affect the validity of the results presented in this section. We focus on diagnosing Byzantine types of problems, for which the usage of a non-deterministic fault propagation model is necessary.

We design the simulation described in this section according to the model we previously used to evaluate another fault localization algorithm [15], which is based on belief propagation in belief networks. We use OR, LR , and SSR to denote the observability ratio ($|\mathcal{S}_O|/|\mathcal{S}|$), ratio of the number of generated alarms that were lost to the number of all generated alarms (i.e., alarm loss rate), and probability that an alarm is generated in a spurious manner (i.e., spurious symptom rate), respectively. We aim at creating a homogeneous set of test scenarios to establish the upper limit on the accuracy of the proposed techniques and its relationship to the parameters of the simulation model. Consequently, we assume that the fault propagation model used in the study accurately approximates the relationships that exist in the real system.

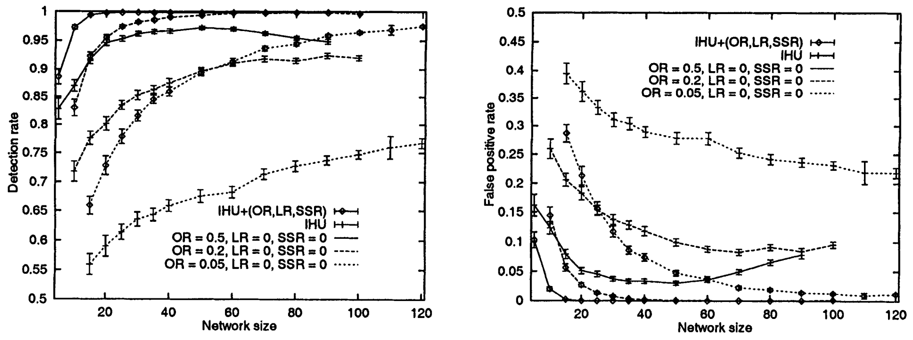
Given the simulation model with parameters OR, LR, and SSR for a given network topology of size n, where n represents the number of intermediate network nodes, we design 100 simulation cases. We build a random tree-shaped topology, and generate the probability distribution in the fault localization model. The independent failure probabilities and conditional probabilities are uniformly distributed in ranges $[0.001, 0.01]$ and $(0, 1)$, respectively. We randomly choose OR$|\mathcal{S}|$ observable symptoms, and place them in the set of observable symptoms, $\mathcal{S}^o$. In a simulation case, we create a number of simulation scenarios (typically 100-200) as follows. We randomly generate a set of faults that exist in the system, $\mathcal{F}_c \subseteq \mathcal{F}$. Using $\mathcal{F}_c$ and the conditional probability distribution we randomly generate the set of observed negative symptoms $\mathcal{S}_O \subseteq \mathcal{S}^o$. When SSR > 0, we also randomly choose SSR $|\mathcal{S}^o|$ symptoms from $\mathcal{S}^o$, and add them to $\mathcal{S}_O$. When LR > 0, we remove LR $|\mathcal{S}_O|$ random symptoms from $\mathcal{S}_O$. Then, we run algorithms IHU, IHU+, or both to produce the most probable explanation of $\mathcal{S}_O$, $\mathcal{F}_d$, i.e., the hypothesis with the highest value of belief metric in the final set of hypotheses proposed by the algorithm. We compare $\mathcal{F}_d$ to $\mathcal{F}_c$, and calculate the detection rate DR $= \frac{|\mathcal{F}_c \cap \mathcal{F}_d|}{|\mathcal{F}_c|}$ and false positive rate FPR $= \frac{|\mathcal{F}_d - \mathcal{F}_c|}{|\mathcal{F}_d|}$.

5.1 The impact of including positive symptoms

To evaluate the impact of including positive symptoms into fault localization, we set LR $= 0$, and SSR $= 0$ in the simulation model. Correspondingly, we use $p_{\text{loss}}(s_i) = 0$

and $p_s(s_i) = 0$ in the fault propagation model. While setting OR to 0.5, 0.2, or 0.05, we compare DR and FPR achievable with IHU, which does not take positive symptoms into account, and IHU+, which includes positive symptoms in the analysis.

As presented in Figs. 1(a) and 1(b), including positive symptoms in the process of fault localization allows DR to be significantly increased and FPR to be significantly decreased. Overall, thanks to the positive information, the fault localization accuracy improves. The smaller OR, the bigger the improvement. Parameter OR determines the system instrumentation level defined as the average number of symptoms that may be observed per fault. (Note that the average number of symptoms in the system is a squared function of n, thus the system instrumentation level naturally improves when n increases.) It may be concluded that, in poorly instrumented systems, positive symptoms may be effectively used to improve the accuracy of the fault localization process without worsening its performance.



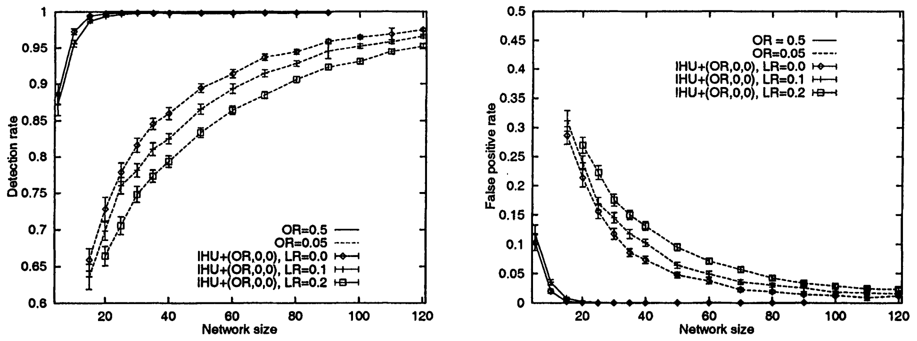
(a) Detection rate (b) False positive rate

Figure 1. Accuracy achievable with algorithms IHU and IHU+ for various ORs.

5.2 The impact of ignoring symptom loss on the accuracy of fault localization

To isolate the impact of symptom loss on the accuracy of fault localization, we set SSR $= 0$, and vary LR from 0.0 to 0.2. In the fault propagation model, we use $p_{\text{loss}} = 0$, and $p_s = 0$. (The fault localization algorithm effectively ignores the symptom loss.) We apply algorithm IHU+ to this model.

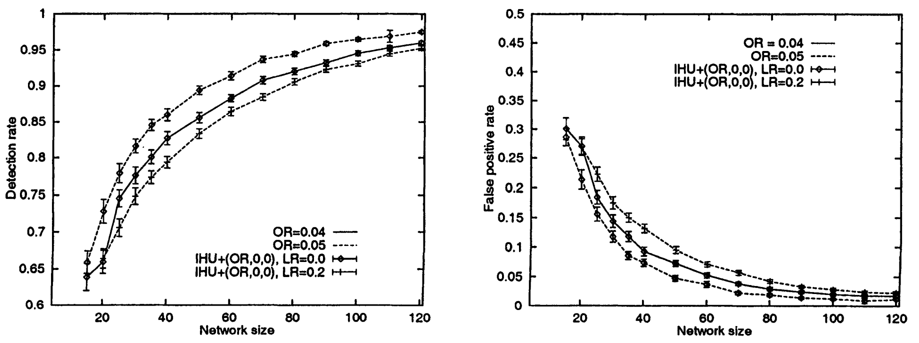
Symptom loss, when ignored by the fault localization process, does indeed decrease its accuracy: we observe a decrease of DR (Fig. 2(a)) and an increase of FPR (Fig. 2(b)). The strength of the symptom-loss impact on the accuracy is related to the value of LR and the system instrumentation level. Nonetheless, the decrease of accuracy caused by symptom loss is small (within 5% for both DR and FPR), which allows us to conclude that IHU+ is resilient to symptom loss even when it relies on positive information to perform fault diagnosis and does not include the explicit representation of lost symptoms in its model. To determine whether including this representation may improve the fault localization accuracy, we observe that decreasing accuracy when symptoms may be lost is due to two factors: (1) fewer symptoms are observed and therefore the system instrumentation level perceived by the fault management application decreases, and (2) some symptoms are incorrectly interpreted as positive ones. The relative contribution of these two factors determines the upper bound on the possible increase in the accuracy resulting from including symptom loss in the fault propagation model. Observe that the impact of only the second factor may be alleviated by including the representation of symptom loss in the model.



(a) Detection rate (b) False positive rate

Figure 2. The impact of symptom loss on the accuracy for various ORs and LRs.

To estimate the relative impact of factors (1) and (2), we perform another experiment. We execute the simulation study using the following parameters of the simulation model: (1) OR $= 0.05$, LR $= 0.0$, (2) OR $= 0.05$, LR $= 0.2$, and (3) OR $= 0.04$, LR $= 0.0$. The amount of information provided to the fault localization algorithm in the second and third cases is the same, because 0.05(1-0.2)=0.04. Thus the difference between the accuracies observed in the first and second cases represents the impact of factor (1). The difference between the accuracies observed in the second and third cases represents the impact of factor (2). As shown in Figs 3(a) and 3(b) the overall decrease of accuracy due to symptom loss is split evenly between the two factors. This lets us conclude that, were symptom loss represented in the fault propagation model, the resulting improvement in accuracy could not be greater than 2-2.5%. Indeed, our experiments with a fault propagation model using $p_{\text{loss}}(s_i) = 0.2$ did not reveal any statistically provable improvement in accuracy. With higher values of LR, some small improvement in accuracy has been observed.



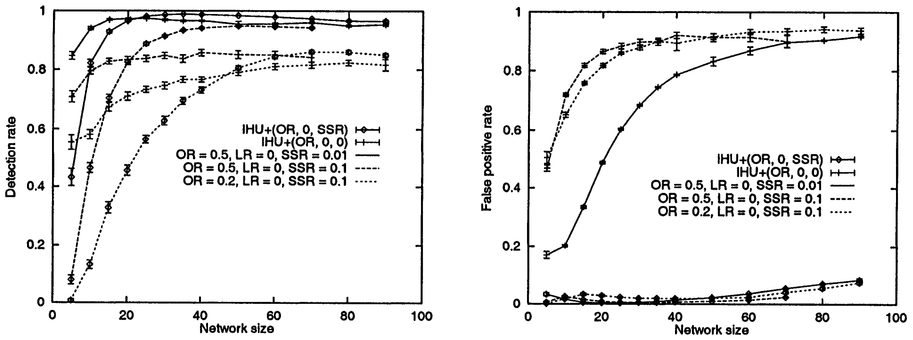
(a) Detection rate (b) False positive rate

Figure 3. The impact of factors (1) and (2) in system with OR $= 0.05$ and OR $= 0.2$.

This simulation study assumes that all symptoms are equally likely to be lost, while in reality $p_{\text{loss}}(s_i)$ is different for different symptoms, e.g., when symptom are transmitted in-band. We expect that when the symptom-loss probabilities are not equal, the benefit of including symptom loss in the analysis would be more evident.

5.3 *The impact of analyzing spurious symptoms*

The impact of including spurious symptoms in the fault localization process is evaluated by applying IHU+ to fault propagation models using $p_s(s_i) = 0$ and $p_s(s_i) =$ SSR, respectively. We vary OR between 0.5 and 0.2, and use SSR of 0.01 and 0.1. As shown in Fig. 4(a), the inclusion of spurious symptoms in the fault localization process in small networks decreases DR. This is explained by the fact that in poorly instrumented networks only a few symptoms are available to the fault localization process. When the possibility of spurious symptoms is taken into account, and the amount of available evidence is small, the algorithm concludes that there is no sufficient evidential support for the existence of faults, and considers all the observed symptoms spurious. Otherwise, DR would be higher (Fig. 4(a)) but FPR would be very high as well (Fig. 4(b)). When system instrumentation improves, so does the DR of IHU+ with an accurate representation of spurious symptoms in the fault propagation model. Overall, we conclude that including spurious symptoms in the fault propagation model has a big impact on the accuracy of the fault localization algorithm. However, to take full advantage of this capability, the system instrumentation level should be increased correspondingly to the rate with which spurious symptoms are generated.



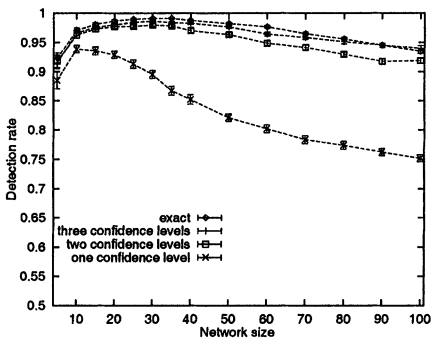
(a) Detection rate (b) False positive rate

Figure 4. The change of accuracy as a result of spurious symptoms analysis.

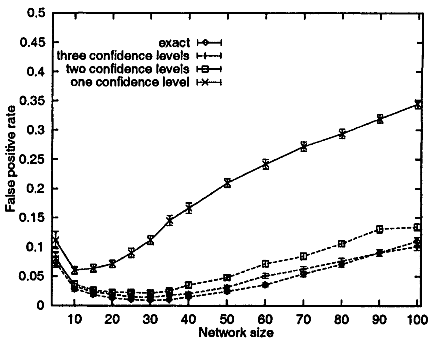
5.4 *The impact of conditional probability estimation errors*

In the final set of experiments we evaluate the impact of conditional probability estimation errors on the fault localization accuracy. We consider a scenario in which instead of the accurate conditional probability values, a small number of confidence levels, c, are being used. To represent the real-life probability p, the model uses the i^{th} confidence level, where $i = \lfloor pc \rfloor$. Thus, effectively, real-system probability p is mapped into propagation-model weight $\frac{\lfloor pc \rfloor}{c} + \frac{1}{2c}$. The creation of the probability model by a human is feasible, if high fault-localization accuracy may be achieved even when only a small number of confidence levels is used.

Fig. 5(a) and 5(b) compare the DR and FPR of Algorithm IHU having exact knowledge of the probability distribution with the DR and FPR achieved using one, two, and three confidence levels for various observability ratios. The figures prove an important property of the algorithm presented in this paper: it allows the expert to use a small set of meaningful qualitative probability assignments such as *unlikely*, *possible*, and *likely*, rather than exact probabilities, while preserving very high accuracy.



(a) Detection rate



(b) False positive rate

Figure 5. Accuracy for various granularities of confidence levels.

6. Conclusion

The technique proposed in this paper isolates the most probable set of faults through incremental updating of the symptom explanation hypothesis. It uses a probabilistic model, which makes the technique applicable to systems with a high degree of non-determinism. While assuming the pre-existence of such a model, the technique is robust against the model's imperfection. As shown in the simulation study, the technique offers high accuracy, even in the presence of observation noise. It also has low polynomial complexity. When applied to the problem of end-to-end service failure diagnosis, our implementation of the technique solves multi-fault scenarios in networks composed of more than 100 routers or bridges within less that 10 seconds.

Since fault localization is not a new problem and many fault localization techniques have already been proposed, it is important to consider comparing these techniques with respect to their accuracy and performance. Unfortunately, the techniques proposed in the literature [4, 7, 10, 15–17] that are suitable for bipartite models differ with respect to assumptions they are based on, capabilities, and problems they aim at addressing. Some of the properties that distinguish IHU from the previously published approaches are introduced in Section 1. The different assumptions and capabilities render the techniques difficult to compare in quantitative terms as they make any such comparison inherently unfair. A set of objective criteria that allow the comparison to be performed have yet to be identified, which is an interesting future research problem.

Nevertheless, IHU may be compared to our previously investigated fault localization approach, which is based on belief updating in belief networks [15, 16]. The belief-network approach is more flexible as it does not constrain the shape of a fault propagation model to a bipartite one, but it is not incremental and its computational complexity, even in bipartite models, is higher. Thus, while the belief-network approach offers similar accuracy and resilience to model imperfections and observation noise as IHU, its scalability is significantly lower.

Some of the observations made in the simulation study presented in this paper, e.g., the dependence of the benefit resulting from positive symptoms analysis on the system instrumentation level or necessity to increase system instrumentation level in systems with high spurious symptoms rates, are rather natural and could have been anticipated. Our study allows these observations to be quantified. Since similar results have also been obtained in the analogous study on the belief-network approach [15], we believe these results apply to the fault localization problem in general.

Future work will include designing a distributed version of the algorithm, which explores the domain semantics of management systems. In the application to end-to-end service failure diagnosis, the distributed technique will follow the initial ideas presented in [14]. The algorithm presented in this paper assumes that alternative causes of the same event should be combined using logical OR. It will be extended to allow other models such as AND or NOT models. [2]

References

[1] K. Appleby, G. Goldszmidt, and M. Steinder. Yemanja–a layered event correlation system for multi-domain computing utilities. *Journal of Network and Systems Management*, 10(2):171–194, 2002.

[2] A. T. Bouloutas, S. Calo, and A. Finkel. Alarm correlation and fault identification in communication networks. *IEEE Transactions on Communications*, 42(2/3/4):523–533, 1994.

[3] J. D. Case, K. McCloghrie, M. T. Rose, and S. Waldbusser. *Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)*. IETF Network Working Group, 1996. RFC 1905.

[4] C. S. Chao, D. L. Yang, and A. C. Liu. An automated fault diagnosis system using hierarchical reasoning and alarm correlation. *Journal of Network and Systems Management*, 9(2):183–202, 2001.

[5] R. H. Deng, A. A. Lazar, and W. Wang. A probabilistic approach to fault diagnosis in linear lightwave networks. In *Integrated Network Management III*, pp. 697–708. Apr. 1993.

[6] A. Dupuy, J. Schwartz, Y. Yemini, G. Barzilai, and A. Cahana. Network fault management: A user's view. In *Integrated Network Management I*, pp. 101–107. May 1989.

[7] M. Fecko and M. Steinder. Combinatorial designs in multiple faults localization for battlefield networks. In *IEEE Military Commun. Conf. (MILCOM)*, McLean, VA, 2001.

[8] P. Hong and P. Sen. Incorporating non-deterministic reasoning in managing heterogeneous network faults. In *Integrated Network Management II*, pp. 481–492. Apr. 1991.

[9] G. Jakobson and M. D. Weissman. Alarm correlation. *IEEE Network*, 7(6):52–59, Nov. 1993.

[10] I. Katzela and M. Schwartz. Schemes for fault identification in communication networks. *IEEE Transactions on Networking*, 3(6):733–764, 1995.

[11] S. Kliger, S. Yemini, Y. Yemini, D. Ohsie, and S. Stolfo. A coding approach to event correlation. In *Integrated Network Management IV*, pp. 266–277. May 1995.

[12] R. Perlman. *Interconnections, Second Edition: Bridges, Routers, Switches, and Internetworking Protocols*. Addison Wesley, 1999.

[13] M. Steinder and A. S. Sethi. Non-deterministic diagnosis of end-to-end service failures in a multi-layer communication system. In *Proc. of ICCCN*, pp. 374–379, Scottsdale, AZ, 2001.

[14] M. Steinder and A. S. Sethi. Distributed fault localization in hierarchically routed networks. In *Int'l Wksp on Distributed Systems: Operations and Management*, pp. 195–207, Montreal, QC, Oct. 2002.

[15] M. Steinder and A. S. Sethi. Increasing robustness of fault localization through analysis of lost, spurious, and positive symptoms. In *Proc. of IEEE INFOCOM*, New York, NY, 2002.

[16] M. Steinder and A.S. Sethi. End-to-end service failure diagnosis using belief networks. In *Proc. Network Operation and Management Symposium*, pp. 375–390, Florence, Italy, Apr. 2002.

[17] S. A. Yemini, S. Kliger, E. Mozes, Y. Yemini, and D. Ohsie. High speed and robust event correlation. *IEEE Communications Magazine*, 34(5):82–90, 1996.

[2]The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied of the Army Research Lab or the U.S. Government.

HIERARCHICAL END-TO-END SERVICE RECOVERY

Mohamed El-Darieby[1], Dorina Petriu[1] and Jerry Rolia[2]
[1]Systems and Computer Engineering Dept., Carleton University, Ottawa, ON, Canada K1S5B6 {mdarieby, petriu}@sce.carleton.ca

[2]Internet Systems and Storage Lab., Hewlett-Packard Labs, Palo Alto, CA, 94304, USA jar@hpl.hp.com

Abstract: Failed networks, for example MPLS, can cause signaling storms the size of which can grow dramatically with network size. This paper presents a new scalable fault notification protocol that reduces the size of this storm. The protocol causes failure notification signals to travel vertically up and down a network hierarchy instead of horizontally along the service routes. This reduces the signaling required for longer paths and provides signaling scalability for larger network sizes. The protocol also alleviates the number of fault notification signals by hierarchically aggregating them. Aggregation is based on path management information maintained in the network hierarchy. We show how the protocol reduces the size of the signaling storm analytically and with simulation. However, this comes at the price of increased storage overheads when compared with existing restoration techniques.

Keywords: Service recovery, Hierarchical networks, Fault notification, MPLS

1. INTRODUCTION

The efficient provisioning of network services is a competitive differentiator among network Service Providers (SP). A SP has to ensure service survivability by recovering network failures in a responsive manner. In general, end-to-end service recovery requires the notification of ingress network nodes about such failures. Fault notification constitutes a major part of recovery time and is a fundamental criterion for evaluating the performance of the recovery process [3]. It results in flooding the network with recovery signals (a.k.a. signaling storm [2]). The magnitude of the signaling storm grows dramatically with network size because 1) services are expected to span larger numbers of network nodes as the network size grows; and 2) the number of services carried over an individual link grows proportionally as the transport technology matures (e.g. OC 92 versus OC 3 links).

In this paper, we propose a new Scalable end-to-end Path Recovery Protocol (SPRP) that addresses scalability aspects of the fault notification problem. SPRP works in conjunction with a service creation protocol called the Hierarchical Distributed Routing Protocol (HDRP) [9]. Both HDRP and SPRP use a standard hierarchical network structure (e.g. the ATM PNNI). However, in SPRP and HDRP, hierarchical nodes/ (Bandwidth Brokers) participate in service provisioning (routing, signaling & fault notification), whereas Peer Group Leaders in PNNI only reflect static network connectivity information and do not participate in fault notification.

To provide signaling scalability for longer paths, failure notification signaling messages travel vertically up and down the network hierarchy instead of horizontally along the routes traversed by the paths. This is enabled by the path state information maintained at nodes at higher levels of the hierarchy by HDRP.

SPRP alleviates the number of fault notification signals by aggregating signaling messages based on the physical proximity of ingress routers. The protocol uses path management information maintained in the network hierarchy to aggregate the signals. A single notification message will be raised per path aggregate (instead of one per individual path).

We evaluate analytically the performance of SPRP and compare it to existing fault notification approaches. Analytical analysis gives insights into why and how SPRP reduces the size of the signaling storm compared to these approaches. The simulation experiments we conducted confirm these analytical results.

To the best of our knowledge, this work is the first to propose hierarchical end-to-end service recovery. We present SPRP in the context of Multi-Protocol Label Switching (MPLS) [5, 13] because MPLS is the lowest layer of the network architecture that has knowledge of end-to-end paths. This implies that the fastest end-to-end path recovery is achievable at this layer. In addition, MPLS is becoming the *de facto* standard for controlling many IP-based network transport infrastructures [2]. An MPLS service is a path/ route that traverses many networks.

This paper is organized as follows: In section 2, we describe the proposed recovery protocol, and in the following section, we relate our protocol with other research and standards. Section 4 provides analytical analysis and simulation results regarding the overhead of SPRP and compares it to other schemes. Conclusions are offered in the last section.

2. HIERARCHICAL SERVICE RECOVERY

This section describes our approach to hierarchical networks, the Scalable Path Recovery Protocol and introduces signaling aggregation to enhance its performance.

A hierarchical network is a traditional solution to the network-scaling problem [12]. Nodes are organized into different interconnected sub-networks called Autonomous Systems (AS)/ domains. An AS consists of a number of interconnected network nodes (i.e. IP routers or ATM switches). We assume Service Level Agreements are in place among ASs. A Bandwidth Broker (BB) manages and maintains topological and state information about the nodes and links of an AS. BBs are connected to each other and to physical nodes with fault tolerant connections.

The process of grouping BBs (at one hierarchy level) into logical ASs and abstracting such ASs via a BB (at the next higher level) is done at all levels of the hierarchy (see Figure 1). The physical nodes of the network belong to level 1, and their managing BBs to level 2. Level-2 BBs are interconnected in a manner that corresponds to the interconnectivity of level-1 ASs. Level-1 BBs are grouped into logical level-2 ASs. Each level-2 AS is represented and managed in turn by a level-3 BB, which maintain topology and state information about level-2 ASs, and so on. This hierarchy is assumed to be static. Fault-tolerant signaling channels between different BBs (same-level and child-parent) are also assumed. Internet standards (e.g. OSPF and BGP) propose a two-level hierarchy. The ATM PNNI standard supports a hierarchy of up to 105 levels [12].

BBs are software processes that can run on network nodes or on separate server nodes. We assume the latter approach in accordance with recent directions on the surveillance and control of the MPLS/TE-based networks that suggest the use of a centralized resource manager (i.e. BB) within each Autonomous system [15, 16]. Server nodes are cluster-based server farms that can grow in capacity based on the intensity of route calculation requests. Server nodes are implemented in branch offices or Internet Data Centers. As a result, storage and communications overhead are not a system concern.

2.1 The Scalable Path Recovery Protocol (SPRP)

SPRP solves the problem of very long paths by signaling vertically, instead of horizontally, up and down the network hierarchy. We assume that SPRP messages have highest priorities. Figure 1 gives an example of SPRP signaling. The SPRP algorithm is shown in Figure 2. We use the following notation:

- v_{ik}: a network node or a BB with address i, at level k. Physical nodes have $k = 1$ and BBs have $k > 1$
- *pid:* (ID, v_{sk}): a pair that uniquely identifies a path
- $Parent\_BB_k(v_{i1})$: a level k BB that has a view of node v_{i1}
- P_{ik}: $(i, v_{sk}, v_{ak}, v_{bk}, \ldots v_{tk})$: a level k path with *pid*= i that has v_{sk} as ingress node, v_{tk} as egress node
- $Plist_k$: list of *pids* of failed level k paths
- $AggP_{jk}$: $(j, v_{jk}, v_{(j-1)k}, \ldots v_{sk}, Plist_k)$: a level k aggregate route that aggregates a set of $AP_{i(k-1)}$ with *pids* in $Plist_k$ and that traverses the route listed

We use the "." notation to refer to members of a data structure. For example, to refer to the ingress router for path P_{ik}, we use $P_{ik}.Ingress$.

SPRP is executed in response to a failure detected by a node, v_{fl}. A *Failure* message is sent vertically up the hierarchy that contains a list of *pid*'s and ingress routes of paths affected by the failure. As a *Parent_BB*$_k$ (v_{fl}) recieves a *Failure* message, it scans the list of affected paths and if it views P_{jk}. Ingress where $P_{jk} \in Plist_k$, it notifies its child BB that views this ingress router. This is accomplished by sending a *FailureNotify* message down the hierarchy. The entry for P_{jk} is then removed from $Plist_k$. Then, *Parent_BB*$_k$ (v_{fl}) sends a *Failure* message up the hierarchy to its parent BB (i.e. *Parent_BB*$_{k+1}$$(v_{fl})$).

This is repeated up the hierarchy until *Parent_BB*$_{L+1}$$(v_{fl})$, which we call *RootBB*, is reached. *RootBB* is defined to have a view of *all* ingress routers of the paths that belong to $Plist_L$. *RootBB* sends a *FailureNotify* message down the hierarchy to each *Parent_BB*$_L$$(P_{tL}.ingress)$ where $P_{tL} \in Plist_L$.

A *Parent_BB*$_k$ $(P_{jk}.ingress)$ that receives a *FailureNotify* message forwards it down the hierarchy to notify *Parent_BB*$_{k-1}$ $(P_{jk}.ingress)$ about the failure in P_{jk}. This is repeated down the hierarchy till the *FailureNotify* message reaches $P_{jk}.ingress$. This completes the notification process for this path.

As an example, Figure 1 shows three uni-directional paths that are affected by a failure in the link connecting nodes 2.1.2 and 2.3.2. The paths follow the thick lines at level 1. The ingress and egress nodes of the three paths are as follows: 2.3.1 and 1.4.1 for path #1, 2.1.5 and 2.2.3 for path #2 and 4.3.4 and 2.2.1 for path #3. The master node for the failed link (node 2.1.2) sends *Failure* message (#1 in Figure) to BB 2.1 that contains the *pids* of the affected paths (#1, 2 and 3). BB 2.1 scans the list of ingress nodes of failed paths looking for the nodes that it has a view of. It sends a *FailureNotify* message (#2) to node 2.1.5 about the failure in path #2 and removes the *pid* for path #2 from the list of affected paths. As the list is not empty yet, BB 2.1 sends a *Failure* message (#3) to BB 2. BB 2 finds the ingress node of path #3 to be within its view, so it sends *FailureNotify* message (#4) to BB 2.3 and removes path #3 from the list. BB 2.3, in turn, sends a *FailureNotify* message (#5) to physical node 2.3.1. Now, BB 2 realizes that some paths (i.e. #1) are out of its view, so it sends a *Failure* message (#6) to BB *. BB * has a view of node 4.2.1 so it sends a *FailureNotify* message (#7) to BB 4. BB 4 notifies BB 4.3 that in turn notifies node 4.3.4 of the failure (message #8 and #9). Now, BB * realizes there are no path left to be processed and it stops forwarding messages up the hierarchy.

The number of SPRP vertical notification messages required for each affected path will be $2*(K-1)$ where the BB that has a view of the failure master node and the ingress router of the affected path is at level K. One extreme case (e.g. path #2) is if the two nodes are in the same network domain then *RootBB* is at level 2 and the number of notification messages is 2. The other extreme (e.g. path #1) is where *RootBB* coincides with the root of the network hierarchy (level 4) where the number of notification messages will be 6 messages.

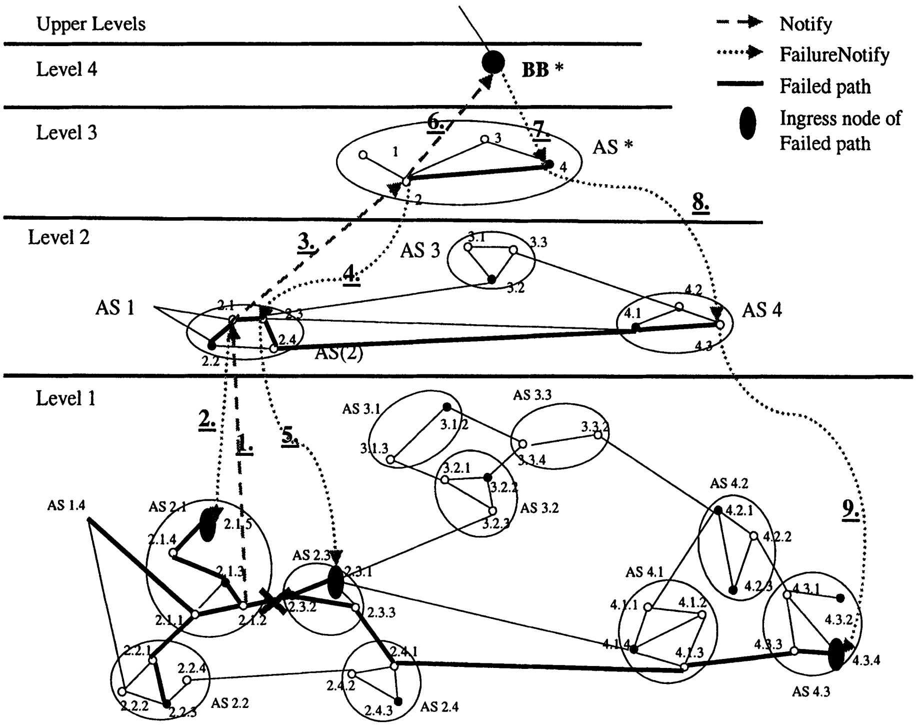


Figure 1. The Hierarchical Restoration Protocol

2.2 SPRP with Signaling Aggregation

To enhance the performance of SPRP, we propose to take advantage of the physical proximity of the ingress routers of some of the affected paths, such that one notification message is sent for them instead of a message for each path. This exploits path management information maintained in the network hierarchy by HDRP.

Information about individual paths that traverse the same route and that have ingress routers within a network domain can be aggregated. Aggregated information describes an *aggregate path* such that network nodes along this route identify the set of individual paths (the aggregate path) by a single *pid*. The concept of an aggregate path resembles that of Aggregation Areas [11]. An individual path is a (non-aggregated) physical or abstract path. HDRP implements path aggregation at all levels of the network hierarchy to reduce information storage overheads.

Aggregation is recursive. An aggregate path may aggregate a number of aggregate paths, each of whom aggregates in turn a number of lower level (aggregate or individual) paths. In other words an aggregated path is a rooted hierarchy (tree) of aggregate and individual paths.

```
Switch ($msg \rightarrow type$){
  Case Failure ($FailedPlist_{(l-1)}$) {
    Initialize $FailedPlist_l$ = NULL;
    For (each path $P_{kl} \in FailedPlist_{(l-1)}$){
    If ($v_{il}$ is $Parent\_BB$ ($P_{kl}.v_{s1}$))
        Send ($FailureNotify$, $P_{kl}.v_{s1}$, $FailedP_{i(l-1)}$);
    Else{
      If ($P_{kl} \in AggP_{jl}$) {
        If ($AggP_{jl} \notin FailedPlist_l$)
              $FailedPlist_l$ += {$AggP_{kl}$};
              $FailedPlist_l \rightarrow AggP_{kl} \rightarrow Plist_l$+= $P_{kl}$
      }
      Else $FailedPlist_l$+={AP($P_{kl}$);}
      }
    }
    Send ($Failure$, $Parent\_BB$ ($v_{il}$), $FailedPlist_l$);
  }/*end case*/

  Case FailureNotify ($FailedPlist_{(l+1)}$) {
    If ($v_{il}$ is a physical node) Switchover & stop();
    For (each path $P_{kl} \in FailedPlist_{(l+1)}$){
      If ($P_{kl}$ is an Aggregate path){
        For (each $P_{jl} \in P_{kl}$)
          Send ($FailureNotify$, $Parent\_BB(P_{jl}.v_{s1})$, $P_{jl}.list$)
          }
      Else
        Send ($FailureNotify$, $Parent\_BB(P_{kl}.v_{s1})$, $P_{jl}.list$);
    }
  }/*end case*/
}
```

Figure 2. The SPRP algorithm

SPRP uses aggregate paths to reduce the size of the signaling storm. SPRP raises *one* notification message for each aggregate path instead of a notification message per each individual path. The message contains the *pid* of the aggregate path rather than *pids* of individual paths.

A scenario for SPRP with signaling aggregation is as follows: as node v_{f1} detects a network failure, it sends a *Failure* message up the hierarchy to its parent BBs. $Parent\_BB_k(v_{f1})$ applies the SPRP algorithm with an extra step: $parent\_BB_k(v_{f1})$ composes a new list of affected paths replacing individual component paths with their aggregate path. If an individual path is not to be aggregated with other paths, it is added to the list as an individual path. Then, $Parent\_BB_k(v_{f1})$ forwards the *Failure* message to $Parent\_BB_{[k+1]}(v_{i1})$. This is repeated up the hierarchy until *RootBB* has been notified of the failure. *RootBB* sends *FailureNotify* messages to its child BBs that have these ingress routers within their views.

A BB_{ik} that receives a *FailureNotify* message has a view of the ingress routers of the failed path (aggregate or individual) in the message. In case of an aggregate path, BB_{ik} deaggregates it into its children paths. A *FailureNotify* message is sent per each child path down the hierarchy. In the case of an individual route, only one *FailureNotify* message is sent down the hierarchy. This is repeated down the hierarchy until all the ingress routers have been notified of the failure.

For example, Figure 3 shows 4 paths following the thick lines at level 1. The ingress and egress nodes of the three paths are as follows: 4.3.4 and 2.2.1 for path #1, 4.3.2 and 2.2.1 for path #2, 4.2.1 and 2.2.1 for path #3, and 4 2.1.5 and 1.4.1 for path #4. As the physical link connecting node 2.1.2 and node 2.3.2 fails, node 2.1.2 sends a *Failure* message (#1,in Fig. 3) to BB 2.1. BB 2.1 realizes that the ingress for path #4 is within its view. So, it sends a *FailureNotify* message (# 2) to the ingress of that path (node 2.1.5) and removes its entry from the list. BB 2.1 also realizes that paths #1 and #2 have the same ingress router at level 1. So, it replaces the entries for paths #1 and #2 with a signle entry for their aggregate path (which we call $AggP_{12}$). Then, it sends a *Failure* message (#3) to BB 2 with the new list of affected paths. BB 2 finds it has no view of ingress routers of the paths in the list. BB 2 realizes that $AggP_{12}$ and path #3 have the same ingress router at level 2, so it removes replaces their *pid*'s with the *pid* of their aggregate path ($AggP_{13}$). Then, it sends a *Failure* message (#4) to BB *. BB * has a view of the ingress routers of the aggregate path. It sends a *FailureNotify* message (#5) to BB 4 that views the ingress nodes of $AggP_{13}$. BB 4 decomposes the aggregate path into its component paths (which are $AggP_{12}$ and path #3). It sends *FailureNotify* messages to BB 4.2 about the failure in path #3 and to BB 4.3 about the failure in $AggP_{12}$. BB 4.2 sends a *FailureNotify* message to node 4.2.1 about the failure in path #3. BB 4.3 notifies the ingress routers of the different component paths of $AggP_{12}$. That is, it sends a *FailureNotify* messages to nodes 4.3.4 and 4.3.2. This completes the failure notification process.

3. RELATED WORK

Network recovery has two basic models [11]: 1) *restoration* where a recovery path is established on demand as the network nodes are notified with the fault; and 2) *protection switching* where the recovery path is established prior to the occurrence of the fault. The scope of the recovery process may span a single network link, a segment (a set of links) of a path or an end-to-end path.

MPLS path recovery involves the following processes [13]: working and recovery path setup, fault detection and notification, switchover from the working path to the recovery path, fault repair [8] and a switchback from the recovery path to the working path.

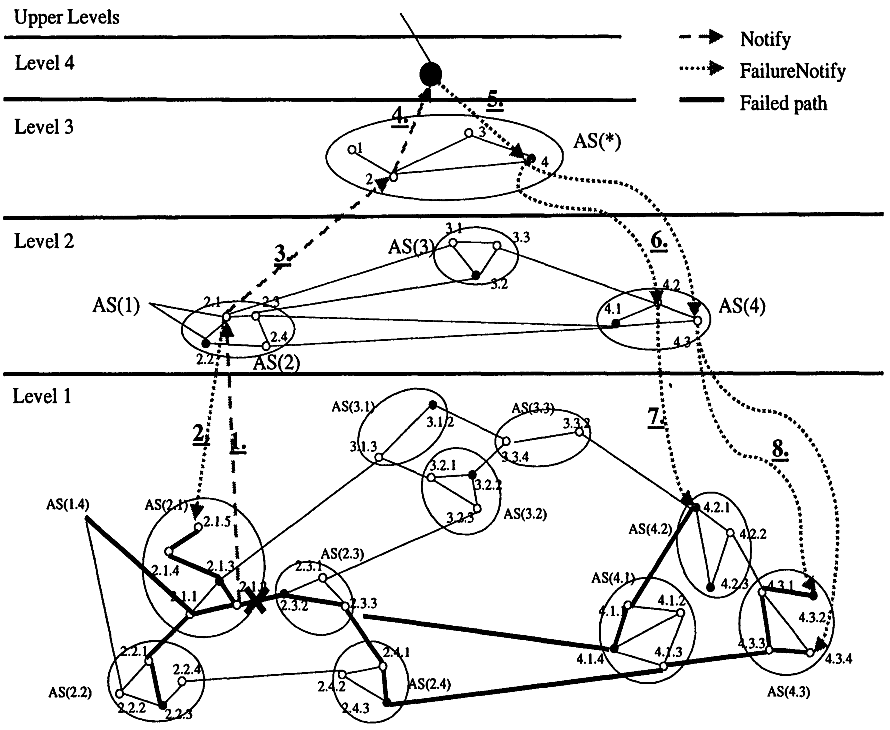


Figure 3. Hierarchical Service Recovery Signaling

In [1], a signaling protocol for fast restoration in optical networks is proposed. The protocol may be used for span and end-to-end restoration. The end-to-end restoration mechanism is as follows: A *failure indication* message is sent by the master node of the failed link along the working path towards the ingress node of the path. This message is transmitted periodically every 90 ms until a *Failure acknowledgment* message is received from the ingress node. Then, the ingress node sends a *switchover request message* towards the egress router of the path along the recovery path. This message is transmitted periodically every 90 ms until a *switchover response message* is received from the egress node. The authors define the data structures maintained at different nodes, and the general format for the used messages.

Existing MPLS path restoration schemes [1, 11] require the master node of the failed link to re-transmit failure notification signals periodically if they do not receive an acknowledgement from the ingress node of the MPLS path. Consequently, as an ingress node for a path receives a failure notification message, it acknowledges that through sending a *FailureAck* message. In SPRP, the *FailureAck* message travels exactly the same route as the failure notification message but in the reverse direction.

In [5], a "scalable and bandwidth efficient" path recovery mechanism for MPLS networks is proposed. The mechanism does not require acknowledgements or handshaking because it is based on the periodical transmittal of failure notification messages until the "switching over" nodes learn of the failure. Specifically, the authors focus on fault notification for aggregated/ (merged) MPLS paths. The mechanism is scalable because it builds a point-to-multipoint *Reverse Notification Tree (RNT)* that is the exact mirror of the aggregated working paths. Due to the sending of only one signaling message along the shared segments of the RNT, the mechanism enables a reduction in the signaling overhead. We evaluate this method and compare it to SPRP in the following section.

4. EVALUATION

In this section we evaluate analytically the performance of SPRP and compare it to existing fault notification approaches. Using multicast is a price to pay for faster fault notification. This is not exclusive to SPRP (see RNT described in section 3). Then we present the results of our simulation experiments that evaluate how SPRP reduces the size of the fault notification tree compared to standard approaches [13].

The analytical comparison is based on the power-law relationship [4] that relates the total number of multicast links and the average unicast path length in terms of the number of receivers:

$$L(m) \approx \mu . \mathrm{m}^{k} \qquad (1)$$

where *L(m)* is the total number of multicast links in the multicast tree, μ is the average path length between any two nodes, *m* is the number of receivers in the tree (which represents the number of ingress routers to be notified) and $k \approx 0.8$ for a number of real life and generated topologies.

Equation (1) characterizes the reduction in the size of the signaling storm achieved by the RNT [5] approach that builds a multicast tree as compared to unicast approach [1].

SPRP hierarchically unicast fault notification signals up and down the hierarchy. SPRP with aggregation builds a number of multicast trees to perform fault notification. Each multicast tree is rooted at a BB that has a view of the master node of the failed link and an ingress router. The multi cast tree has the ingress routers of the affected paths as its leaves. Equation (1) characterizes the effect of aggregation on SPRP. Aggregating singling messages takes advantage of the savings multicast achieves as compared to the unicast case. This comes at the non-trivial cost of implementing multicast. In SPRP, μ_{SPRP} is on average equal to *L*, which is the number of levels in the network hierarchy.

SPRP with aggregation further reduces the size of the signaling storm with respect to RNT. This is because SPRP has much smaller μ than RNT. We can estimate μ_{RNT} as the effective diameter of the network, which gives the average number of hops between any two nodes in the network with high probability (about 80%). According to [10], the effective diameter of an *N*-nodes network (i.e. μ_{RNT}) is:

$$\mu_{RNT} = \left(\frac{N^2}{N + 2E} \right)^{1/H} \qquad (2)$$

where H is the hop-plot exponent which is a constant value for the graph. For inter-domain data sets, the average value for H is 4.7 [10]. E is the number of edges in the graph and can be estimated by:

$$E(N) = \frac{N}{2(R+1)} \left(1 - \frac{1}{N^{R+1}} \right) \qquad (3)$$

μ_{RNT} depends on network size. We choose R = -0.80 which is the value for many Internet topologies discussed in [10]). Substituting from (3) into (2) with the values given above, then μ_{RNT} can be roughly approximated by:

$$\mu_{RNT} \approx \left(\frac{0.2 * N}{1.2 - N^{-0.2}} \right)^{1/4.7} \qquad (4)$$

For our simulation experiments, we used the OMNet++ simulator [14]. We developed simulation models for different topologies of the network and for different hierarchy structures. We considered networks of 256 and 1296 nodes and hierarchies of 3 and 5 levels. The toplogies are described in Table I.

TABLE I. DIMENSIONS FOR NETWORK HIERARCHY STUDIED

Hierarchy	H_1	H_2	H_3	H_4
Network Size	256	256	1296	1296
Number of links E given by (1)	53464	53464	199517	199517
Number of levels $K+1$	3	5	3	5
Number of nodes per domain m	16	4	36	6
Number of phsyical domains	16	64	36	216

Path creation requests are assumed to arrive following a Poisson distribution. The source and destination of each connection is randomly chosen. 2000 requests are generated during each simulation run. At the end of each simulation run, network failures are generated on each node of the physical network that carries a specific number of paths. The number of SPRP messages caused by each of these failures is counted. Multiple runs are carried out and their results are averaged. We used the fully connected topology aggregation scheme when building the hierarchy.

Hierarchical message aggregation reduces the size of the fault notification-signaling storm. This is shown in Figures 4 and 5 that compare the average size of the storm for standard unicast approaches to its average size with SPRP. The size of the storm is characterized in terms of the number of affected paths. The Figures show the effect of the aggregation of signaling messages on the size of the signaling storm by showing the average size of the storm with no signaling aggregation to its size with signaling aggregation.

SPRP proves to be a scalable technique for failure notification. Hierarchical signaling aggregation results in small increases in the size of the signaling storm as the network size increase. For example, the average size of the signaling storm for 300 failed paths is 1310 and 1520 messages for hierarchies H_2 and H_4 (which have the same number of levels), respectively. This is relatively a small increase in the size of the signaling storm compared with the increase from 4300 to 6970 messages with the unicast approach (for the same network sizes and number of paths).

The deeper the hierarchy is, the larger the size of the signaling storm for the same network size and for the same service workload. Analytically, a deeper hierarchy has a larger μ. A deeper hierarchy also results in a smaller domain size and consequently the ingress routers of the affected paths will be more sparingly distributed among network domains. This results in larger number of smaller multicast trees built by SPRP. This alleviates the overhead of building and maintaining them as compared with RNT.

Figures 6 and 7 show the control overhead required for SPRP operation. The average number of BBs managing a path is shown in terms of the path length. The effect of different hierarchy structures on the control overhead is also shown. The required number of BBs to manage a path is a good indication for the overhead of SPRP. It reflects the amount of management data that has to be maintained in the hierarchy as a function of the length of the corresponding path.

For the same network size at the physical level, as the hierarchy grows deeper, the domain size becomes smaller, the number of physical domains traversed by a path of a given length is likely to become larger, and consequently a larger number of managing BBs is required.

For example, the domain sizes are 16 and 4 for hierarchies H_1 and H_2, respectively. In Fig. 6, a path of 20 nodes is managed by an average of 9.4 BBs in H_1, and of 15.5 BBs in H_2. The same argument applies for different networks considering hierarchies with the same depth (e.g. H_1 and H_3). To manage path of 20 nodes, H_3 requires an average of 5.6 BBs (in Fig. 7) as opposed to the 9.4 BBs required by H_1. H_1 and H_3 correspond to the two-tier architecture of the Internet.

5. CONCLUSIONS

We propose a new Scalable end-to-end Path Recovery Protocol (SPRP) that results in faster recovery times. In SPRP, signaling messages travel vertically up and down the network hierarchy instead of horizontally along the paths. This reduces the number of signaling messages to notify the ingress router of the path. The protocol uses path management information maintained in the network hierarchy to aggregate signals so that a single notification message would be raised per the path aggregate.

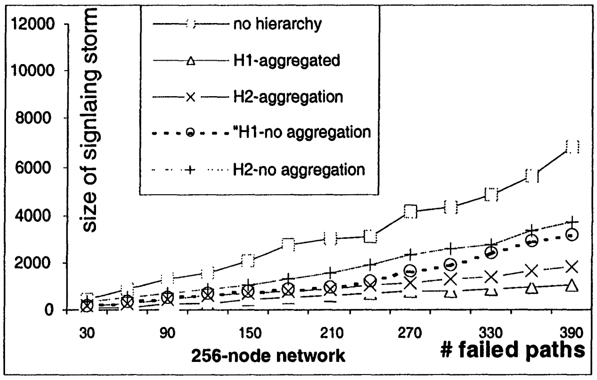


Figure 4. Average size of signaling storm in terms of path length

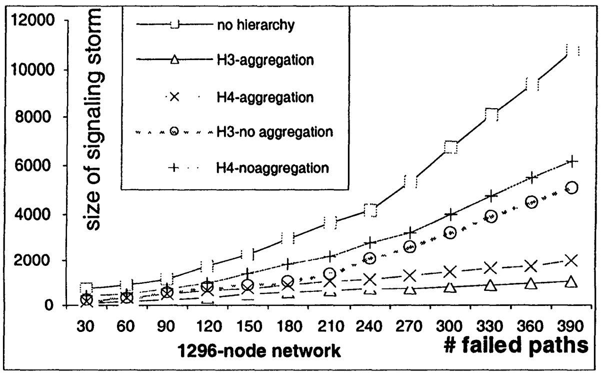


Figure 5. Average size of signaling storm in terms of path length

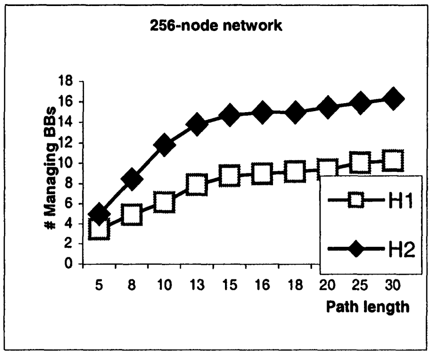


Figure 6. Average number of BBs managing a path in terms of path length

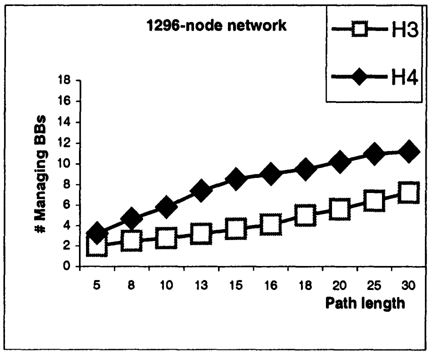


Figure 7: Average number of BBs managing a path in terms of path length

SPRP is a scalable technique for failure notification as the size of the network grows as shown by analytical analysis and simulation results. The performance of SPRP is a function of the number of levels in the network hierarchy while the performance of RNT depends on network size. The protocol provides imporved performance when compared with the RNT and standard unicast approaches. Hierarchical structure affects the size of the signaling storm. The deeper the hierarchy, the larger the size of the signaling storm for the same network size and for the same service workload. These advantages come at the cost of increased control overhead required in the network hierarchy.

REFERENCES

[1] D. Brungard et al, “Generalized MPLS Recovery Functional Specification,” IETF Draft, “Work In progress,” August 2002.

[2] G. Bernstein, et al, “Optical Inter Domain Routing Considerations,” IETF draft, “Work In progress,” Nov. 2001.

[3] G. Li, J. Yates, R. Doverspike, and D. Wang; “Experiments in Fast Restoration using GMPLS in Optical/Electronic Networks,” Optical Fiber Comm. Conf., March 2001.

[4] J. Chuang and M. Sirbu, “Pricing Multicast Communications: A Cost-Based Approach,” Telecommunication Systems 17: (3), July 2001.

[5] K Owens et al, “ A Path Protection/Restoration Mechanism for MPLS Networks,” IETF draft, “Work In progress,” July 2001.

[6] M. El-Darieby and J. Rolia, "Performance Modeling of a Service Provisioning Design," in Lecture Notes in Computer Science, Vol.1890, Springer Verlag, 2000, pp. 81-92.

[7] M. El-Darieby, D. Petriu, and J. Rolia, "Scalability Analysis of Virtual Network-Based Service Provisioning," In IFIP/IEEE Integrated network Management, Seattle, USA 2001.

[8] M. El-Darieby, and A. Biezschad, "Intelligent Mobile Agents: Towards Network Fault Automation" In IFIP/IEEE Integrated network Management, Boston, USA 1999.

[9] M. El-Darieby, D. Petriu, and J. Rolia, "HDRP: A Hierarchical Distributed Routing Protocl,” Technical Report-07-03, Systems and Computer Engineering Dept., Carleton Univesity, Submitted for publication.

[10] M. Faloutsos, P. Faloutsos and C. Faloutsos, “On Power-Law Relationships of the Internet Topology,” In ACM SIGCOMM’99, Boston, MA, USA, 1999.

[11] F. Baker, C. Iturralde, F. Faucheur and B. Davie, "Aggregation of RSVP for IPv4 and IPv6 reservations," Internet Draft “Work In progress,” <draft-ietf-issll-rsvp-aggr-02.txt>.

[12] R. Cohen, R. Emek and E. Felstine, "Framework for Multicast in Hierarchical Networks," In IEEE *INFOCOM*, Tel-Aviv, March 2000.

[13] V. Sharma et al, “Framework for MPLS-based Recovery,” Internet Draft, “Work In progress,” July 2001.

[14] The OMNET++ Simulator: http://www.hit.bme.hu/phd/vargaa/omnetpp.htm

[15] G. Ash, "Traffic Engineering & QoS Methods for IP-, ATM- & TDM based Multiservice Networks," Internet Draft, “Work In progress,” <draft-ash-te-qos-routing-01.txt>.

[16] S. Salsano et al, "Inter-domain QoS Signaling: the BGRP Plus Architecture," Internet Draft, “Work In progress,” <draft-salsano-bgrpp-arch-00.txt>.

SESSION 13

Power and Optical Management

Chair: Adarshpal Sethi
University of Delaware, USA

GMPLS FAULT MANAGEMENT AND ITS IMPACT ON SERVICE RESILIENCE DIFFERENTIATION

M. Brunner, C. Hullo
NEC Europe Ltd., Adenauerplatz 6, D-69115 Heidelberg, Germany, brunner@ccrle.nec.de
EANTC AG, Sprembergerstr. 6, D-12047 Berlin, Germany, hullo@eantc.de

Abstract: Generalized Multi-Protocol Label Switching (GMPLS) is currently under standardization. It basically reuses the MPLS control plane (IP routing and signaling) for various technologies such as fiber switching, DWDM, SONET, and packet MPLS. Since GMPLS runs in core networks, fault management is of major concern. However, fast fault recovery and backup capacity assignments are very expensive and not all customers need this or are willing to pay for it. Therefore, we propose in this paper to use several protection and bandwidth-sharing schemes on the same network in order to provide differentiated services in the resilience space. This means an operator can offer and provide several customized services. The service management system implementing the schemes is built on top of a GMPLS network management system developed in our Lab.

Key words: Optical Networks, Fault Management, Resilience, and Service Management

1. INTRODUCTION

Fault management aims at helping network operators automatically detecting and recovering from faults before human beings, and so before customers notice them. By localizing faults when they happen and improved speed of repair, they might save paying refunds against broken SLAs (Service Level Agreement).

Recently, a technology called Generalized Multi Protocol Label Switching (GMPLS) has showed up and is currently under standardization at the IETF [1]. The primary goal of GMPLS is reusing the MPLS [2] control plane, namely IP routing protocols and path setup protocols for different kinds of networks such as SONET, DWDM, and packet MPLS. Since these technologies operate mainly in backbone networks, fault management is of primary interest, as it ensures the reliability of data delivery.

On the other hand, fault management might be expensive in terms of allocating resources for failure cases and on mechanisms (hardware or software) for fast detection and restoration. So fault management has to respond to the trade-off between making the network robust and avoiding too much resources to be allocated for not directly paid use, but for fail-over cases.

Another key feature of GMPLS is the fast and dynamic provisioning of optical or packet paths using an IP/MPLS control plane. This implies that also the fault management issue needs to be concerned about the low provisioning times and the dynamic behavior of the network.

On the other hand various customers have various requirements and demands on the service availability. So it does not make sense to install only one mechanism for all customers. That's where differentiation of the service in terms of resilience is used.

In the literature, different possible schemes for fast restoration and protection switching for various technologies have been published ([3], [4], and [5]). However, none of them propose the approach of differentiation of services based on different protection and bandwidth sharing schemes in a dynamic service creation environment. In this paper, we propose to integrate different schemes into a single network, in order to offer various availability parameters for customers. The availability needs to be specified on the service request, and it must be provided by different means in the network.

The only work the authors are aware of proposing a similar scheme is [13]. However, they have looked into IP and MPLS restoration only. They do not consider the GMPLS case, where different technologies are controlled and managed by the same management system. Additionally, they have not implemented their scheme for real world hardware as we did. And we tried to apply the different random generated topologies, where they had one topology of their test network. However, they have been more extensive in simulating various algorithms without mentioning the results in the publication, whereas we have chosen only one algorithm.

2. GENERALIZED MULTI-PROTOCOL LABEL SWITCHING (GMPLS)

GMPLS is, as its name suggests, a generalization of MPLS. GMPLS first generalizes the control plane such that it is not only used for packet switched networks, but also for optical switched, TDM-based, and physical networks. This requires that the control-plane and data-plane are no longer only logically separated but might also be physically separated. Second, GMPLS extends the notion of a label in order to support multiple switching layers. For instance a label might be an optical wavelength number.

The major goal of GMPLS is to control optical backbones as flexibly and dynamically as IP backbones today. GMPLS intend to reuse existing technologies by combining MPLS control technology namely RSVP-TE and CR-LDP [6] with operational experience of IP routing with some extensions towards GMPLS in order to provide a general network solution.

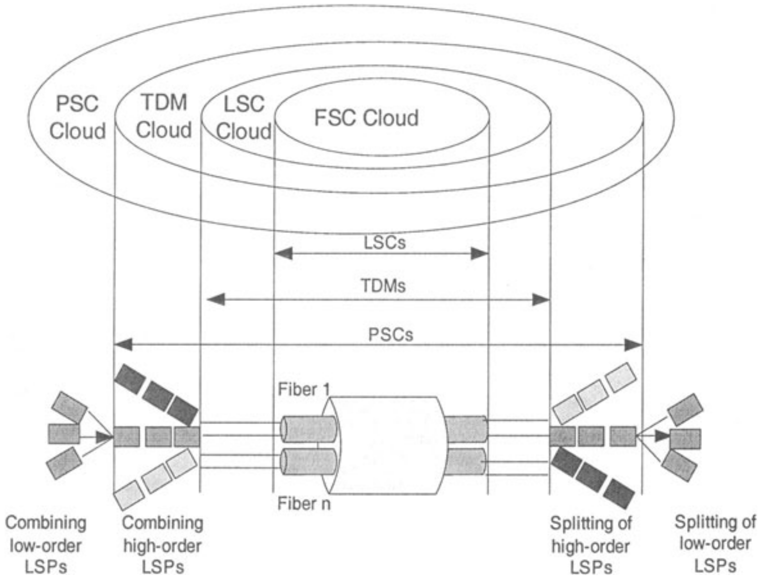


Figure 1. GMPLS Architecture

The strength of GMPLS is to also provide a uniform semantic for network management, operations, and control in hybrid networks. Additionally, GMPLS provides a tool for real-time service provisioning of different types of channels as well as easy and cheap equipment operation and management. Finally, MPLS allows performing traffic engineering on packet-based network, and its generalization GMPLS to perform this on optical networks as well.

The following are the most often-heard interface types to be supported with GMPLS: (1) Packet Switch Capable (PSC) interfaces (e.g. IP, MPLS, and Ethernet). (2) Time-Division Multiplex Capable (TDM) interfaces (e.g. SONET/SDH Cross-Connect). (3) Lambda Switch Capable (LSC) interfaces (e.g. Photonic Cross-Connect (PXC) or Optical Cross-Connect (OXC)). (4) Fiber-Switch Capable (FSC) interfaces.

As a consequence of the various interfaces, GMPLS establishes a link hierarchy (Figure 1). And as MPLS Label Switched Path (LSP) can be nested, optical channel trails have discrete bandwidth granularity in units of individual wavelength capacity.

A generalized label contains enough information to allow the receiving node to program its cross connect (switching hardware), regardless of the type of this cross connect. Since the nodes sending and receiving the new forms of label know what kinds of link they are using, the generalized label does not contain a type field, instead the nodes are expected to know from context what type of label to expect.

3. FAULT MANAGEMENT IN GMPLS

Fault management includes detection, localization, and recovery of/from failures. The detection of the fault by continuous or periodic checking is the very first step in order to take provisions to repair it. Various mechanisms allow for immediate localization of the fault, others need more work to find out where the fault really is, by analyzing or testing. Fault notification includes notifying an entity to perform recovery and possibly raises alarms in the operations center. Or an entity that performs service management is notified about the inability to provide the network part of a service (e.g. alarms).

3.1 Fault Detection

For fault detection, various approaches exist. Those mainly used for plain MPLS, where some of them also work in more general cases. For MPLS networks that can be classified into three groups: the first one using IP capabilities to detect MPLS defects (ICMP extensions, GTTP, LSP Ping), the second one using the MPLS layer only to detect MPLS defects (Y.1711), the third one operating only on a hop-by-hop basis (LMP, data plane encoding). Note that the first two groups work end-to-end of an LSP.

ICMP Extensions [7] use ICMP messages to convey control information to source hosts. Extensions to ICMP allow an LSR to append MPLS stack information to ICMP messages. Generic Tunnel Trace (GTTP, [8]) supports enhanced tunnel-tracing applications that network operators use to trace paths through an IP or MPLS network's forwarding plane. LSP PING [9] verifies the availability of a connection (Label Switched Path). ITU-T Recommendation Y.1711 [10] provides mechanisms for user-plane fault management, by defining OAM packets sent over an LSP on a predefined label. GMPLS introduces a new protocol called the Link Management Protocol (LMP, [11]). It runs between adjacent nodes and is responsible for establishing control channel connectivity as well as failure detection. LMP also verifies connectivity between channels. The detection of optical data-channel failure is measured by detecting Loss Of Light (LOL). LOL propagates downstream along the connections path and therefore all downstream nodes may detect the failure.

3.2 Fault Localization

The fault localization of the control-link is done simultaneously with the fault detection in many cases, since it is applied locally. E.g., the localization of a data-link failure might be achieved by the Link Management Protocol's (LMP) fault localization procedure that sends LMP Channel-Status messages between adjacent nodes over a control channel maintained separately from the data-bearing channels. In other cases, it is pretty difficult to localize the fault, since only end-to-end fault detection mechanisms are involved.

3.3 Fault Notification

Depending on where the faults are detected, there is an upstream notification needed or at least a management system alarm needs to be raised.

RSVP specifies that errors be notified to upstream node using PathErr messages and to downstream nodes using ResvErr messages. Moreover, a Notify message (that contains the affected LSP and failed resource) has been added to RSVP-TE for GMPLS to provide a mechanism for informing non-adjacent nodes of LSP-related failures. These Notify messages do not replace existing RSVP error messages as they differ from them in that they can be targeted to any node other than the immediate upstream or downstream neighbor.

3.4 Fault Recovery

The purpose of fault recovery is to trigger for corrective actions when failures occur. The goal is that this must happen as fast as possible. However, the recovery speed has an impact on the potential service downtime and on the resources allocated within the network. That's where service differentiation makes a lot of sense, because also prices will be differentiated.

Several differentiators are possible including the following:

Recovery is done via the control plane or the management plane, where control plane recovery is faster, but needs configuration to do the expected thing.

Secondary paths are pre-established (protection) or established on demand in case of a failure (restoration).

The failure might be addressed either at the transit node where the failure is detected, or at the endpoints. The transit node handling is faster, since it does not need a signaling action taking place in order to notify the end nodes of a path. However, it might mean that several secondary paths are established from several transit nodes, if protection switching is used.

Different bandwidth sharing schemes are possible. No sharing at all, share several secondary paths for the same primary LSP, share bandwidth for several primary paths.

Secondary, pre-established paths are carrying traffic all the time (1+1 protection) or it will be switched over in case of failure (1:1 protection). The fist choice is faster or even no service interruption at all.

In the following we constrain ourselves to only part of the problem.

4. GMPLS SERVICE MANAGEMENT

GMPLS Service Management is applied to administrative boundaries such as the user-to-network or the network-to-network interface. In general, two ways shown in Figure 2 are envisioned to request a GMPLS-based service. The managed way is over any communication means the service request is received by the service management system (in our case we use a web interface). The second way, mainly envisioned by the IETF allows using signaling messages (RSVP-TE or CR-LSP) for requesting an end-to-end service.

Since, none of the standardization bodies use resilience as a service parameter so far, and since we provide other services such a Gigabit Ethernet over MPLS as well, we favor the managed way, but will work on the signaled way in the future.

Another issue for fault management in GMPLS is that the control plane is applied to different underlying technologies. Therefore not all of the possible protection schemes are possibly implemented. This means the managed approach allows for better targeting the service requested knowing the service able to be provided.

Finally, there is the issue of hierarchy, where GMPLS is applied to several networking technologies within the same domain and potentially IP runs over a GMPLS network. In this case, one needs to decide on what layer what protection scheme is used. Otherwise all protection schemes will be implemented at all layers, which is an overhead not needed. On the other hand, the layer performing fault recovery must be chosen based on protection and restoration capabilities and operators policy.

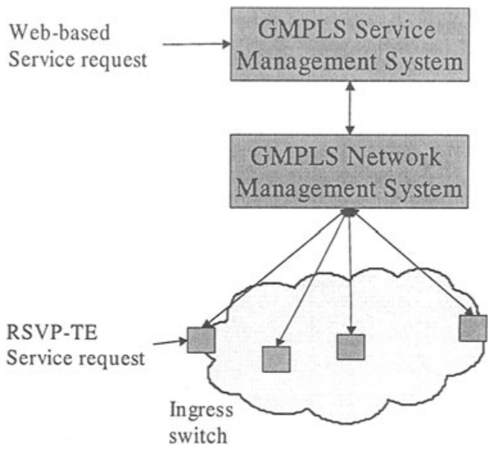

Web-based Service request
GMPLS Service Management System
GMPLS Network Management System
RSVP-TE Service request
Ingress switch

Figure 2. General System Overview

Figure 2 also shows the interaction between the service management and the network management system and the interaction between the network management system and the GMPLS switches. The functionality available on the network management level is in our case topology discovery, basic configuration management, monitoring, and viewing capabilities. For setting up a service the network management system gets the GMPLS LSP (the primary and secondary paths) and configures it into the network using different means. It mainly needs to manage label space or it triggers RSVP-TE to setup the LSP.

4.1 Service Definition

The service definition for bandwidth and resilience might include several parameters depending on what is possible in the network and what a customer requires, and what technology is used. In our service definition the following parameters are used:

Source and Destination Address of the service, where we currently use IP addresses, but GMPLS in general allows for unnumbered links, where an interface is a number on a switch.

Service Type specifies what type of service a customer requests. At the moment we can support packet MPLS (packet over anything), lambda paths (DWDM), Gigabit Ethernet over MPLS (a specific application).

Bandwidth, specifically, for packet MPLS cases we have a high granularity, for other technologies the granularity is given by the technology, e.g. SONET granularity.

Maximum Service Interruption Time specifies the time a customer is willing to accept a service outage. Which means the service is not available at all during that time.

Minimum Bandwidth in failure cases denotes the bandwidth, which needs to be available when a failure occurs. This translates into the bandwidth allocated for secondary paths, where the traffic is switched to in case of a failure, or the bandwidth signaled for in fast rerouting.

4.2 Application to an Optical Label Switch

In the following we are getting more specific towards our implementation. For that reason some explanations of the underlying hardware restrictions and constraints are given. See [12] for a more extensive description.

The main functionality of the Optical Label Switch (OLS) is that it runs GMPLS for optical paths and for MPLS packet paths. We implement the overlay model only, which means that the links seen on the packet layer are optical paths. Both path setups are triggered by the management system and use RSVP-TE with GMPLS extension in order to setup paths in the network.

For fault recovery we have a constraint that on the optical level path setup, update, and switching to a secondary path are very slow. On the other hand, on MPLS level switching to a secondary path is basically instantaneous after the failure has been detected. Failure detection is possible on the sending side of an optical path. This means on the MPLS level each hop of an MPLS LSP can switch to a secondary path.

Naturally, any failure raises an alarm in the GMPLS network management station. In case of the optical path having troubles, the management system can decide on what secondary path the traffic might be mapped. So here we use on-demand, managed secondary path setup scheme only. Due to a limitation of the number of optical interfaces per OLS, it is in many cases not possible to even setup a secondary optical path.

On the packet MPLS level however, we are able to use the fast protection switching capability of the OLS. However, this means we need to pre-compute and pre-setup secondary paths in advance during service provisioning time.

4.3 Protection Schemes Used

In the following, we describe the protection schemes evaluated. However note again that most of them are well known from the literature [13], [6], [5]. As described in the previous chapter the schemes and the evaluation take into account some of the constraints of the OLS specific features. So we assume only pre-established paths, where the establishment is different and the failure notification mechanism is different.

For protection schemes we use end-to-end, link local, and local-to-egress, (see Figure 3 - Figure 5).

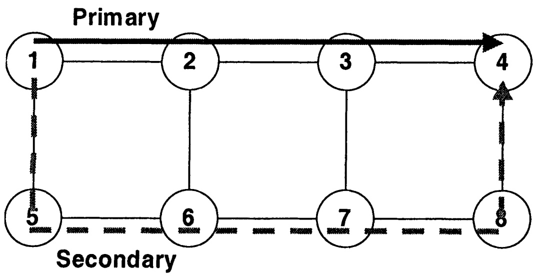


Figure 3. End-to-end Protection Scheme

End-to-end means one secondary path is setup from the ingress switch to the egress switch. The benefit of this scheme is the number of secondary LSPs used for backup. Namely there is only one secondary used per primary and the bandwidth allocated for the secondary is pretty low depending on sharing schemes. However, the worst-case notification time is bigger then in the following schemes. Additionally, a signaling protocol is used to signal failures upstream such that the traffic can be switched to the secondary path.

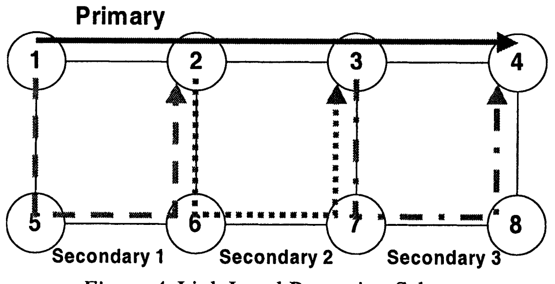


Figure 4. Link Local Protection Scheme

Link local requires the setup of secondary paths from the ingress and from each intermediate switch to the following switch on the primary path (in Figure 4 from 1 to 2, from 2 to 3, from 3 to 4). The benefit lies in the fast reaction time in case of failures. Basically only link failure detection is needed and no upstream signaling is required. However, the large number of secondary is a problem, as well as the bandwidth allocated for the secondary LSPs is a problem, since in most cases they do not share the same path for the same primary LSP.

Local-to-egress means, from each intermediate switch a secondary path is setup to the egress switch (in Figure 5 from 1,2,3 to 4). In this scheme, we have the same fast recovery as in the Link-Local Scheme, but we have better sharing capabilities.

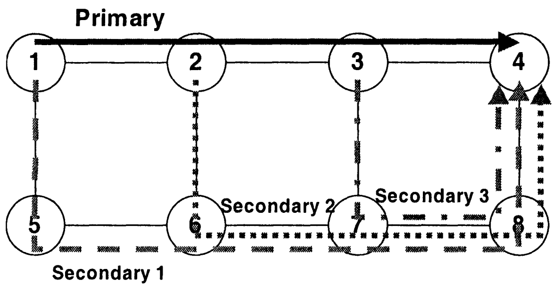


Figure 5. Local-to-Egress Scheme

For certain topologies and a small number of nodes one can run linear programming based optimization and achieve about the same sharing ratio as the end-to-end protection scheme [14]. One of the drawbacks of this scheme is, that it is very difficult to implement it in the control plane. It is difficult to change the signaling protocols to work such that the protection scheme is setup. In our case, this is not a problem, because we mainly rely on managed networks and only setup the LSP with pre-computed secondary paths.

4.4 Sharing Schemes

For bandwidth sharing schemes we use no sharing, per path sharing, max bandwidth sharing, percent sharing per path, and percent sharing global.

No sharing means there is no bandwidth sharing at all. Also in link-local or local-to-egress protection schemes, all the bandwidth is allocated on the secondary paths. Specifically, in local-to-egress, this is a bad solution. In Figure 5 it would mean to allocate three times the bandwidth of the primary LSP on the link from 7 to 8.

Per path sharing is useful for local-to-egress and link local protection schemes, where several secondary paths are used for the same primary path. In Figure 5 it would mean to allocate only once the bandwidth of the primary LSP on the link from 7 to 8.

Max bandwidth sharing allocated only the maximum bandwidth of all secondary paths crossing through that link. This does not give any guarantee on the bandwidth, but the hope is that in cases where only one primary path is affected it still has enough backup capacity. So the backup capacity is a function of the number of failed LSPs using a particular link as backup.

The percent sharing schemes allocate a certain percentage of the primary path to the secondary path. This is mainly used for service specification, where a customer is willing to reduce his bandwidth demand for a certain time in failure cases.

Percent sharing per-LSP is an optimization such that in protection cases, where several secondary paths are allocated and that they share the bandwidth. This is basically a combination of per-LSP and percent sharing.

5. EVALUATION

In the following, we give a numerical evaluation of the different schemes with a set of topologies. Since we do not have enough Optical Label Switches, we need to simulate the schemes, however we have implemented some of the schemes also for the real system. We use a homegrown Java-based simulation tool for that task. Some of the simulation results are qualitatively known in advance, but if it comes to charging for a service, we need to get some quantitative measures of the different schemes and their behavior.

The routing algorithm used is constraint shortest path first (CSPF), which has known problems but is very easy to implement. All the numbers are averaged numbers over 100 simulation runs and averaged over all links in the network. The stopping criterion is defined as 10 consecutive failed service setups. The link bandwidth of all topologies is 2.4 Gbit/s and the degree of the topology is 4. We have chosen the degree of 4, because of hardware limitations on the OLS.

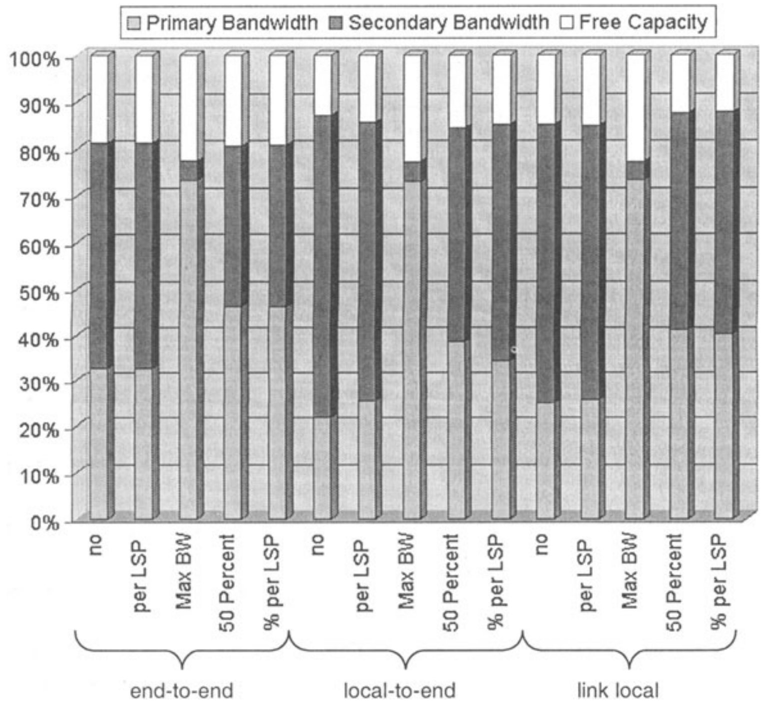


Figure 6. Topology 16 Nodes, Degree 4, Ring-Mesh

5.1 Experiment 1

In the first experiment, service requests are chosen with random originating node, terminating node, and bandwidth in the range of 10-100 Mbit/s. Results of one topology with 16 nodes are shown in Figure 6. The topology is basically a ring with additional links between each second and third node in the ring.

Figure 6 also shows the effect of sharing schemes. For end-to-end protection it does not matter whether we have no or per LSP sharing, because there is only one secondary path anyway. Also in the other protection schemes it does not matter too much. Not surprisingly we see that the Max BW sharing scheme performs best in terms of secondary path bandwidth consumption. However, no guarantees can be given that these paths get the requested bandwidth in failure cases. In case of sharing based on 50% of the primary path the primary to secondary bandwidth ratio is much better compared to other schemes.

In summary, the Max BW sharing scheme does perform by far the best, with the least guarantee. So all service requests with very small bandwidth guarantee in failure cases will use this scheme. Most likely this service type also does not need fast protection switching. So this class of service will use end-to-end with Max BW allocation for secondary paths.

5.2 Experiment 2

As seen before per LSP sharing naturally makes no sense for end-to-end protection schemes. Therefore we excluded it and all Max BW sharing schemes in the following simulations. In this experiment we study the effect of topology on the results of the scheme. We use three different topologies, the random, the ring-mesh, and a mesh, all of them again with degree 4. The difference from the ring-mesh to the mesh is that in ring mesh the links are nearer to the ring, where in the meshed topology links run across the complete network. Figure 7 shows the result of simulating various schemes in various topologies. The values shown are the ratio between total capacities allocated for primary paths divided by the total capacity allocated for secondary paths.

We definitely see that end-to-end protection is performing better compared to local-to-egress or link local. However that scheme needs longer detection and notification time. In case of link failure detection, the head end of an LSP must be notified. Or in case of end-to-end connectivity check the failure detection takes more time. So we use this scheme for service requests with moderate service interruption time allowed, because the capacity cost is less than with the other schemes.

Furthermore, it can be seen that for link local and local-to-egress the per-LSP sharing scheme (with or without percentage) performs well compared to without sharing. Since per LSP sharing does not influence the service performance but is using the capacity more efficiently we choose this one.

The other important observation is that depending on the topology local-to-egress and link-local performs different. Since in more meshed topologies such as mesh 16 and mesh 32 in Figure 7, the local-to-egress scheme performs better, we have chosen that one for our implementation. However, there is more work needed to exactly figure out the impact of topological issue influencing what part of the problem.

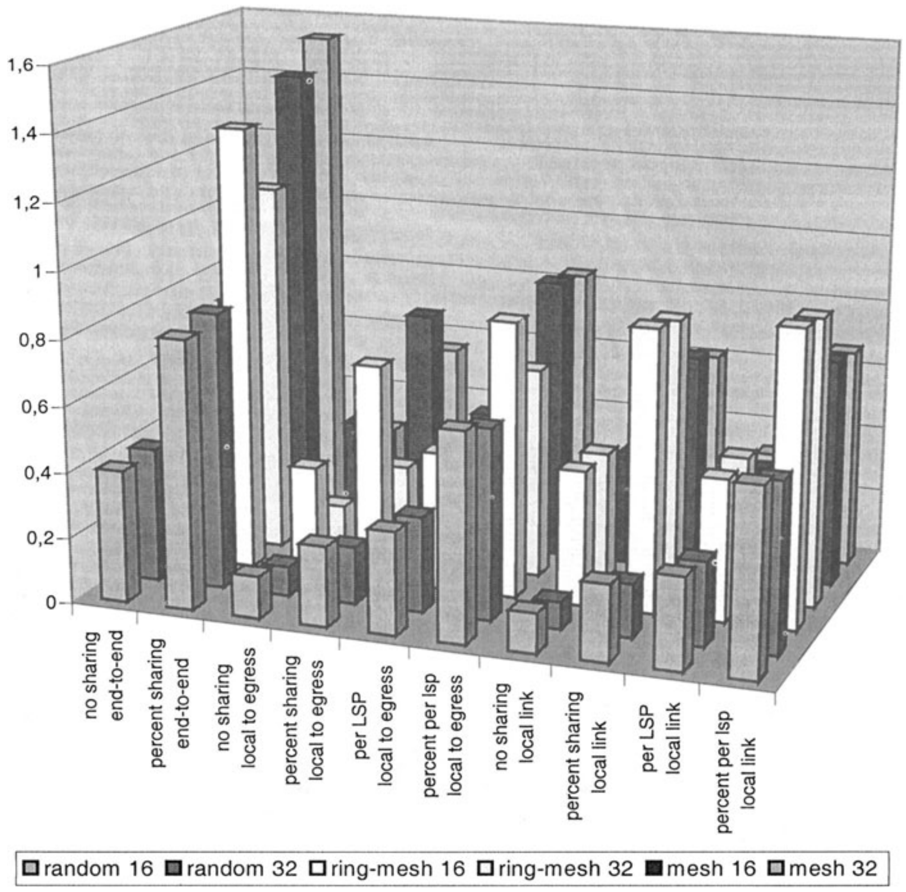


Figure 7. Ratio Primary to Secondary Capacity Used (Y-axis)

Comparing per-LSP and percent per-LSP sharing, we see that percent per-LSP sharing performs better. But naturally secondary paths might not have full capacity. They only get guaranteed X% of the primary capacity. However, taking into account that only a small set of failures occurs, it still might get full capacity, but without any guarantee.

5.3 Experiment 3

So far we have simulated only one scheme per network. In the following we simulate various schemes running at the same time on the network. This is the simulation of different service types running on one network. With the above observations we choose end-to-end and local-to-egress protection scheme, and per-LSP, percent sharing, and percent sharing per-LSP.

Additionally, we have broadened the topology scope and added a 64-node topology of each type, and added two new random generated types of topologies know as the Waxman topology [15] and the Barabasi and Albert topology model [16].

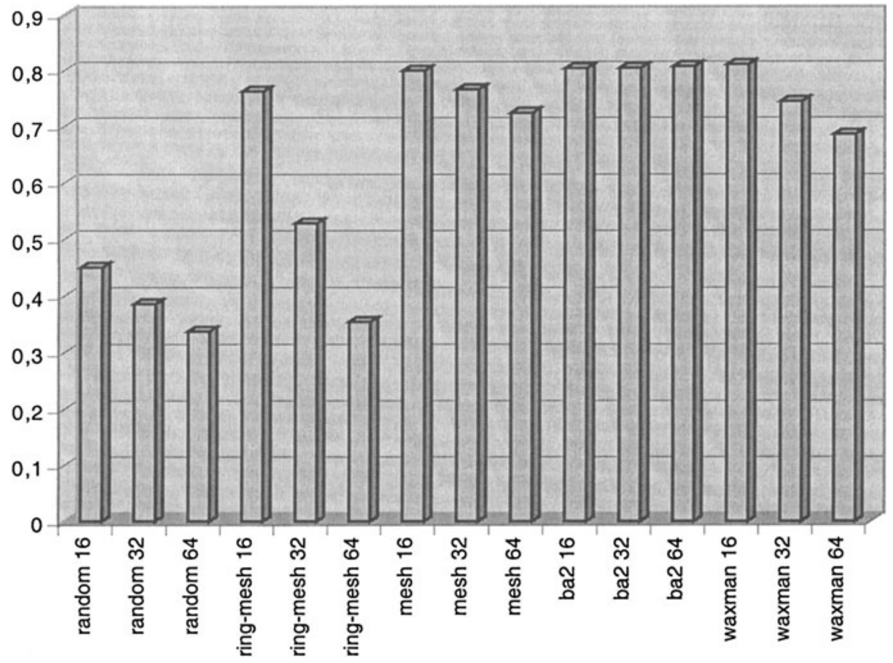


Figure 8. Primary/Secondary BW Ratio (Y-axis) for Services Mixed on the Network

Figure 8 shows results, which basically are similar for all the topologies. The bad numbers for the random topologies again stem from many unavailable secondary paths (no two distinct to reach the nodes). The remarkable result here is in the numbers for the ring-mesh 32 and 64 topology. Here we get worse result. This stems from the average hop count of primary paths allocated in the network. The average hop count for the "ring-mesh 32" is 4.31, whereas it is 2.33 for "ring-mesh 16", 2.04 for "mesh 16", and 3.16 for "mesh 32". And it is even worse for ring-mesh 64, where the average hop count is 8.25. Note that the bigger the average hop count the more secondary paths are allocated for local-to-egress protection scheme, which impacts the overall allocation of mixed protection schemes.

6. CONCLUSION

The idea of differentiating GMPLS-based services proposed in this paper is a reasonable way of efficiently use the bandwidth of a network and still provide the customers the service they want. We mainly differentiate the service based on resilience parameters. We implemented the most appropriate schemes as an add-on for a GMPLS network management system built for an Optical Label Switch (OLS). However, we used simulation as a method to numerically evaluate the schemes first to figure out the benefits and drawbacks.

So far we have not considered the hierarchical nature of GMPLS in the simulations. For the less general GMPLS service management system implementation, we included also the hierarchical issue. Basically, we kept the two hierarchies pretty independent, but changed the algorithm for the secondary path calculation such that not only the underlying 'virtual' topology is used, but also the physical topology beneath is taken into account for secondary path calculations.

The simulation regarding the mixture of different services on the same network needs much more attention and work to detect the influencing parameters. Additionally, we have not considered a dynamic system, where customers come and go. One of the problems is the estimation of the call arrival process and the service type distribution.

REFERENCES

[1] Mannie et al., "GMPLS Architecture", IETF Internet Draft, work in progress, draft-ietf-ccamp-gmpls-architecture-03.txt, August 2002.

[2] Davie and Rekhter, "MPLS Technology and Applications", Morgan Kaufmann Publishers, 2000.

[3] Banerjee et al., "GMPLS: An Overview of Routing and Management Enhancements", IEEE Communications Magazine, January 2001.

[4] G. Li et al., "Control Plane Design for Reliable Optical Networks", IEEE Communications Magazine, Feb 2002.

[5] Sharma and Hellstrand (Editors), "Framework for MPLS-based Recovery", IETF Draft, draft-ietf-mpls-recovery-frmwrk-08.txt, work in progress, draft-ietf-mpls-recovery-frmwrk-08.txt, October 2002.

[6] Banerjee et al., "GMPLS: An Overview of Signaling Enhancements and Recovery Techniques", IEEE Communications Magazine, July 2001.

[7] Bonica et al., "ICMP Extensions for Multi Protocol Label Switching", Internet Draft, draft-bonica-icmp-mpls-02.txt, work in progress, Nov. 2000.

[8] Bonica et al., "Generic Tunnel Tracing Protocol (GTTP) Specification", Internet Draft, draft-bonica-tunproto-01.txt, work in progress, July 2001.

[9] Pan et al., "Detecting Data Plane Liveliness in RSVP-TE", Internet Draft, draft-pan-lsp-ping-02.txt, work in progress, 2002.

[10] ITU-T Draft Recommendation Y.1711, "OAM mechanism for MPLS networks", work in progress, 2002.

[11] Lang (Editor), "Link Management Protocol (LMP)", Internet Draft, draft-ietf-ccamp-lmp-07.txt, November 2002.

[12] M. Arai et al., "High Performance Network with Merged Optical and IP", IEEE High-Performance Switching and Routing Workshop (HPSR'02), Tokyo, May 2002.

[13] A. Autenrieth, A. Kirstaedter, "Engineering End-to-End IP Resilience Using Resilience-Differentiated QoS", IEEE Communications Magazine, Vol 40(1), January 2002.

[14] H. Saito, M. Yoshida, "An optimal recovery LSP assignment scheme for MPLS fast reroute", 10th International Telecommunication Network Strategy and Planning Symposium (Networks 2002), Munich, Germany, June 2002.

[15] B. Waxman, "Routing of Multipoint Connections", IEEE Journal of Selected Areas of Communications (JSAC), December 1988.

[16] A.L. Barabasi and R. Albert, "Emergence of Scaling in Random Networks", Science, 286:509–512, October 1999.

FUNCTIONAL EVALUATION OF AN INTEGRATED IP OVER WDM MANAGEMENT SOLUTION

Serrat,J.[1], E.Grampín[1], L.Raptis[2], F.Karayannis[3], K.Vaxevanakis[4], D.Chronis[5], H.Katopodis[5], G.Hoekstra[6], W.Romijn[6], A.Galis[7] and E.Kozlovski[7]
(1) Universitat Politècnica de Catalunya; (2) National Technical University of Athens; (3) OTE-Consulting; (4) Ellemedia Technologies; (5) OTE; (6) Lucent Technologies; (7) University College London

Abstract: The management of heterogeneous and hybrid networks has been always a challenge for network operators. Different frameworks and architectural approaches have been proposed and investigated in the literature. The purpose of this paper is to present the evaluation results of an integrated network management solution for the provisioning and maintenance of IP over WDM end-to-end services with network parameters derived from Service Level Agreements (SLAs). A detailed description of the test-bed environment as well as an integrated scenario for architectural evaluation is also included.

Key words: Management of IP over Optical Networks

1. INTRODUCTION

Procedures in the field of network provisioning have become increasingly complex. The manual configuration of connectivity pipes on the network is costly, time consuming, and error prone. As the demand for timely delivery of more innovative services increases, the requirement to automate the provisioning process is stronger than ever. Automating the provisioning process allows service providers to scale their operations and improve quality of end-to-end deployment for new customers.

In the area of network technology, current trends are pointing IP over an Optical Transport Network based on WDM as the approach that will prevail in the future. These two layers need to be integrated with the goal of providing enough bandwidth for quality-differentiated services. Here again an automated provisioning process bound with a personalised service management mechanism is required.

Different approaches have been proposed for the smooth, fast and reliable provisioning and management of Internet services over the optical layer. Most of the research in the area has been focused in the control plane by extending the distributed Internet network control approach to the optical layer using signalling mechanisms either in an overlay model or a peer model. Such efforts, driven by different standardisation bodies, are among others the ASON/ASTN [1] and the Generalised MPLS frameworks [2].

Complementing the above mentioned trends, in July 2000 the IST project WINMAN [3] was launched to look for an integrated network management solution capable of providing end-to-end Integrated Connectivity Services (ICSs) with parameters derived from SLAs. The WINMAN solution consists of the extension of the telecom-style network management approach to the IP layer through the use of MPLS, which could be treated as a connection-oriented technology. The Internet services would be offered mostly by the management plane, but some features of the control plane, such as restoration and protection mechanisms or routed connection set-up could be exploited.

The WINMAN network management solution and system architecture for managing hybrid IP over WDM networks have being developed and presented so far [4],[5] . In addition the initial results of the evaluation of the WINMAN solution have been available quite recently.

This paper is precisely the presentation and evaluation of these results along with the test-bed that was necessary to show up the system behaviour. Hence, after the summary presentation of the WINMAN management approach of Section 2, we devote Section 3 to describe the test-bed that was installed with the purpose of validating the functional characteristics of the WINMAN system. This section covers details of the IP and WDM infrastructure as well as the adaptations that were required to use proprietary management systems, at NE level, with the WINMAN network management system prototypes. Section 4 presents the integrated testing scenario that was conducted with emphasis on the applications that were executed and the detailed steps that were conceived. Section 5 presents the results obtained along with comments on the fulfilment of the specified functional behaviour. Finally we end with remarks on what we have learnt from these results.

2. THE WINMAN MANAGEMENT APPROACH

The WINMAN scope is defined through a model following the TMN layers. As shown in Figure 1, WINMAN focuses in the Network Management Layer (NML) of the TMN pyramid. The NML is further subdivided in two sub-layers, one being the integrated or inter-technology network management layer, and the other one being the technology dependent layer. This approach is currently being adopted by the ITU SG4 and in particular Q7/4.

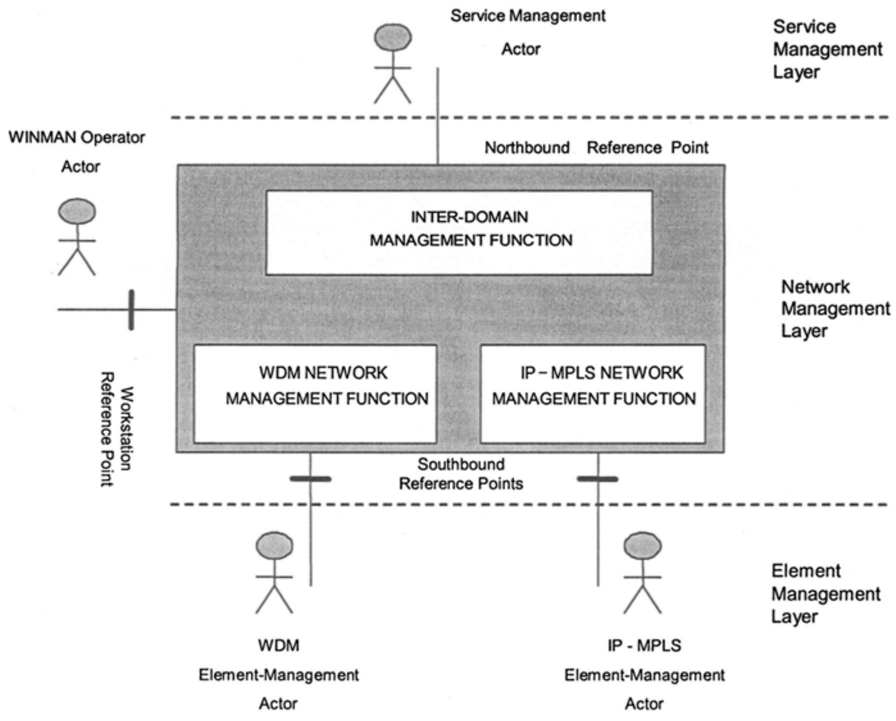

Service Management Actor
Service Management Layer
WINMAN Operator Actor
Northbound Reference Point
INTER-DOMAIN MANAGEMENT FUNCTION
Network Management Layer
Workstation Reference Point
WDM NETWORK MANAGEMENT FUNCTION
IP – MPLS NETWORK MANAGEMENT FUNCTION
Southbound Reference Points
Element Management Layer
WDM Element-Management Actor
IP - MPLS Element-Management Actor

Figure 1. WINMAN High Level Functional Architecture

The reference points for interactions with the outer world are the *northbound reference point* towards other Service Management Systems, such as VPN or VoIP and the *southbound reference points* to the Element Management Layer (EML). Also a Workstation function in TMN terms enables the WINMAN operator controlling and monitoring the WINMAN functionality through the corresponding *workstation reference point.* The WINMAN solution provides inter-technology (inter-domain) functionality as well as network layer functionality for WDM and IP functional systems. For testing or demonstration purposes a lightweight Service Management Layer (SML) has been also developed.

As it is well known the TMF has launched a significant activity to capture the needs of network operators and service providers and thus to enable the "end-to-end process automation of telecommunications and data services operations processes". The Telecom Operations Map (TOM) is the framework for accomplishing the above mission [6]. The TOM defines the business processes and their interactions used by Service Providers in the Customer, Service and Network Management areas. This methodology of business processes decomposition has been also adopted by WINMAN making the appropriate adaptation and customisation.

Specifically, WINMAN covers the following processes of the Network and Systems Management layers:

- Network Provisioning
- Network Inventory Management
- Network Maintenance and Restoration
- Network Data Management

The requirements capture of the WINMAN system was also based in the TOM Business Process decomposition. Specifically, the functional requirements considered in the area of network provisioning are:

- The Provisioning of end-to-end IP paths over light-paths using MPLS technology (the Integrated Connectivity Service or ICS). In this context the WINMAN system is capable of calculating, designing and creating MPLS Label Switched Paths (LSPs) over the corresponding light-paths in the optical domain.
- Support traffic and QoS parameters (network level parameters) for MPLS LSPs derived from SLAs. Policies are also applied in the path-provisioning process.

Secondary functions supporting the above ones are: the discovery of network resources; the maintenance of an inventory of all the network resources with their status and their hierarchical relationship; the notification to the SML about service status and network parameters identified in the SLAs and the updating of the Fault and Performance Management units with the network configuration changes. Network Data Management and Network Maintenance and Restoration although being part of the WINMAN solution are not considered in this paper, because at the time of the writing no results from a test-bed environment were yet available. More detailed information is provided in [7].

From the physical point of view, the WINMAN solution was implemented as three interconnected network management systems (NMSs), namely one is for managing the IP, another for the WDM and the third one is meant for the integrated view of the end-to-end connectivity. These NMSs were specialised from a generic NMS solution, thus giving the design the flexibility to incorporate other technologies. A more detailed description of the physical architecture is contained in [5].

3. TEST-BED SET-UP FOR THE EVALUATION OF THE WINMAN SOLUTION

The test-bed includes the WDM infrastructure, the IP infrastructure as well as the corresponding element management systems (EMSs). The EMSs are out of the WINMAN solution and therefore they must be considered in the test-bed as complementary modules along with the data transport equipment. In this section we present the main aspects of this test-bed.

3.1 Description of the WDM test-bed infrastructure

As shown in Figure 2 the WDM infrastructure consists of a hub and 3 remote nodes, connected via a 2-fibre counter-rotating ring. In fact the hub can be considered as a remote node as well. On the outer fibre ring dedicated (fixed) wavelengths are used for communication between the hub and the remote nodes. On the inner fibre ring, each node is provided with a programmable optical add and drop multiplexer (OADM) to set up wavelength paths with other remote nodes or with the hub. At every node aggregated IP traffic from the LAN is routed to the appropriate wavelength. The remote nodes are located at business sites where IP over 10/100 Base-T is the major service.

The distance between two remote nodes is 8 km on average, with a total ring circumference of 32 km. An optical supervisory channel at 1310 nm is used to transport the management information.

The WDM test-bed contains the following management systems (not represented in Figure 2):

- A NE manager per optical node. This software presents a management view of the OADMs to the agents that control the optical components by means of an RS-232 link.
- An EMS, which will control the four agents. The information model and the northbound interface of this system are aligned with the TMF-MTNM standards [8].

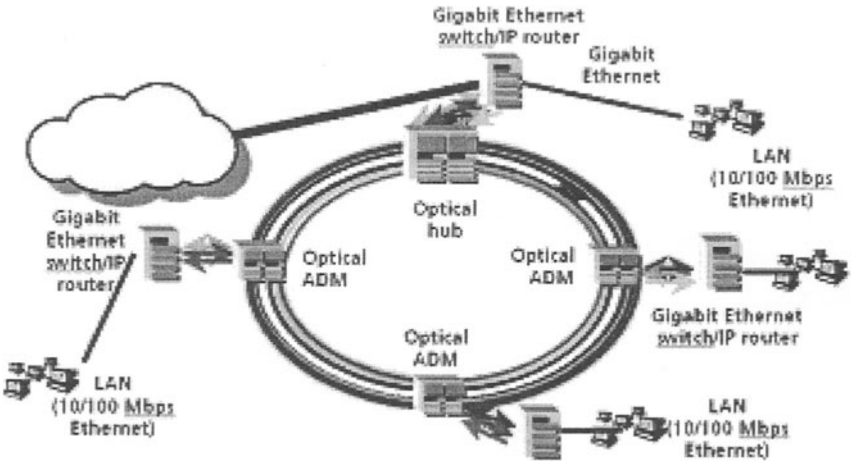

Gigabit Ethernet switch/IP router
Gigabit Ethernet
LAN (10/100 Mbps Ethernet)
Optical hub
Gigabit Ethernet switch/IP router
Optical ADM
Optical ADM
Gigabit Ethernet switch/IP router
Optical ADM
LAN (10/100 Mbps Ethernet)
LAN (10/100 Mbps Ethernet)

Figure 2. The WDM validation test-bed

3.2 Description of the IP test-bed infrastructure

The IP part of the test-bed is depicted in Figure 3 and consisted of the following devices:

- 3 CISCO 7200 routers, namely R1, R2 and R3 comprising the Provider Network. The R1 and R2 are considered Provider Edge routers, connecting to the customers, while R3 is considered a Network Provider router. Each of the 3 routers has 3 Fast Ethernet interfaces. Two such interfaces at each router are needed to establish the triangle transport network of Figure 3. The third Fast Ethernet port is used to connect traffic analyser equipment with Fast Ethernet interfaces, particularly in router R1 and R2. Each router also has serial links used to connect to the customer LANs as well as to the DCN.
- 2 CISCO 1700 routers, serving as Customer Edge routers (CE), namely CE_1 and CE_2. In the customer LANs suitable workstations are connected in order to generate real time traffic such as video. In the particular tests mentioned hereafter, a video conferencing application [9] was used involving two users exchanging video streams between each other.
- 1CISCO 1700 router serving also as an auxiliary site support represented as CE_3 in the figure.

The IP 7200 routers were connected to the WDM equipment through Avaya Cajun switches. The Avaya Cajun switches provide Gigabit Ethernet (GbE) interfaces with Gigabit Interface Converters (GBICs), operating at specific wavelengths. This way, the signal provided by the Avaya Cajun switch is adapted to the wavelength needed by the OADM.

In the opposite direction, the wavelength that is dropped at the switch is directly passed to the client receiving interface. Therefore, the transport stack IP/GbE/WDM at 1.25 Gbit/s is used. The GbE-switches in the test-bed are equipped with WDM lasers, which allow them to interface directly with the WDM layer.

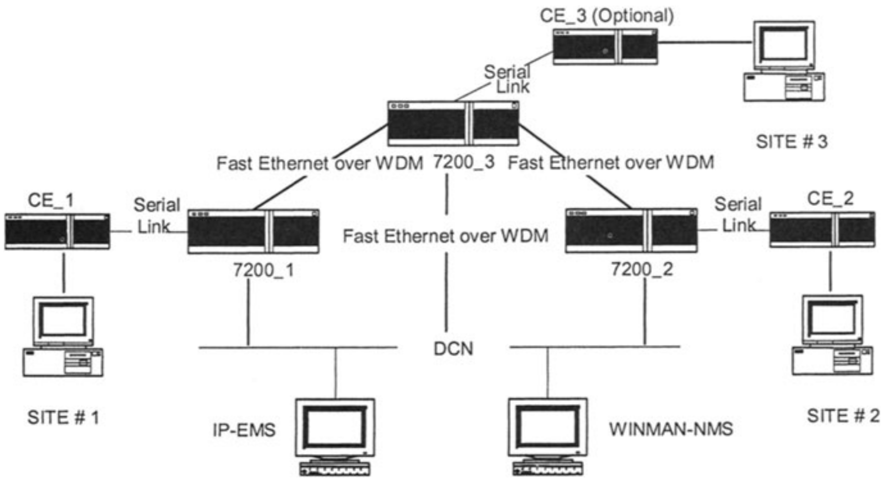


Figure 3. The IP validation test-bed

3.3 Adaptation of the technology dependent EMSs to the WINMAN southbound interface

Once the requirements are met and the test-bed has been set-up, the network management systems under test have to be connected to it through a DCN. This set-up is based on existing NEs, which may have proprietary management interfaces towards the DCN. Consequently, EMSs responsible for those elements are equipped with their native Application Programming Interfaces (APIs) implemented in a proprietary way. Those APIs are incompatible with each other and with the southbound interface of the WINMAN IP and WDM NMSs. Therefore, an adaptation layer between native test-bed APIs and the WINMAN southbound interface is needed. Although the context of this section is tailored to the WINMAN solution, the methodology can be easily applied to any other NMS solution. This methodology consists of the specification and design of an adaptation component on top of each EMS (IP and WDM). These adaptation components appear as a top layer of the test-bed infrastructure and they are the only test-bed entities visible from the WINMAN perspective.

Further down there are different possibilities for the adaptation of the EMS for the IP or WDM technologies. The objective is to adapt through the corresponding EMS either for the WDM network or for the IP network or directly to the management interface of the NEs (e.g. by means of SNMP).

3.3.1 Adaptation of the WDM EMS

Adaptation of the WDM-EMS is based on the TMF-MTNM [8] interface for the WDM-EMS to NMS interconnection.

The corresponding Information Model (IM) describes the configuration of the subnetwork managed by the EMS. This model gives the objects that can be present in the subnetwork, and their relationships. Instances of these objects are contained in the database of the EMS. This database has a static and a dynamic part. In the static part, the physical configuration like the NEs, their physical ports and the links between those ports is represented. The dynamic configuration contains the connections that are sustained by the NEs, together with their connection termination points. If a new optical NE is added, the static configuration changes; if a new sub network connection is created, the dynamic configuration is altered.

3.3.2 Adaptation of the IP EMS

The adaptation between the WINMAN Management System and the EMSs is also based on the MTNM specification. The IP-EMS is conceived more as a mediation device than a stand-alone EMS. The IP NMS to EMS adaptation ensures the establishment of IP Connectivity Services through configuration of LSPs with bandwidth constraints. Hence, the IP-EMS performs this basic functionality, provided that the basic IP and MPLS configuration is already set-up. The interface between the IP-EMS and the routers is through CLI commands with additional SNMP support for monitoring purposes. In fact, The IP-MPLS configuration of routing devices can be accomplished by different means; basic SNMP "Set" commands and CLI commands. As the IP equipment in the test-bed is composed by Cisco routers that currently do not support SNMP "Set" commands for MPLS, the configuration was done by CLI commands.

The initial idea was to have the IP devices pre-configured with basic connectivity (Layer-3) and the MPLS enabled. WINMAN performs the configuration of LSPs with bandwidth assurance, in response to ICS establishment requests. The routing computation for such LSPs will be done by the IP-NMS, while the set-up will be done using the control plane through the RSVP-TE signalling protocol; the Control Plane can also provide dynamic backup for these LSPs in case of failure.

4. INTEGRATED SCENARIO FOR VALIDATION OF THE WINMAN FUNCTIONALITY

The WINMAN functionality has been validated using an integrated network provisioning scenario. The main purpose of this scenario is to establish an ICS between routers 7200_2 and 7200_1 using the 7200_3 intermediate node appearing in Figure 3 and shown again here in Figure 4, which is a complete picture representing all the layers involved in the testing scenario.

The IP link between R2 and R3 is configured but not operational because there is no physical (WDM) connectivity. In order to provide an ICS between the R1 and the R2 through R3, the management system has to establish first the physical link between R2 and R3 and then create a bi-directional LSP (7200_2-7200_3-7200_1 and 7200_1-7200_3-7200_2). The business case behind this scenario is that the WINMAN system is capable to find the resources and provide an ICS with QoS constraints, in a case where the default shortest path cannot accommodate the request due to shortage of resources (e.g. lack of bandwidth).

This scenario involves multiple provisioning requests and all the scenario steps along with the start-up conditions are listed below.

4.1 Initial set-up

There are two customers, namely IXIA and DEMO as shown in Figure 4:

- IXIA: Consists of two traffic generators connected to R1 and R2.
- DEMO: Two multimedia clients connected also to R1 and R2.

There is pure Layer-3 (IP) connectivity between routers R1-R3 and between routers R1-R2. The IP link between R2-R3 is configured but not operational because there is not physical (WDM) connectivity, so communication between R2 and R3 is only feasible through R1.

The 10/100 Ethernet ports between the core routers are configured at 10 Mbps, so the link R1-R2 can be easily saturated. This kind of configuration also limits the bandwidth available in each WDM λ to 10Mbps. But this does not reduce the functionality of the network; it will only decrease its bandwidth to values that are more suitable for testing purposes.

The WINMAN system has the following policies activated:

- A positive authorisation policy at IP-MPLS level allows the establishment of links with guaranteed bandwidth between R1 and R2.
- A negative authorisation policy at IP-MPLS level is not enabling the routing of traffic between routers R3 and R1.

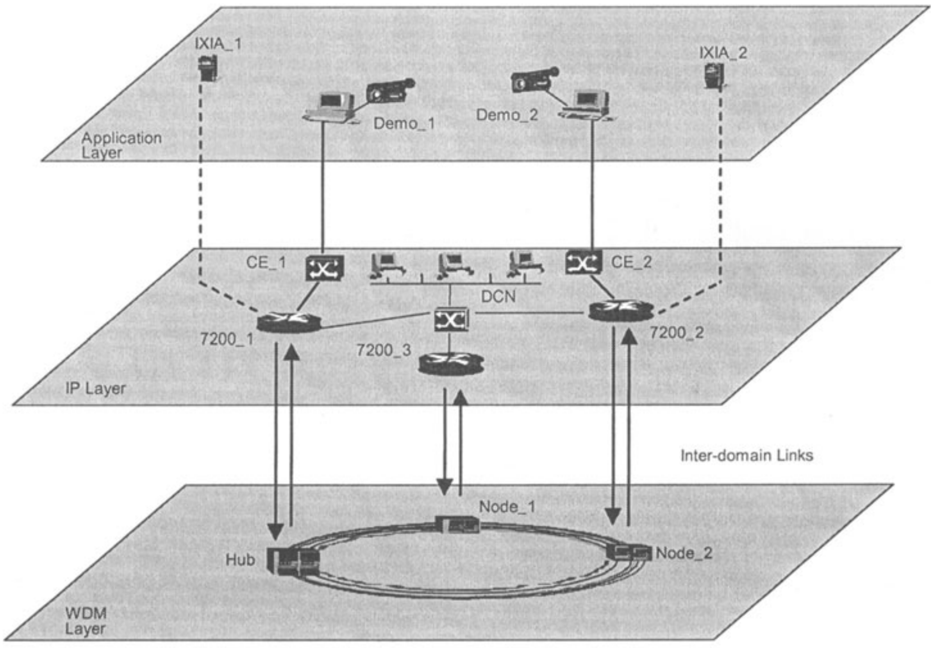


Figure 4. Integrated functional validation scenario

4.2 Scenario Steps

4.2.1 Creation of an ICS for the IXIA customer

The IXIA generators belong to a WINMAN customer that needs to exchange information between its two sites. Therefore, one of the clients requests a WINMAN service between R1 and R2 asking for 9.5 Mbps always available. The WINMAN system creates the ICS making the necessary bandwidth reservation and establishing two unidirectional LSPs between the involved routers. The two generators start working and they really use all the bandwidth requested to the WINMAN system.

Under these network conditions, the DEMO customer requests a connection between the same locations specifying 1.5 Mbps bandwidth. Clearly, the request can't be satisfied through the same path.

4.2.2 Creation of an optical path

The WINMAN system will be aware of the lack of bandwidth in the shortest path between R1 and R2 and therefore will look at the WDM network and conclude that there is a possibility to create an additional optical trail connecting R2 to R3. The request for the creation of the optical trail is forwarded by the WINMAN system to the WDM-EMS.

4.2.3 Synchronisation of the network inventory

The new optical trail is created and a new IP link is discovered by the IP-EMS between R2 and R3. The WINMAN network inventory is updated with the new WDM and IP links, and the GUI is displaying the actual network configuration.

The bi-directional SNC creation R2-R3-R1 and R1-R3-R2 is not yet possible because the routing policy *Disable IP link between R3 and R1 for routing* is applied. Then the WO manually changes the disable policy for a positive authorisation one.

4.2.4 Creation of the ICS for the DEMO customer

Once the new optical path has been created, the new IP link is detected and the applicable policies allow it, WINMAN will create a new ICS through R1 <-> R3 <-> R2. The shortest path was excluded because it didn't have enough bandwidth. The traffic generated by the DEMO multimedia clients is now forwarded through the new ICS and the video service reaches its SLA.

5. RESULTS EVALUATION

The integrated scenario defined in the previous section was completed successfully.

The GUI always presents the most up-to-date status of the managed network to the WINMAN operator. The policy user interface is launched directly from the main GUI window for additional convenience. There was no situation of having misalignments between network status and network map. This proves the good functioning of the southbound interface and the network inventory manager functionality. End-to-end routing triggered by the provisioning functionality exhibits sophisticated router-based methods speeding discovery of available or potential routes.

Nevertheless, some of the nodes and termination points are not easily distinguishable, and in case of many depicted elements the screen should be adjusted to higher resolutions to have a better view of the underlying network. All nodes (termination points, devices) in the views can be dragged on the screen so that they can be allocated in a manner that is "readable" to the operator.

The views are not automatically refreshed when a change takes place in the system. It is only updated with the changes when it interrogates the system using a *refresh function*. That is why there is a refresh button by means of which the GUI retrieves all the newly available information.

The experiments referenced were performed using the WINMAN GUI and not sending requests by an SMS system. To test that WINMAN is open to other high level management systems we sent some SMS commands using Tcl/tk scripts emulating the whole process. The outcome was as expected.

The waiting time to see a connection displayed in the GUI since the moment of issuing the connectivity request is quite long. However this is not considered a drawback because the project was more oriented to a proof of concept than to a precompetitive product. In fact, this was assumed since the early beginning when, for instance, the use of two platforms with a mediation gateway in between was adopted; no special adoption of protocols or operative systems was considered in regards to efficiency, etc. For this reason we haven't provided numerical data of performance measures in the different steps involved in the management processes because they wouldn't be representative of an operative system. Nevertheless to give an example we have estimated that with a single platform type, making use of high performance hardware and with an optimised software implementation, the total process described in the scenario would take around 1 minute.

The experiments validated that WINMAN exhibits and leverages network relationships. The WINMAN system succeeded in carrying out the provisioning of IP connectivity services with guaranteed QoS (in terms of guaranteed end-to-end bandwidth provisioning) in an automatic way by making the appropriate changes to the IP and WDM networks. Having knowledge of both network layers, network resources are exploited in an easy and consistent way under the supervision of policies. System installation is rather complex but as we said before WINMAN is a just an experimentation prototype.

Traditionally, each layer of hybrid transport networks is independently managed having its own requirements, problems and unique operational characteristics. WINMAN is the first system that can deal with the integrated management of IP and WDM technologies providing increased flexibility, services and utilisation of resources.

The results on the adaptation of the network infrastructure to the WINMAN southbound interface, executed for the IP network and the WDM network, were also successful both in terms of functionality and also satisfactory in respect to the system overhead. The diversity of interfaces between EMSs and their managed NEs and also the interfaces offered to the NMSs is really high. Nevertheless from the WINMAN perspective all that is required is an adaptation to a TMF interface. This adaptation will adopt the shape of a Q-adaptor, IDL mediation or any other. Of course this is valid under the assumption that the northbound interfaces presented by the NE managers are open.

6. CONCLUSIONS

Experimental test scenarios were designed and executed for the functional evaluation and demonstration of the WINMAN integrated management solution. These experiments were done in an integrated scenario in the context of the first release of the WINMAN solution (that covers Configuration Management only). Clearly, providing an integrated IP and WDM Configuration Management is only half way towards an integrated solution for managing IP and WDM networks. Fault and Performance management and especially alarm monitoring, network restoration, in conjunction with SLA and QoS monitoring are challenging and are currently being addressed. But the execution of experiments in the currently available version has proven that WINMAN provides a feasible solution with increased flexibility and utilisation of resources. As the same design concepts and tools are used for the extended functionality system we believe that the results will be also satisfactory. Therefore we conclude that WINMAN is an integrated and automated provisioning solution of IP-based network services over an Optical Transport Network that empowers service providers to quickly bring new differentiated services to market. In addition, WINMAN is paving the way for flow-through automation in conjunction with other operations support system (OSS) applications, by means of standardised interfaces.

The main outcome of the WINMAN evaluation is that not only it provides the design guidelines for an integrated management system based in a 3-tier concept that makes it extensible to manage any connection-oriented technology, but also and the most important perhaps is that it provides the means for a smooth evolution path towards the full integration of IP/MPLS and WDM. This is important because the peer integration based on the control plane will be for sure delayed especially due to the telecom crisis-slowdown; so intermediate solutions having the management plane as a basis and possibly supported by the control plane features (especially of the mature IP layer) should be promoted. This paper shows that these solutions are viable.

The WINMAN solution proves that the design, implementation and integration of such a complex system is feasible and can actually work if: a) state of the art software technology like component-based frameworks and distributed architectures like CORBA are used; b) standardised interfaces between the main sub-systems (IP-NMS, WDM-NMS, INMS) are adopted and the appropriate extensions are proposed in order to cover the managed networks (i.e. MTNM extensions to cover IP/MPLS); c) a systematic and well-defined methodology (RUP-like) is applied for the whole life-cycle of the system (design, implementation, integration).

7. ACKNOWLEDGMENTS

This paper describes part of the work undertaken and in progress in the context of the WINMAN-IST 13305, a two and a half years research and development project during 2000-2003. The IST program is partially funded by the Commission of the European Union.

8. REFERENCES

[1] Aboul-Magd, O. et. al., "Automatic Switched Optical Network (ASON) Architecture and Its Related Protocols", draft-ietf-ipo-ason-00.txt, work in progress, July 2001.

[2] Ashwood-Smith, P. et. al., "Generalized MPLS- Signalling Functional Description", draft-ietf-mpls-generalized-signaling-05.txt, work in progress, December 2002.

[3] Project IST-1999-13305 WINMAN http://www.telecom.ntua.gr/winman/.

[4] Serrat, J. et al., "Integrated Management for IP end-to-end Transport Services over WDM Networks" IFIP/IEEE International Symposium on Integrated Network Management IM 2001, 14-18 May 2001, Seattle, USA.

[5] Karayannis, F. et al. "Management vs. Control Plane approaches for integration of IP and WDM layers– A synergy paradigm", 8th IFIP/IEEE Network Operations and Management Symposium NOMS 2002, 15-19 April 2002, Florence, Italy.

[6] Enhanced Telecom Operations Map (eTOM): The Business Process Framework for the Information and Communication Services Industry – GB921 v3.0.

[7] Raptis, L. et al, "Integrated Management of IP over Optical Transport Networks", IEEE International Conference on Telecommunications ICT 2001, 4-7 June 2001, Bucharest, Romania.

[8] TMF 608 Multi-Technology Network Management Information Agreement NML-EML Interface. Version 2.0. October 2001.

[9] Meeuwissen, H.B., H. J. Batteram, and J.L. Bakker, "The FRIENDS Platform- A Software Platform for Advanced Services and Applications", Bell Labs Tech. J., Vol.5, No.3, Jul.-Sep. 2000, pp. 59 -75.

A NETWORK-ORIENTED POWER MANAGEMENT ARCHITECTURE

LUIS F. POLLO and INGRID JANSCH-PÔRTO
Universidade Federal do Rio Grande do Sul – Instituto de Informática
P.O. Box 15064, 91501-970 – Porto Alegre, RS, Brazil
{pollo, ingrid}@inf.ufrgs.br

Abstract: This paper presents the proposal and implementation of a power management architecture for local area network environments. Our main goal is to contribute to more efficient use of electricity, reducing energy waste by facilitating the configuration of power policies and providing a means to automate their execution.

The architecture we propose is based on the SNMP (Simple Network Management Protocol) framework. The management process is coordinated by a central element, which applies configurable power policies to computers and other electronic devices connected to the network, either in accordance to consumption preferences defined by the network administrator, or in response to changes in the supply of electricity, detected through monitoring of UPS (Uninterruptible Power Supply) devices. In the latter case, applying the pre-configured power policies allows the UPS to sustain power to essential parts of the network (e.g. servers) for a longer period of time. Alternatively, the definition of power policies can be based exclusively on administrative preferences, in which case the goal is to minimize consumption of electricity during non-business hours by shutting down inactive equipment or putting it in low-power modes.

We have measured UPS autonomy during utility power outages and found that it can increase by a factor of 7 in a small network setup, by using the management system to automatically power off 90% of the computers. Potential economy resulting from the decrease in consumption during non-business hours alone is estimated to be as high as US$ 36 per computer during the period of a year.

Key words: power management, energy efficiency, network management, SNMP, ACPI, APM.

1. INTRODUCTION

Electric power consumption is increasingly becoming an essential aspect of computing. The wide-spread use of mobile computing devices, such as personal digital assistants (PDAs) and notebooks, has boosted research in the area of power management, where several contributions have been made, most of which focused on reducing consumption in order to allow longer battery-powered operation. However, the impact of energy consumption extends far beyond this category of devices. In the United States, recent studies have shown that the electricity used by office and network equipment corresponds to a considerable percentage of the total amount consumed, with a high estimate of energy waste due to poor use of existing power management mechanisms [1]. Those findings only reinforce the importance of power management for all sorts of computing equipment, from the smallest pocket device to the largest corporate server.

Another recent trend in the computer industry that is likely to have a very significant impact on energy consumption is that of the "appliance-PC" – a personal computer that is always ready to use at the push of a button, the same way a TV or another home appliance would be. Industry giants such as Intel and Microsoft have been working on hardware manufacturing guidelines for the so-called "instantly-available computer", as well as on software architectures that allow the PC to be put into different low-power modes according to the level of system activity, instead of turning it off completely. Their early work resulted in the Advanced Power Management (APM) specification [2], in 1992, which has evolved into a much broader, more complex specification called ACPI (Advanced Configuration and Power Interface), presented in conjunction with other industry leaders in 1996 [3]. Both the APM and ACPI specifications define a set of interfaces between power-manageable hardware and power-aware software. While the "appliance-PC" does not become a reality, these power management mechanisms are being used to help lower energy consumption on existing PCs.

But the power consumption of each computer viewed as a single, stand-alone entity is only the beginning of a much larger problem. With the ever-growing need for interconnection of devices over networks, more and more computers are being left on after regular office-hours, in an attempt to prevent disruptions in network services and inaccessibility to shared resources, or to allow administrative tasks (e.g. backups) to be performed during periods of inactivity. Network operating systems have clearly not been able to keep up with the advances in power management technology, and commonly still require that a host maintains its network connections in order to respond to periodic server queries. That leads to two immediate problems: first, depending on the network hardware installed on a PC, these periodic queries can cause enough activity to keep the computer awake, defeating power management; second, if the PC succeeds in entering a low-power mode, it might be unable to respond to the server, which will then assume the PC is off and terminate network services to it [4]. Certain newer network adaptors are equipped with a technology called "wake on LAN", which allows the computer to be turned on remotely upon reception of a special message (a "magic packet") [5], thus eliminating part of that problem. But the fact still remains that a large portion of networked equipment is unnecessarily left on during non-business hours, whether to comply with corporate guidelines or as result of simple misuse. A study conducted by Lawrence Berkeley National Laboratory (University of California), at two major metropolitan areas of the U.S. showed that only 44% of computers, 32% of monitors and 25% of printers in office environments are turned off at night [6].

Our goal is to investigate and explore exactly the network-related aspects of power management. We believe that energy, as much as any other shared network resource, should be managed from a global perspective of the entire LAN, and not only from the isolated view of each device. In this paper, we propose centralizing the configuration of power policies at a management station in order to minimize energy waste due to badly configured devices. The possibility of configuring power policies for several devices from a single point can be a valuable asset to network administrators, who are in the best position to decide which parts of the network are required to stay on during non-business hours, tailoring energy consumption down to the absolutely necessary. Furthermore, we believe that this capability to oversee and configure power consumption by networked devices should seamlessly integrate into existing network management platforms. Our approach is to utilize the Simple Network Management Protocol (SNMP [7]), the *de facto* standard management framework, and the existing power management capabilities of computer hardware, as the basis for a simple, yet efficient, network-oriented power management architecture.

This paper is organized as follows. Section 2 discusses pertinent related work. Section 3 presents our proposed architecture. Section 4 gives details of the implemented prototype environment. Section 5 presents a few experimental results. Section 6 concludes the paper.

2. RELATED WORK

It is important to point out that the existing power management features of computer hardware already make a significant contribution to energy savings. Since the creation of the Energy Star program by the United States Environmental Protection Agency (EPA), in 1992, a long way has been covered, and the vast majority of PCs manufactured today ships with some sort of power management capability. According to estimates made by the Energy Analysis Department of the Environmental Energy Technologies Division, at LBNL, based on data for 1999, power management saves about 23 TWh per year in the United States [1], which would correspond to US$ 1.95 billion at an average price of 8.5 cents per KWh. To give a clearer picture of just how much electricity that is, it would be enough to serve over 2 million average American households for an entire year [8].

The availability of power management features in computer hardware has allowed researchers to concentrate on the conception of mechanisms through which the low-level physical components can be driven to achieve the greatest possible savings [9, 10, 11, 12]. Others have explored the collaboration between different levels of software (e.g. operating system, device drivers and applications) to allow more efficient power management [13, 14, 15, 16]. But, even though those areas have been extensively investigated, there appears to be little work concentrating on ways to overcome the obstacles that network environments pose to power management.

There is, however, a research area where the paths of network and power management do cross. For many years, the UPS (Uninterruptible Power Supply) industry has been providing their customers with data protection systems that can trigger the unattended shutdown of computers in the event of utility power failures. These systems all share a common event-driven structure and have an inherent notion of power policies, even though those policies are based on a simplistic on-or-off approach. Their configuration typically associates the occurrence of energy-related events, detected through UPS monitoring, to the execution of a set of predefined actions, such as remotely shutting down specified computers or notifying a network administrator via e-mail. Communication between the monitoring station and the client computers is typically done via a proprietary protocol.

3. OVERVIEW OF THE POWER MANAGEMENT ARCHITECTURE

What we have done is to identify the potential of this industry-standard data protection architecture for adaptations that would make it suitable to perform power management tasks in a network environment. If we could extend the event association capabilities of the original architecture to allow the execution of predefined actions at specified times (e.g. "after 6 P.M., everyday"), we would be able to use the same communication infrastructure to perform coarse-grained (on-or-off based) remote power management of the computers in the network, reducing consumption during periods of inactivity. Additionally, if we could map this communication infrastructure to a standard network management protocol (such as SNMP), we would guarantee compatibility with existing management platforms and tools. Finally, if we could extend the set of possible actions to include intermediate low-power states for the remote hosts, we would have fairly flexible, fine-grained control over the overall use of energy in the network, which could have a positive effect not only on reducing energy waste but also on extending backup-power autonomy during utility power outages.

Those are the basic characteristics of the power management architecture we propose. We have been able to validate the feasibility of the model by implementing it in a prototype environment, which will be discussed in detail in Section 4. For now, it is enough to define the elements of the architecture, which can be seen as a specialized derivation of the traditional SNMP model, with *agents* residing on every manageable electronic device of the network and a *manager* that oversees their operation and determines when to apply changes in their power consumption, according to the pre-defined policies mentioned before. The agents hold the specific knowledge about the energy consumption characteristics of the devices they control, and communicate with the manager through SNMP, exchanging information as defined in a *management information base* (MIB) designed specifically for power management. Figure 1 depicts the main elements of the architecture.

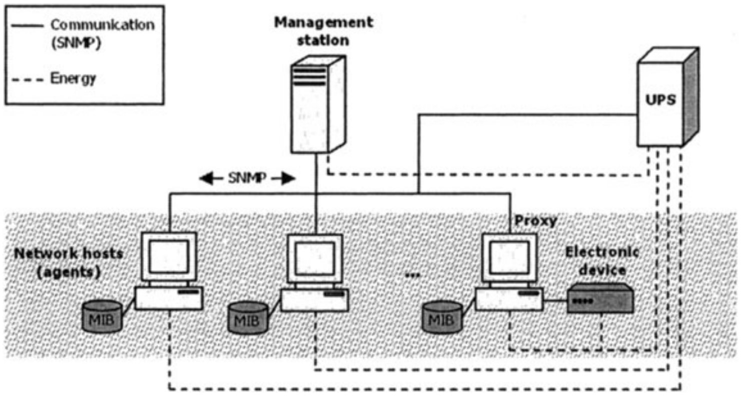


Figure 1. SNMP-based power management

In theory, any electronic device connected to the network can be managed, as long as it fulfills two basic requirements: first, it must be accessible via the TCP/IP network (either directly or through a proxy); second, it must provide some sort of power management capability that can be controlled by the agent (even if it is as simple as turning the device on or off, for example).

3.1 Power policies and the event-driven operation model

The basis for the operation of our proposed power management system is an event-driven model that is inspired in the reactive operation model of data protection systems. As we mentioned before, those systems act upon the occurrence of special conditions in the supply of energy to guarantee data integrity across the network. We call those conditions *energy events*. Energy events have intrinsic associated semantics. For example: "the external power has been disrupted" or "the level of the UPS batteries is low". We extend that model by defining *programmed events*, which can trigger the execution of actions at specified times. A programmed event does not have an inherent "type" like an energy event, but instead allows us to use an "alarm-clock" abstraction to initiate the execution of actions based on time criteria (e.g. "turn off the displays of all computers at 6:30 PM, everyday"). By adapting this abstraction to the original model, we are able to use a single configuration structure for both purposes. In other words, we can specify power policies that are applied in a reactive fashion either in response to energy events (to improve data protection) or according to predefined time criteria (to reduce energy waste).

Having a uniform association paradigm for both purposes also facilitates the dispatch of actions to the remote hosts, which can be implemented in a single component that handles the conversion of configured actions to their corresponding SNMP requests and transmits them to the appropriate destinations.

In our approach, power policies are specified as lists of actions that can be associated to either energy or programmed events. Five different types of actions are supported, as indicated in Table 1. Each action in a power policy is destined to a specific *target* – a device or group of devices where the action must be executed[1]. Except for the WAKEUP action, all others are mapped to an SNMP Set request intended to alter a specific object in the remote agent's MIB. The remote agent is responsible for interpreting the received value and executing the appropriate action, which is why we say the operation model is based on the association of events to *remote actions*.

Table 1. Supported action types and parameters

Action type	Parameters	Meaning
SHUTDOWN	-	Turn the device off
WAKEUP	-	Turn the device on
SET_POWER_STATE	component, state	Change the power state of the specified component to the specified state
RUN_COMMAND	command	Execute the specified command
SHOW_MESSAGE	message	Show the specified message to the user(s)

WAKEUP actions are used to remotely power on a device or bring it to an active state if it had been "sleeping". In those cases, the remote agent will not be running, therefore the correspondence with an SNMP message does not exist. Instead, a *Wake-on-LAN magic packet* is used to wake up the corresponding host. Obviously, only hosts that have a compatible network adapter will be able to detect the packet. Wake-on-LAN enabled adapters remain powered when the computer is turned off or put in a low-power state, and continually scan incoming network packets for a special data pattern. When that pattern is detected, the adapter triggers a boot sequence in the BIOS.

SET_POWER_STATE actions also deserve a little more attention. This category of actions was designed with flexibility in mind. Instead of defining a large fixed (and thus limited) set of state-changing actions for all possible components of a computer (e.g. one to turn off the monitor, another to spin down the hard disk, and so on), we chose to define a single, configurable type of action that can be used to request changes to the state of any component in a flexible manner. For each SET_POWER_STATE action, a hardware component name and the desired state must be specified. This approach allows the system to be universally compatible with different types of power management standards, such as APM or ACPI, for which specific agents could be implemented. There is no constraint on the names of components or states used in the configuration, except that every agent must support at least one component named "global", which represents the entire device. That means that the successful execution of an action depends on the correct

[1] We use the term "device" to refer to each *networked device* (e.g. a computer or another manageable electronic device), and "component" to refer to *hardware components* that may be individually controlled *within* a device.

interpretation of the messages by the agent, and on the semantic equivalence between the actions configured in the NMS and those supported by the agent.

3.2 Manager-agent communication

In typical SNMP applications, the network management station (NMS) is responsible for initiating communication with the remote agents to update its view of the network. In order to monitor the status of the network, the NMS is usually configured to perform periodic queries of every host's agent to determine its current state, including possible error conditions. This procedure is commonly referred to as *polling*.

However, this strategy would not be appropriate in our power management architecture, since the agents might frequently be unresponsive due to their hosts entering low-power states. On the other hand, as we have mentioned before, intensive incoming traffic could prevent the network hosts from entering such low-power states, which is another reason why polling is not an adequate approach. Instead, we use the alternative method: agent-initiated notification. In other words, whenever a relevant change in a computer's power state occurs, the agent must send an unsolicited notification to the manager. This avoids interference with the local execution of power management on each computer, and also reduces SNMP traffic considerably. The agents of those devices that are in an active state are also required to send periodic *"I'm alive"* notifications so that the manager can update its network map.

This passive behavior of the manager is only used for monitoring purposes, of course. Actual power management of the remote hosts requires that the NMS sends an SNMP `Set` request to the corresponding agents, which triggers the remote execution of the desired state change.

A thorough description of the Power Management MIB, which must be implemented by the agents, is included in [19].

3.3 Security

Unauthorized access to the power management agent on a host must be prevented, since it allows the requesting entity to perform operations that could result in unavailability of services provided by that host or to execute other potentially harmful actions. Luckily, version 3 of the Simple Network Management Protocol [17] provides both authentication and confidentiality, thus guaranteeing that only authorized requestors can access objects on an agent. Therefore, as long as an agent accepts only valid SNMPv3 requests, its presence in the system is no more of a threat than any other secure remote operation service (e.g. an SSH server).

After authorization, further processing of a request is entirely up to each agent's implementation. For example, an agent could make sure that only harmless commands are allowed to execute in response to a `RUN_COMMAND` action, preventing the management station to delete files, stop network services, and so forth.

4. PROTOTYPE ENVIRONMENT

Given the prohibitive cost of commercial network management software and the additional complexity of integrating a customized solution into such a platform, our approach was to develop a stand-alone management application, entirely in Java, using publicly available libraries. Additionally, a prototype agent has been developed for the Microsoft Windows 2000 platform, which provides one of the richest power management APIs amongst all PC operating systems. This section details the implementation of the prototype manager and agent.

4.1 The manager

The manager we have implemented performs the following basic tasks:

- monitoring of energy events reported by UPS devices;
- generation of programmed events at specified times;
- dispatch of remote actions in response to energy or programmed events according to the configuration;
- monitoring of power management agents for network map update.

Each of these tasks is delegated to a specialized component of the manager, as depicted in Figure 2. We will discuss the manager's components next.

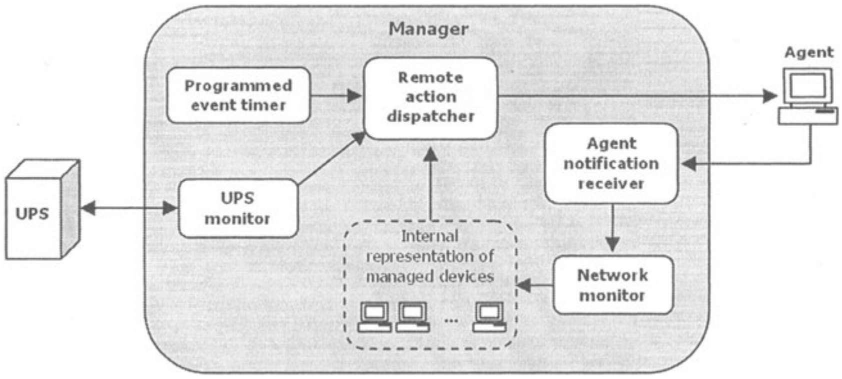


Figure 2. Composition of the manager

4.1.1 UPS monitoring

UPS devices can be monitored in several different ways, but the most typical are either via a serial line or via SNMP. Many vendors support the standard UPS-MIB [21] for SNMP management; others support their own specialized MIBs. We have implemented SNMPv1-based UPS monitors for the standard MIB and for two other vendor-specific MIBs that are used for supervision of UPS models available in our laboratories. In order to organize shared access to a common UDP port used for trap reception, we have implemented a single trap handling component that analyses incoming traps, identifies the source UPS agent, and forwards them to the appropriate monitor. This is easily accomplished by having all monitor classes implement a common interface – each monitor class "understands" a certain type of MIB, and is responsible for creating one or more monitor instances to handle several UPSs that can be managed through that MIB, at its own discretion. Using this approach, we can dynamically install and uninstall UPS monitors as small software plug-ins, without having to recompile the manager.

4.1.2 Programmed events

The other component capable of triggering the dispatch of actions is the programmed event timer. This timer is initialized from information in the manager's configuration file that describes the dates and times when specified actions should be automatically executed. Besides a scheduled execution time, a repetition rate can be specified for each programmed event. An event can be scheduled to repeat daily, weekly or monthly at the same time, which facilitates the configuration of typical power saving policies such as "automatically turn off all computer monitors after work, everyday".

4.1.3 Dispatch of remote actions

The remote action dispatcher is the central point for event processing in the manager. As we have briefly described before, it operates in response to either energy events reported by UPS monitors or programmed events generated by a timer. The basic task of the dispatcher, upon detection of an event, is to identify the actions associated with that event, convert them to the appropriate SNMP commands, and send them out to their respective targets. Since actions might be configured for delayed execution, the dispatcher also handles possible conflicts between pending actions and those scheduled for immediate execution.

4.1.4 Network monitoring

As we have mentioned before, the process of updating the network map in the manager is based on the reception of agent notification. The network monitor is the component responsible for processing incoming notifications in order to maintain an updated view of the network. Depending on the type of notification received and the current state of the originating device, the network monitor executes a full SNMP query of the agent's MIB to fill in relevant information about the device. The work performed by this monitor is important because the internal representation of the devices instruments the decision making process in the action dispatcher. Maintaining a faithful view of all power-managed devices also facilitates graphical representation of the network for administrative purposes.

4.1.5 Configuration

Configuration of the manager is done via the Extensible Markup Language (XML), which has become a widely used format for this purpose because of its hierarchical structure and broad support in all major programming languages. Besides providing a relatively large set of application options (e.g. UDP ports, timeout limits, logging options, etc.), the main goal of the configuration file is to describe the elements of the network and the associations between events and action lists (i.e. power policies). A group metaphor is provided to facilitate the configuration process – devices can be arranged according to arbitrary logical criteria, such as hardware similarity, common power management features, etc. Figure 3 shows sample sections of the configuration file illustrating the general process for defining a time-based power policy.

```
...
<device name="pc1" address="10.0.0.5"/>
<device name="pc2" address="10.0.0.6"/>
<device name="pc3" address="10.0.0.7"/>
...
<group name="pcsGroup">
     <group-member ref="pc1"/>
     <group-member ref="pc2"/>
     <group-member ref="pc3"/>
</group>
...
<programmed-event name="Automatic monitor night turn-off"
                  startTime="07/01/2002 19:00:00"
                  repeat="DAILY">
     <action target="pcsGroup" type="SET_POWER_STATE">
          <param name="component" value="display"/>
          <param name="state" value="off"/>
     </action>
</programmed-event>
...
```

Figure 3. Sample manager configuration

4.1.6 Graphical operation

Besides operating in the unattended mode described so far, the manager can also be controlled through a graphical tool, via RMI (Java's Remote Method Invocation API). This allows the administrator to visualize the network map and execute remote actions on demand.

4.2 The agent

In order to test and evaluate the power management architecture, we have also implemented a prototype agent for the Microsoft Windows 2000 operating system. This agent can perform the following basic functions:

– detection of power management-related system messages;
– notification of the manager upon system power-on, transition to low-power states, and resumption from low-power states (wake-up);
– processing of SNMP `Set` requests that allow the manager to command the host's transition to three different states: power-off, standby, and hibernation.

Starting with Windows 2000, Microsoft's operating systems have vast support for power management operations. Applications are notified of changes in the power status of the computer through a specific system message, WM_POWERBROADCAST, which can indicate several different events (e.g. "system entering low-power mode", "system resuming from low-power mode", etc.). Our prototype agent simply interprets these messages as they are received from the OS, sending out the appropriate SNMP notifications when necessary.

The agent understands three different values for the global state of the computer: "off"; "standby" (also referred to in the literature as "suspend to RAM", a state in which memory contents might be lost due to abrupt loss of power); and "hibernate" (or "suspend to disk", when memory contents are safely transferred to non-volatile storage and the PC is completely powered down).

We have also tested the ability to remotely power on the machine by sending it a "magic packet" from the management station, and ensured proper agent behavior upon successful remote wake-up.

5. EXPERIMENTAL RESULTS

A series of experiments allowed us to validate the feasibility of the proposed architecture and to estimate the potential economy that may result from effectively deploying the system. The most important results are summarized in the following sections.

5.1 Estimates of energy savings during non-business hours

Perhaps the most appealing reason for a power management system that can be integrated into and take advantage of existing network management platforms is the enormous potential for power savings during non-business hours. As we have mentioned previously, an LBNL study released in 2001 estimates nightly turn-off rates for office equipment to be considerably low (44% for PCs and 32% for monitors) [6]. That same study estimates that only 3 to 8% of computers and 38% of monitors are put into low-power states after office-hours, leaving 30% of monitors and over 50% of computers that are potential candidates for power management, not to mention other equipment such as printers, copiers, fax machines and so forth, all of which could potentially be power-managed remotely.

Considering a typical 9-to-5 workday[2], an average of 250 workdays per year, and an average price for electricity of 8.5 cents per KWh, we can make the following (quite obvious, yet frequently overlooked) additional observations:

- each desktop PC station (CPU and monitor) consumes about 1.2 KWh during a typical workday, if it is permanently active[3]; that amount of energy equals a cost of US$ 25.50 per year;

[2] We consider the actual length of a "9-to-5 workday" to be 9.5 hours for more realistic estimates.

[3] We consider the following average consumption values in active, low-power, and off modes for a common desktop and a 15 inch monitor, respectively: 50/75 Watts; 25/5 Watts; and 1.5/0.5 Watts.

- if each PC is kept completely active during non-business hours, it consumes an additional 1.8 KWh of electricity per day – an unnecessary expense of US$ 38 per year;
- if the monitor alone is put into a low-power state after office-hours, US$ 21.50 can be saved each year, per PC;
- if the entire PC is put into a low-power state at the end of a workday, a total of US$ 29 can be saved per PC each year;
- turning off the PC completely allows even further savings: U$ 36.50 a year.

5.2 Increase in UPS autonomy during power outages

We have conducted measurements in a typical network setup in order to determine the potential increase in UPS autonomy during utility power outages. Our experimental environment consisted of a small Ethernet LAN of 10 PCs (one of which acted as a server, running the manager), all drawing power from a 2 kVA UPS equipped with 12 batteries that can supply 9Ah each.

These measurements correspond to backup power autonomy during a blackout, in three different scenarios. In the first scenario, the manager was not configured to react to the start of battery-powered operation, and all 10 PCs remained active until the batteries were depleted. In the second scenario, the manager was instructed to put the 9 client PCs in standby mode. In the third and last scenario, the manager was configured to power down the same 9 PCs. The average measured consumption per PC is 125 Watts in active mode, 30 Watts in standby mode, and 6.5 Watts in soft-off mode.

The results of those three rounds of measurements are presented in Figure 4. The graphic shows that autonomy increases by a factor of 3.5 between scenarios *a* and *b*, and by a factor of 7 between *a* and *c*. It should be noted that we have omitted possible user intervention in the process of altering the power state of client PCs in this experiment. In other words, the picture represents an ideal case, in which no users were present at the time of the power failure, or in which all users peacefully accepted the power management agent's request to power off the PC or put it in standby mode. It is likely that, in a real-life situation, most stations would be in use and would have to remain active for some time so that users could finish pending tasks. Nevertheless, it is clear that the potential for reduced consumption during a power failure exists, and could be better explored through the use of an automated solution, such as the one we have presented. Assuming a power failure during regular business-hours, the increase in autonomy would most probably appear in intermediate ranges of the graphic, but would still help amplify the chances of utility power returning before UPS batteries were depleted, avoiding a complete network shutdown. We have yet to conduct measurements in such circumstances to obtain precise numbers.

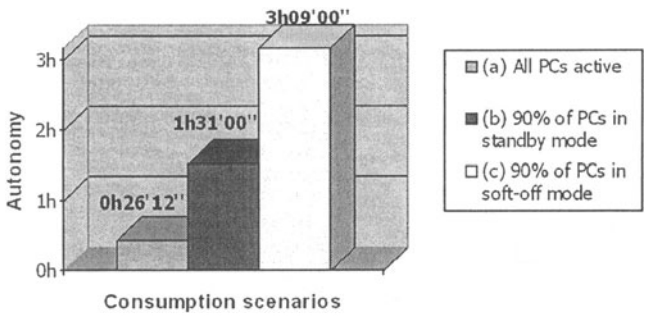


Figure 4. UPS autonomy in different consumption scenarios

6. CONCLUSION

This paper described the design and implementation of a power management architecture for local area networks. The architecture is based on the SNMP management framework, relying on *agents* that reside on each power-manageable device of the network, and whose operation is supervised by a central element, the *manager*. Manager and agents communicate via SNMPv3, a standard protocol which provides the required security characteristics, and exchange information according to a *management information base* specifically designed for power management.

Our main goal is to facilitate power management in network environments, where many factors might interfere with successful execution of power management on each device, particularly poor configuration and operation patterns of network systems and services. We believe that the network administrator is in a privileged position to determine which devices are best suited for power management, given his knowledge of the services that run on the network and their requirements, and would be in an even better position to define power consumption policies for the various devices if it could be done from a central point, which provides a global view of the entire LAN.

We do not intend to replace local power management (i.e. power management that is executed on each device independently of the influence of the manager). In fact, we encourage power management features to be enabled and functional on all electronic devices of the network to obtain the greatest possible energy savings. Our goal is simply to aid in the configuration of power policies in the network environment, possibly acting on those devices that have not been correctly set up for local power management. The architecture is designed so as to prevent interference with local execution of power management.

There are several possible areas for further exploration within the context of this research. The most prominent ones are probably the development of full-featured agents for a wider variety of operating systems and integration of the stand-alone management application with an existing network management platform, such as Hewlett-Packard's OpenView, for example.

REFERENCES

[1] K. Kawamoto et al. Electricity Used by Office Equipment and Network Equipment in the U.S.: Detailed Report and Appendices. LBNL-45917. Lawrence Berkeley National Laboratory, University of California. February 2001.

[2] Intel Corp. and Microsoft Corp. Advanced Power Management (APM) BIOS Interface Specification. Rev. 1.2. 1996. http://www.microsoft.com/hwdev/archive/BUSBIOS/amp_12.asp

[3] Compaq Computer Corp., Intel Corp., Microsoft Corp., Phoenix Technologies Ltd., Toshiba Corp. Advanced Configuration and Power Interface specification. 1996. http://www.acpi.info/index.html

[4] B. Nordman et al. User guide to power management for PCs and monitors. LBNL-39466/UC-1600. Lawrence Berkeley National Laboratory, University of California. January 1997.

[5] ADVANCED MICRO DEVICES, INC. Magic Packet technology white paper. 1995. http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/20213.pdf

[6] C. A. Webber et al. Field surveys of office equipment operating patterns. LBNL-46930. Lawrence Berkeley National Laboratory, University of California. September 2001.

[7] J. Case et al. A Simple Network Management Protocol (SNMP). RFC 1157.

[8] Energy Information Agency, United States Department of Energy. A Look at Residential Energy Consumption in 1997. DOE/EIA-0632 (97), p.17. November 1999.

[9] F. Douglis, P. Krishnan and B. Marsh. Thwarting the Power Hungry Disk. In Proceedings of the 1994 Winter USENIX Conference, pp.293-306, January 1994.

[10] R. Kravets and P. Krishnan. Power Management Techniques for Mobile Communication. In Proceedings of the 4th International Conference on Mobile Computing and Networking (MOBICOM98), pp.157-168, October 1998.

[11] J. Lorch and A. J. Smith. Reducing processor power consumption by improving processor time management in a single-user operating system. In Proceedings of the 2nd ACM International Conference on Mobile Computing (MOBICOM96), pp.143-154, November 1996.

[12] M. Stemm and R. Katz. Measuring and Reducing energy consumption of network interfaces in hand-held devices. In Proceedings of the 3rd International Workshop on Mobile Multimedia Communications (MoMuC-3), September 1996.

[13] C. S. Ellis. The Case for Higher-Level Power Management. In Proceedings of the 7th Workshop on Hot Topics in Operating Systems, March 1999.

[14] J. Lorch and A. J. Smith. Software Strategies for Portable Computer Energy Management. IEEE Personal Communications Magazine, v.5, n.3, pp.60–73, June 1998.

[15] Y. Lu, T. Simunic and G. De Micheli. Software controlled power management. In Proceedings of the 7th International Workshop on Hardware/Software Codesign, pp.157-161, Rome, Italy, May 1999.

[16] A. Vahdat, A. R. Lebeck, C. S. Ellis. Every Joule is precious: the case for revisiting operating system design for energy efficiency, In Proceedings of the 9th ACM SIGOPS European Workshop, September 2000.

[17] J. Case et al. Introduction to version 3 of the Internet-standard Network Management Framework. Request for Comments 2570. 1999.

[18] J. Case et al. UPS Management Information Base. RFC 1628. 1994.

[19] L. F. Pollo. Power management system for local area networks (in Portuguese). Master's thesis. PPGC – UFRGS. Porto Alegre, Brazil, 2002. http://www.inf.ufrgs.br/~pollo/netpower/

Erratum to: Integrated Network Management VIII

Germán Goldszmidt[1] and Jürgen Schönwälder[2]

[1] IBM Research, USA
[2] University of Osnabrück, Germany

Erratum to:
G. Goldszmidt and J. Schönwälder (Eds.)
Integrated Network Management VIII
DOI: 10.1007/978-0-387-35674-7

The book was inadvertently published with an incorrect name of the copyright holder. The name of the copyright holder for this book is: © IFIP International Federation for Information Processing. The book has been updated with the changes.

The updated original online version for this book can be found at
DOI: 10.1007/978-0-387-35674-7

PANELS

Co-Chairs: Nelson Fonseca
University of Campinas, Brazil

Felix Wu
University of California at Davis, USA

Panel 1:

Web-Services for Internet Management: Yet Another Hype?

Chair: Aiko Pras
University of Twente, The Netherlands

Panel 2:

Overlay Networks and Management: Real Solution or New Hype

Chair: Ehab Al-Shaer
De Paul University Chicago, USA

Panel 3:

Resource Management for Enterprise Application Grids

Chair: Jerry Rolia
Hewlett-Packard Laboratories, USA

Panel 4:

Security and Privacy: How Can We Resolve the Conflicts of Law Enforcement and Freedom Lovers?

Chair: Rob Kolstad
SAGE, USA

Panel 5:

On Skepticism of Intrusion Detection Technologies

Co-Chairs: Marc Dacier
Eurecom, France

Felix Wu
University of California at Davis, USA

A synopsis of these panels will be posted to http://www.im2003.org after the conference.

Author Index