Elena Gaura
Lewis Girod
James Brusey
Michael Allen
Geoffrey Challen
Editors

Wireless Sensor Networks

Deployments and Design Frameworks

Springer

Wireless Sensor Networks

Elena Gaura • Lewis Girod • James Brusey
Michael Allen • Geoffrey Challen
Editors

Wireless Sensor Networks

Deployments and Design Frameworks

Springer

Editors
Elena Gaura
Coventry University
Fac. Engineering and Computing
Cogent Computing Applied
Research Centre, Priory Street
CV1 5FB Coventry
United Kingdom
e.gaura@coventry.ac.uk

Lewis Girod
Massachusetts Institute of Technology
Computer Science and Artificial
Intelligence Lab. (CSAIL)
Vassar St. 32
02139 Cambridge Massachusetts
Stata Center, Bldg. 32
USA
girod@nms.csail.mit.edu

James Brusey
Coventry University
Fac. Engineering and Computing
Cogent Computing Applied
Research Centre, Priory Street
CV1 5FB Coventry
United Kingdom
j.brusey@coventry.ac.uk

Michael Allen
Singapore MIT Alliance
for Research and Technology
S16-06-10, 3 Science Drive 2
Singapore 117543
michael@smart.mit.edu

Geoffrey Challen
Harvard University
School of Engineering
and Applied Science
Oxford St 33
02138 Cambridge Massachusetts
USA
challen@eecs.harvard.edu

ISBN 978-1-4419-5833-4 e-ISBN 978-1-4419-5834-1
DOI 10.1007/978-1-4419-5834-1
Springer New York Dordrecht Heidelberg London

Library of Congress Control Number: 2010935725

To those who left (Mosu and Cary), the newly arrived (Sascha and Ellery) and the ones who stood by us (Stephanie and Sarah).

Foreword

The twentieth century ended with the vision of *smart dust*: a network of wirelessly connected devices whose size would match that of a dust particle, each one a self-contained package equipped with sensing, computation, communication, and power. Smart dust held the promise to bridge the physical and digital worlds in the most unobtrusive manner, blending together realms that were previously considered well separated. Applications involved scattering hundreds, or even thousands, of smart dust devices to monitor various environmental quantities in scenarios ranging from habitat monitoring to disaster management. The devices were envisioned to self-organize to accomplish their task in the most efficient way. As such, smart dust would become a powerful *tool*, assisting the daily activities of scientists and engineers in a wide range of disparate disciplines.

Wireless sensor networks (WSNs), as we know them today, are the most noteworthy attempt at implementing the smart dust vision. In the last decade, this field has seen a fast-growing investment from both academia and industry. Significant financial resources and manpower have gone into making the smart dust vision a reality through WSNs. Yet, we still cannot claim complete success. At present, only specialist computer scientists or computer engineers have the necessary background to walk the road from conception to a final, deployed, and running WSN system. Nevertheless, the path unfolds into a huge design space, paved with a number of potential issues, challenges, and pitfalls. Devices cannot be randomly scattered in most cases, and may still require a significant amount of manual tuning to provide satisfactory performance. Probably, no other field in computer science or computer engineering has hitherto faced similar challenges. Most of the challenges stem from the intimate interactions between the WSN and the real world that it is immersed in.

As researchers in WSNs, we are confronted with these issues on a daily basis. As an example, we recall the weeks spent working on a structural monitoring deployment with lab-mates and advisers, trying to figure out the reason why nodes kept rebooting at a specific hour in the evening. After lengthy and painful debugging sessions, having hypothesized many esoteric explanations for the weird behavior, we discovered that a kind cleaning lady, coming every day at the same time, used to spray soap over our nodes, which caused them to reboot. The people involved in the project learned in this manner the importance of packaging, even during the testing

phases. Nevertheless, all WSN researchers who dare to push their research up to the deployment phase must continue to learn lessons like this every day ... the hard way!

This book comes as an invaluable aid to address the issue above by providing a unified view on the methodologies and best practices for deploying WSN systems. We believe that this book will work as a guide for researchers and practitioners to drive their own WSN-related activities. In Part I of the book, based on a number of successful experiences in the field, the editors elicit the concept of "design for deployment". Indeed, by considering the issues tied to the deployment activity early on, the design space shrinks and several pitfalls and issues can be anticipated, saving effort, time, and money.

Part II of the book, on the other hand, includes seven contributions from authoritative researchers in the field describing their own experiences in deploying WSNs. Every contribution was carefully chosen by the editors to highlight specific facets of the subject matter. Challenges such as data losses, sensor miscalibration and failures, harsh environments, large-scale systems, and heavily-regulated environments are covered by pointing out the specifics of the issue at hand and the various (not necessarily successful) solutions applied. Learning from others' mistakes is as valuable as reading of their successes.

We are thus confident that "Wireless Sensor Networks: Deployment and Design Frameworks" has the potential to be a stepping stone for both WSN beginners and also more experienced researchers and practitioners, bringing the Wireless Sensor Network outside the computer laboratory and applying it in the real-world, fulfilling its visionary role as a useful tool for scientists and engineers.

Stockholm, Sweden — Luca Mottola
March, 2010 — Thiemo Voigt

Preface

Wireless sensor networks are one of the most promising technologies of the new millennium. The opportunities and challenges of programming networks of small, lightweight, low-power, computation- and bandwidth-limited nodes have attracted a large community of researchers and developers. This technology's unique set of capabilities and limitations produces an exciting and complex design space, but one that can easily overwhelm newcomers to the field. In addition, deploying sensing into the physical environment produces its own set of challenges and can push systems into failure modes and problems that are difficult to discover or reproduce in the laboratory.

We believe that deployment is an irreplaceable step in the process of developing sensor network hardware and software solutions. This book aims to facilitate the development and deployment of new embedded sensing solutions by collecting a wealth of practical design, development and deployment experience from prior efforts. To this end, seven chapters provide detailed overviews of deployed systems, each with a specific message or focus. Examining the successful projects presented here reveals a unifying approach: *design for deployment*. Incorporating deployment into the design process exposes a different set of requirements and considerations. Low data fidelity or sparse data, harsh deployment environments, data rates exceeding the network's communication capabilities, limited in-situ debugging opportunities, and end-user requirements and usability issues are examples of challenges that arise in even the most carefully planned deployments.

By designing with deployment in mind, many of these stumbling blocks can be avoided. Part I of the book lays out the design for deployment approach in detail and attempts to formulate a useful body of knowledge for the practical developer. These chapters advise on:

1. Choosing an appropriate design view and hardware base as the starting point for development in a particular project,
2. Planning and accounting for the deployment environment,
3. Making best use prototyping and iteration, and
4. Mitigating the design space in terms of its key parameters – data rate, budget, network size, maintenance, deployment environment and usage model.

The case studies found in Part II cover:

1. Effective implementation of data-reduction algorithms for high data-rate applications,
2. Incorporating in-network processing into real-time sensing and actuation systems,
3. Dealing with data loss, sensor failure and system maintenance,
4. Containing cost in high data rate applications by using hybrid topologies,
5. Deploying large-scale networks,
6. Developing for harsh environments, and
7. Satisfying stakeholders in heavily-regulated environments.

Taken as a whole, this book provides a deployment-driven entry point for those new to sensor networks as well as a wealth of practical experiential knowledge useful to the seasoned practitioner. Our hope is that the information presented here will lead to more – and better – future sensor network deployments.

Coventry, UK
Boston, USA
2010

Elena Gaura
Lewis Girod
James Brusey
Michael Allen
Geoffrey Challen

Acknowledgments

As this book draws heavily from deployers' experience, it has been contributed to both formally and informally by a large number of researchers. The editors are grateful to all.

Dr Thiemo Voight and Dr Luca Motolla from the Swedish Institute of Computer Science (SICS) have provided insightful and thorough reviews for all chapters. We would like to acknowledge their technical input and guidance.

Application chapters authors have kindly agreed to share, in their contributions, both the successes and failures encountered during deployment of their systems. Colleagues from various groups helped identifying interesting applications and aspects of deployment and design that are of most use to the practical scientist.

Last but not least, we are thankful to the growing community of real-life systems designers for the encouragement to produce this book, and those who have shaped our way of thinking about embedded systems design, championing the need for practical applications and experience.

Contents

Contributors

Michael Allen Singapore-MIT Alliance for Research and Technology, S16-06-10, 3 Science Drive 2, Singapore 117543, michael@smart.mit.edu

Anish Arora Department of Computer Science and Engineering, Ohio State University, Columbus, OH, USA, anish@cse.ohio-state.edu

James Brusey Cogent Computing Applied Research Centre, Coventry University, Coventry, UK, j.brusey@coventry.ac.uk

Nirupama Bulusu Portland State University, Portland, OR 97201, USA, nbulusu@cs.pdx.edu

Geoffrey Challen Harvard University, School of Engineering and Applied Science, Oxford St 33, 02138 Cambridge Massachusetts, USA, challen@eecs.harvard.edu

Chun Tung Chou University of New South Wales, Sydney, Australia, ctchou@cse.unsw.edu.au

Dorothy Curtis Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139, USA, dcurtis@csail.mit.edu

Thanh Dang Portland State University, Portland, OR 97201, USA, dangtx@pdx.edu

Elena Gaura Cogent Computing Applied Research Centre, Coventry University, Coventry, UK, e.gaura@coventry.ac.uk

Lewis Girod Massachusetts Institute of Technology, Computer Science and Artificial Intelligence Lab. (CSAIL), Vassar St. 32, 02139 Cambridge Massachusetts, Stata Center, Bldg. 32, USA, girod@nms.csail.mit.edu

Jane K. Hart University of Southampton, Southampton, Hampshire SO17 1BJ, UK, jhart@soton.ac.uk

Wen Hu CSIRO, Brisbane, Australia, wen.hu@csiro.au

Sanjay Jha University of New South Wales, Sydney, Australia, sjha@cse.unsw.edu.au

Kirk Martinez University of Southampton, Southampton, Hampshire SO17 1BJ, UK, km@ecs.soton.ac.uk

Vinayak Naik Indraprastha Institute of Information Technology, Delhi, India, naik@iiitd.ac.in

Lucila Ohno-Machado Division of Biomedical Informatics, University of California San Diego, La Jolla, CA 92093, USA, machado@ucsd.edu

Esteban Pino Departamento de Ingeniería Eléctrica, Universidad de Concepción, Concepción, Chile, epino@ieee.org

Thomas Schoellhammer The Center for Embedded Networked Sensing, UCLA, Los Angeles, CA, USA, tschoell@cs.ucla.edu

Thomas Stair Department of Emergency Medicine, Brigham and Women's Hospital, Harvard Medical School, Boston, MA 02115, USA, tstair@partners.org

Andrew Taylor University of New South Wales, Sydney, Australia, ataylor@cse.unsw.edu.au

Van Nghia Tran University of New South Wales, Sydney, Australia, vantran@cse.unsw.edu.au

Matt Welsh Harvard School of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02139, USA, mdw@eecs.harvard.edu

Part I
Wireless Sensor Networks Design for Deployment

Chapter 1
Introduction

Elena Gaura, Michael Allen, Lewis Girod, James Brusey, and Geoffrey Challen

Abstract The chapter introduces the topical focus and aims of the book. The book's practical, experience driven outlook onto the domain of embedded wireless networked sensing systems is justified and the potential gains for the reader are highlighted. The authors promote the adoption of a deployment based, experiential research approach as key to the successful development of new wireless sensing applications. By providing a map of the practical design and deployment problems treated in various chapters, the introduction aids the reader identify at a glance where the issues most related to their current work are discussed within the book.

1.1 The Promise and the Challenge of Sensor Networks

Since the late 1990s, the potential of Wireless Sensor Networks (WSNs) to revolutionise our understanding of and interaction with the environment has captured the imagination and enthusiasm of both academia and industry. Experimental applications have been proposed to address virtually every aspect of society from scientific research and compliance verification to health care and industrial monitoring. These applications have provided motivation for the development of WSN systems and theory, leading to an exciting period of research, with many theoretical achievements and much promise.

However, while many of these applications have been explored through proofs of concept, few have passed successfully into the commercial domain. The difficulty in making this transition comes from several sources. While some of these are technical challenges in the computer science and engineering domains, there are also many challenges for which computer science researchers are less well-prepared, such as managing deployment logistics and a deep understanding of the needs and problems of unrelated disciplines. Unfortunately, these non-CS challenges are equally critical

E. Gaura (✉)
Cogent Computing Applied Research Centre, Coventry University, Coventry, UK
e-mail: e.gaura@coventry.ac.uk

E. Gaura et al. (eds.), *Wireless Sensor Networks: Deployments and Design Frameworks*,
DOI 10.1007/978-1-4419-5834-1_1,

impediments to defining, developing, deploying and commercializing WSN applications. A few authors, such as Langendoen et al. [3], have published discussions of the surprises that lie in wait when bringing a system from the lab to the field, but in general these "negative" results are underrepresented. This book is intended to provide a comprehensive discussion of design and deployment problems and provide some guidance for future development.

In the early stages of WSN research, few proposed applications had well-characterized, domain-specific performance requirements. In the absence of clear specifications, many WSN systems were either developed as examples to showcase a particular technology, or were developed to support a very specific application. By and large, projects suffered from the intrinsic difficulties in making these two disparate research programs meet.

Today, as the vision of WSNs has grown more widespread, there is no shortage of well defined applications that fit the model of low-cost distributed sensing systems. To give but a few examples, climate-change and environment-related sensing applications form a rich domain, which includes improving the understanding and modeling of the carbon cycle, and detecting and monitoring air and water borne pollutants in urban and agricultural environments. Cyber-physical systems form another broad application area which focuses on systems to improve the safety and efficiency of infrastructure.

With such compelling applications on the horizon and a ten year research history, we would like to be able to proclaim that building new applications is simply a matter of integrating existing components. Unfortunately, while there has been enormous progress to date, the published research turns out to be less valuable than one might hope: delivering a working fielded application still requires an expensive and specialized research and development effort. The reason for this is circular, rooted in the diversity of *potential* applications over the last decade, relative to *actual* application of the technology. In the absence of clearly defined application needs and a robust industrial market, many of the components that have been developed thus far have been either too generic to be readily applied, or well specified but targeted at a divergent application.

This situation presents both challenges and opportunities. The opportunities are clear: there is great political drive to implement systems to address society's pressing needs. At the same time the challenges must not be underestimated. Successfully deploying environmental or cyber-physical applications, for example, will require understanding and solving complex, cross-disciplinary problems, in addition to the problems that arise in the computer science and engineering domain. The logistical difficulties of deploying and testing these systems under realistic conditions need careful consideration: prototype deployments are generally required to work out the many aspects of systems that can not be addressed in simulation; these deployments will require another set of tools to aid in deployment, eg. methods of visualizing the state of the running system with constrained network access, software update processes to support rapid iteration, etc. Many of the parameters of such systems will also need to be tuned and validated based on data that must ultimately be collected from fielded nodes. Fortunately, researchers can be aided in this effort by studying the prior experience and practice in prosecuting WSN deployments.

The goal of this book is thus to document the experiences of some successful WSN deployments through case studies, and to refine that knowledge into guidance for those striving to resolve real-life applications using wireless sensing. We feel, as domain practitioners, that it is essential to reflect upon and understand the challenges that have been faced thus far in the advancement towards the vision of pervasive and ubiquitous wireless sensors. Although we have come some way, long and strenuous development cycles are still needed to deliver production-ready WSN systems that meet real needs. The authors hope that the book will ease the development effort for both new applications and new practitioners.

1.1.1 Goals of this Book

This book's primary aim is to help the practitioner to approach the design of WSN-oriented application systems from an informed standpoint. To be successful in development and deployment of WSNs, it is important to build upon the past achievements and experiences of others, and to follow examples of good design; this book enables both. We hope that the book will help the practitioner avoid some of the common pitfalls of first time designs and deployments and set realistic expectations of success at various stages of the design–prototype–deploy cycle. To meet these aims, the book:

1. Reviews the state of the art with respect to WSN practical advances and the drivers and roadblocks for wide WSN technology adoption
2. Presents seven notable real-life WSN *designs for deployment* and the associated *researchers' experience*
3. Provides informed, systematic support towards new application design and implementation, for those working towards shorter *development for deployment* cycles
4. Synthesizes *strategies for design and deployment*, based on specific experiences presented as case studies, that can be applied to new projects in academia and industry
5. Highlights key issues and trade-offs encountered when mitigating constraints in the *WSN design parameter space* for real-life applications.

1.1.2 How can this Book Help the Beginner WSN Practitioner?

This book sprang from the authors' desire to share the joy and frustrations experienced while designing and deploying WSN systems. Over the team of editors and contributors, it seems that the observation "everything that *can* go wrong *will* go wrong" applied well to their real-life WSN deployments. It follows that many of the common problems encountered leading up to and during real-life deployments of WSNs can be better understood if deployment experiences, lessons and best-practice approaches are shared within the community.

Many times the experience gained in a deployment spurs algorithmic and system design advancements with wider impact and applicability than the specific motivating application context. Several of the case studies in this book contain examples of these types of discovery, for example: (1) the development of Vigilance, a system designed to monitor deployments and expose the impact that missing data has on the applications expecting it (Chap. 6); (2) the development of Lazy Grouping and Adaptation, sophisticated in-network processing algorithms to resolve latency issues in data-intensive acoustic monitoring applications (Chap. 5); (3) the development of Lance, a wide applicability design framework for WSN applications (Chap. 4).

The editors' exposure to WSN projects conducted by research teams across the world revealed that many projects are ending short of full blown in-situ deployments for numerous reasons:

- Time and budgetary restrictions limiting practical implementation and deployment.
- Difficulty in translating the theoretical ideas into a deployable prototype.
- Taking an inflexible approach to design and development that leaves little scope for dynamic adjustment.
- Lack of certain, important experience in development teams (common examples are systems and hardware development experience and concrete application motivation and expertise).
- Ignoring the deployment process and target environment during the design process.
- Expecting complete success in a first-time deployment rather than planning for iteration.

Thus, in an attempt to minimize the problems above, the book has the following contributions:

1. It brings together reports of *successful deployments*, consolidates individual application experience, and identifies general approaches and lessons to be learned; the seven application-oriented chapters focus on the *design processes* followed by the application designers, identifying *best practice* and framing the *generic concepts* and wider applicability of the respective implementations.
2. It motivates the use of *application centric approaches* for the rapid development of WSN systems.
3. It provides the reader with *practical hints* and *strategies* for design, development and deployment planning, and guides technology choices.
4. It brings forth several generic concepts, tools and algorithms developed as an outcome of deployment experience and thorough understanding of real-life applications needs.

The strength of this book lies in its practical nature: we believe that the best way to convey important WSN concepts and ideas is through experience with real-world systems. This need for real-world experience leads us to conclude that taking an *application centric* viewpoint on the development of WSN systems is the key to

success, not only for the small and medium scale applications considered in this book but also towards larger, more densely deployed and miniaturized WSNs.

The authors base their writing on a wealth of both practical and theoretical experience with WSN systems and the belief that research effort should not go undirected. Much of the next few years' effort in the domain must be focused towards wider adoption of the technology by increasing trust in its value and demonstrating its robustness and reliability. To achieve this, experience of applying WSN technology in earnest to real-world problems must become more widespread within the WSN research community.

It is assumed that the reader is already familiar with wireless sensor networks as a technology, but not experienced in the *development for deployment* aspects of WSN-based applications. Furthermore, it is assumed that the reader wants to quickly understand the underpinning concepts necessary for developing their own real-life, application-oriented WSN system, and wants to be able to approach the deployment process with a realistic view of the common pitfalls, expectations, and measures for success.

The book is *design* and *deployment*-oriented: as such, it does not provide a full treatment of popular WSN research areas, such as energy management, media access for wireless communication or routing protocols. To further understand these areas, the interested reader should consult the following books:

- Wireless Sensor Networks: A Systems Perspective by Bulusu and Jha [1]
- Principles of Embedded Networked Systems Design by Pottie and Kaiser [4]
- Protocols and Architectures for WSNs by Karl and Willig [2]
- Wireless Sensor Networks: An Information Processing Approach by Zhao and Guibas [5].

1.2 Guide to Using this Book

This book has two distinct parts and the reader is encouraged to read them in either order: going from the generic to specific (Part I followed by Part II) or by first selectively reading Part II chapters that treat issues of interest.

1.2.1 Part I: Design Strategies for Deploying Sensor Networks

The first part of this book discusses strategies for designing deployable sensor networks, culled from the experiences documented in the case studies in Part II.

Chapter 2 takes a holistic view on the main trends and drivers for both the research achievements in the WSN domain and the WSN productization efforts. The chapter introduces the characteristics of deployed WSN applications, discusses how they differ from lab-based experiments and distills some general design strategies

from past deployment experiences, with particular reference to applications featured in Part II of the book. A survey of platforms that are currently commercially available to use as starting points for WSN applications is presented, as well as a discussion on the past and current markets for WSN applications and products. The chapter explains the tensions between the theoretical and practical achievements, between the vision-led and industrially focused system development approaches, and between advocacy and the true market response, with a view of promoting a practical development path for forthcoming WSN applications.

Chapter 3 approaches the design of real-world WSNs from the perspectives of software life cycle and key design parameters. The chapter highlights the value of iterative development-deployment cycles and synthesizes essential lessons delivered in the application specific chapters in Part II. It brings to the fore the key constraining factors which will influence most of the WSN applications design decisions. The most commonly encountered design parameters and the trade-offs they ensue are treated, together with examples of how they shaped the system design choices in the applications reported in this book.

1.2.2 Part II: Case Studies

Chapters 4 to 10 describe seven real-life, application-led case studies selected from among the most successful and exciting real-life deployments reported since 2004. Each of the chapters has a different set of core constraints that have influenced the application design, such as size, cost, interactivity, network lifetime and human factors. Together, the Part II chapters cover a wide range of monitoring-based application classes, from soil and animal habitat monitoring, to volcano, glacier, border and patient monitoring applications. Each application faces different requirements, reflected in the choices of system design, implementation and hardware platform. When considered in the context of new applications design or implementation, these chapters provide a comprehensive treatment of the most commonly encountered issues in WSN systems development.

To guide the reader in identifying the most relevant material, we present a structured overview of the Part II chapters in the following three tables. This summary is intended to give the reader an understanding of the issues that differentiate the systems built in each of the chapters, as well as the common challenges that these applications faced and the constraints that influenced their design, implementation, testing and deployment.

Table 1.1 summarizes the most important parameters differentiating the chapters. Note that each chapter explores a different primary theme, in order to represent the spectrum of relevant and important issues currently seen in the WSN field. Most of these applications were built using commercially available platforms such as the 8-bit Mica2 mote, the 16-bit TMote sky, and the 32-bit Gumstix platform, although all of these applications required some degree of custom hardware. Multi-tiered network designs, in which 8- and 32-bit platforms are combined into a single system,

Table 1.1 Summary of case studies

Chapter	Application	Primary theme	Year(s)	WSN platform	Number of nodes	Deployments
4	Volcano monitoring	Addressing data quality with iterative deployment	2004–2007	Mica2, TMote	3–16	3
5	Marmot localization	Timeliness with high data rate sensing	2007–2009	Custom 32-bit	8	1
6	Soil monitoring	Dealing with sensor faults and missing data	2005–2008	Mica2	11	1
7	Toad monitoring	Low power acoustic detection and classification	2003–2008	Stargate, Mica2	5	3
8	Intruder monitoring	Large scale deployment	2003–2005	Stargate, Mica2	1,200	1
9	Glacier monitoring	Full system engineering	2002–2008	Custom 8 + 32-bit	20	5
10	Patient monitoring	Mobility and sensing	2006–2007	Cricket Mote, ipaq	14	2

are described in two case studies: toad monitoring (Chap. 7) and intruder monitoring (Chap. 8). A multi-tiered design allows more processing capability on some nodes while keeping the overall cost of the network within budget.

Some insight into the maturity and growth of each project can be inferred from the number of years the project was active, number of deployments and number of nodes used in deployments. Chapter 8 describes the largest network by at least two orders of magnitude. Although its development was geared towards one full deployment, the network had to be iteratively scaled during testing and validation. In contrast, Chaps. 4 and 9 describe systems with an iterative increase in network size, where the networks grew with each successful deployment.

Table 1.2 presents a cross-chapter breakdown of constraints/requirements imposed by the motivating application. The table's columns are as follows: *lifetime* refers to the need for nodes to manage power usage in order to maximize lifetime, *node cost* refers to the need for the monetary cost of nodes to be low, *N/W scale* refers to the need for the network to be large (or scalable), *form factor* refers to the need for the sensor nodes to be small and light, *online* refers to the need for the data and information produced by the WSN to be available in an on-line or real-time manner so that actuation (generated by the user or some automated procedure) is enabled, *mobile* refers to the need for the sensor nodes to be mobile during their operation and *user interaction* refers to the need for the network to respond to user interaction as part of its normal operation.

Table 1.2 Table of shared constraints/requirements

Chapter/ application	Life-time	Node cost	N/W scale	Form factor	Resource limits	Online	Mobile	User interaction
4: Volcano monitoring	X	X			X			X
5: Marmot localization						X		X
6: Soil monitoring	X							
7: Toad monitoring	X	X			X	X		
8: Intruder monitoring		X	X			X		
9: Glacier monitoring	X	X		X				
10: Patient monitoring		X		X		X	X	X

It is clear from the table that *node cost* is a constraining factor in many of the case studies presented in this book, showing that, in general, real-life deployments are greatly affected by the amount of funds available on a given project. The best example here is found in Chap. 8, where 1,200 nodes were present in the network – in this case, great care had to be taken to choose the correct platform that fulfilled the requirements. The systems in Chaps. 4, 6, 7 and 9 are constrained by node/network *lifetime* as they are long-term deployments. In the case of glacier monitoring, for example, sensor nodes are left within the glacier for several months at a time, many thousands of miles away from the research laboratory. Long term monitoring applications also mean that the deployments are generally unattended, complicating system control and maintenance.

In Chap. 10's application, sensor nodes are attached to patients, and must meet rigorous requirements for health and safety as well as maintaining a level of comfort in order for the user to agree to wear them. Moreover, the system needs to function within a rigorously regulated environment. Thus, *form factor* and *user-interaction* are important constraints.

The marmot localization system in Chap. 5 appears to be less constrained than many of the other case studies; this is because the system is not required to be deployed for long periods, is attended during deployment, and has the benefit of a powerful processor and increased resources compared to standard mote platforms. However, along with the systems in Chaps. 7 and 8, there is a requirement for *online operation*, where the system information is used to make decisions in a timely manner – for Chap. 5 the decisions are to be based on acoustic event detection and localization of animals, for Chap. 7 the theme is event detection and classification of animals, and for Chap. 8 it is event detection, classification and localization of vehicles and humans in a military scenario.

Table 1.3 Table of challenges found in the case studies

Chapter/ app.	Capture phenom. correctly	In-network process decision	Data rate < B/W	Evaluate/ integrate improve-ments	Faulty/ missing data	Mainte-nance	Manage network scaling	Time sync.
4: Volcano monitoring		X	X	X	X		X	X
5: Marmot localization	X	X	X	X				X
6: Soil monitoring	X				X	X		
7: Toad monitoring		X	X	X				
8: Intruder monitoring	X			X		X	X	
9: Glacier monitoring				X		X	X	X

Whilst *resource limits* (referring to the node capabilities, for example communication bandwidth available to nodes and the local processing capability) is not a requirement as such, like other headings in this table, the column is meant to reflect how the hardware chosen for the project has influenced other design choices. As we will show in Table 1.3, the hardware choice has considerable implications for meeting most of the generic application challenges.

Although the case studies in Part II present deployed systems that were developed for different motivating applications, with different specifications and goals, several generic challenges are apparent in all chapters. To this end, Table 1.3 presents the main challenges. The solutions and lessons described in individual chapters are generically applicable to most monitoring applications classes. Since Chap. 10 focuses on mobility-based sensing in body sensor networks, the challenges faced are different than the relatively static monitoring applications presented in Chaps. 4–9; for this reason Chap. 10's challenges are omitted from the table. In the table, *Capture phenomena correctly* refers to the challenge of determining the required rate at which to sample the phenomena and the inter-node spacing necessary to ensure adequate coverage. Chapter 8 deals with this problem in the context of intruder detection, classification and localization by establishing a specific network deployment density based on the required accuracy for the system, and placing nodes in a regular grid over the area of interest. Chapter 6 looks at increasing the spatial density of temperature and soil moisture measurements to identify small-scale environmental variations.

Maintenance refers to the challenges presented by the maintenance of a long-term sensor network deployment. For each chapter dealing with *Maintenance*, the issue has a different significance for the motivating application, leading to different system design decisions. The soil monitoring case study in Chap. 6 finds that post-deployment sensor maintenance is actually harmful to the deployment, as it disrupts the monitored phenomena; therefore maintenance is deferred until absolutely necessary, and missing or faulty data is replaced by sophisticated prediction for as long

as possible. In Chap. 9, the deployment site on a glacier is so remote and seasonally affected that maintenance can only be carried out during specially timed visits.

In-network processing decision refers to the decision made in some of the case-studies to perform some of the end-to-end system processing directly on the sensor node. Chapter 5 shows that the need for dynamic in-network processing was determined only after the full deployment took place. In Chap. 4, the decision to process some data locally, at the node, in order to reduce the amount of data required to be sent (and thus increase network lifetime), was taken after a thorough analysis of deployment data. By assigning "value" to the data gathered by nodes, only the important data was transmitted in further prototypes. Chapter 7 comes to a similar conclusion in deciding that, because only toad calling events are of interest to the system's main goals, these events must be detected, filtered and compressed to minimize the network usage and enable lower-power operation. The generic lesson learned from each of these examples is that dynamic in-network processing should be seen as a tool to help address a particular problem, rather than a design principle for the system – avoid processing in-network unless there is a clear need. As can be seen from the table, the need for in-network processing correlates well with the *Data rate* > *Bandwidth* challenge, where the aggregate volume of data generated by the sensor nodes is greater than the bandwidth of the sensor network.

Manage network scaling refers to the challenges encountered when attempting to increase the scale of an existing network or build a large network from scratch. In Chap. 8, a network of greater than a thousand nodes is created. To achieve the deployment of a network of such magnitude, the testing and validation process iteratively increased the network size, going from simulation to testbed to full deployment. In Chap. 4, increasing the number of nodes in the network also increases the amount of data generated about the volcanic events being monitored. Since the bandwidth cannot support the transmission of all data (another example of the *Data rate* > *Bandwidth* challenge) it is of utmost importance to decide which data are most valuable, given the expense to the node of sending the data.

Evaluate/integrate improvements refers to the challenges that are raised when problems during deployment are identified and subsequent improvements have been proposed. Both Chaps. 4 and 5 present examples of deployment-related problems that caused performance issues or system failures. Both chapters present solutions to their respective problems that are either evaluated in simulation, laboratory-testing or in a new deployment. In order to enable post-deployment analysis of problems, both chapters show that it is necessary to keep detailed documentation and acquire log information, and diagnostic data alongside the actual sensor data gathered. Gathering as much diagnostic data as possible during deployment can help identify problems and aid various investigations which were not considered in the field. Diagnostic data will also enable the designer to quantify the improvements made in all steps of testing, from simulation to emulation to deployment. Deployment raises the hardest challenge, as conditions are likely to change, meaning the researcher cannot isolate all the parameters desired (as might be possible with simulation, for example).

Faulty/missing data refers to the challenge of dealing with both data loss and node failure, resulting in missing data. As previously mentioned, Chap. 6 consolidates this problem with choosing the correct time to perform maintenance on failed sensors, and in doing so must utilize suitable techniques for predicting data.

Time synchronization is a ubiquitous challenge in WSN systems. Since sensor nodes are not connected to a central data acquisition unit, and must communicate wirelessly, the data they sense must be timestamped. Without some form of time synchronization, it becomes impossible to correlate measurements spatially. The accuracy that data must be synchronized to across a WSN depends on the application's processing requirements and the rate at which data is being gathered; Chap. 4 has a high rate requirement on data sampling (100 Hz), and uses a GPS unit in concert with a time synchronization service to synchronize the network. However, bugs in the time synchronization service cause irregularities in the data, requiring off-line repairs of the data to correctly align it. In Chap. 9, time synchronization across several sensor nodes deployed in glaciers is required to be accurate within a second, and is provided by broadcasting the time from a single GPS unit. Chapter 5 requires high accuracy time synchronization to correlate acoustic measurements taken at 48 kHz across nodes separated by tens of meters; in this case, the network is synchronized purely by local, ad-hoc communication.

Setting itself aside from the generic challenges above, the application described in Chap. 10 brings forth a new set of constraints to be mitigated in the design and deployment of a class of WSN systems. Deployment in a heavily regulated environment, compounded by a large number of stakeholders and issues of mobility and wearablity form a core theme in this patients monitoring application. These constraints limited the choices available to designers, both in technical terms (such as for example imposing the use of certified, off the shelf subsystems wherever possible) and regarding the design-deployment-evaluation cycle. Deployment logistics and the potential impact of the WSN system on existing work practice limited the scope of the application. The lack of opportunity for iterative deployment greatly impacted the success of the project in absolute terms. Issues such as these are to be encountered by most applications where the human factor is central to the application scope.

While the above challenges are clearly a focus for the respective chapters, overall, this book offers a wide coverage of most issues encountered in the process of design, implementation and evaluation of deployable WSN systems. It is hoped that, upon using the book, the life cycle of new applications development will be considerably shortened and many of the hurdles encountered by the authors and contributors here will be successfully overcome by the reader.

The editors hope that the book will provide the reader with a comprehensive learning experience and a pleasant journey into one of the most exciting engineering domains.

References

1. Bulusu N, Jha S (2005) Wireless Sensor Networks: A Systems Perspective. Artech House, Boston
2. Holger K, Willig A (2005) Protocols and Architectures for Wireless Sensor Networks. Wiley, New York
3. Langendoen K, Baggio A, Visser O (2006) Murphy Loves Potatoes: Experiences from a Pilot Sensor Network Deployment in Precision Agriculture. In: IEEE International Parallel and Distributed Processing Symposium (IPDPS '06)
4. Pottie GJ, Kaiser WJ (2005) Principles of Embedded Networked Systems Design. Cambridge University Press, Cambridge
5. Zhao F, Guibas L (2004) Wireless Sensor Networks: An Information Processing Approach. Morgan Kaufmann, MA

Chapter 2
Learning from Deployment Experience

Elena Gaura, Michael Allen, Lewis Girod, Geoffrey Challen, and James Brusey

Abstract Early developments in WSNs focused on minimizing the physical size and energy footprint of the nodes, and on exploring the opportunities and problems introduced by very large networks of low-cost nodes. In practice, however, deploying or testing systems at this scale has not been practical. At smaller scales, extreme resource constraints are often artificial. The evidence from surveying deployed applications suggests that in general, concerns about size, power, and large network scales are trumped by practical considerations relating to deployment such as packaging and logistics, and relating to what is commercially available for a reasonable price. In this chapter we introduce the characteristics of deployed WSN applications and discuss how they differ from lab-based experiments. We then distill some general design strategies from past deployment experiences, with particular reference to applications featured in Part II of the book. Next, we present a survey of platforms that are currently commercially available to use as starting points for WSN applications, and finally discuss the past and current market for WSN applications and products.

2.1 Illustrating the Problem: Three Deployments

Wireless embedded networks are difficult to deploy for a wide-ranging set of reasons. The complexity of the environment in which the system must operate leads to many problems that catch the designers by surprise. To better illustrate the nature of these challenges, in this section we present three brief case studies. Each of these case studies gives a succinct overview of the project strategy and then describes some of the unanticipated problems encountered by the developers during deployment. The remainder of the chapter is structured as follows:

Section 2.2 describes the evolution of the WSN domain from the motivating vision of Smart Dust. We discuss views historically adopted by researchers across

E. Gaura (✉)
Cogent Computing Applied Research Centre, Coventry University, Coventry, UK
e-mail: e.gaura@coventry.ac.uk

E. Gaura et al. (eds.), *Wireless Sensor Networks: Deployments and Design Frameworks*,
DOI 10.1007/978-1-4419-5834-1_2,

the WSN applications design space, and argue to support an *application-centric* approach to WSN design, with forward references to the case studies in Part II. Further, we analyze the gap between the theoretical and practical advances to date and thus highlight challenges in applying theoretical developments to real-life implementations.

Section 2.3 offers an overview of off-the-shelf WSN platforms and comments on the suitability of the state of the art to support both the researcher and the commercial markets.

Section 2.4 discusses how the prospects for WSN technology adoption have been improved by demonstrable, real-life application successes and points out the perceived barriers to wide adoption of the technology.

Section 2.5 provides a brief set of strategic recommendations to WSN designers.

2.1.1 Bangladesh Groundwater Monitoring, 2006[1]

In January 2006, we deployed a wireless sensing system in a rice paddy in Bangladesh to help scientists evaluate the relationship between irrigation and arsenic contamination in the groundwater. Tens of millions of people in the Ganges Delta drink well water impacted by arsenic, a massive environmental poisoning projected to result in two million cancer cases each year. The experiment was designed and deployed with scientists and civil engineers studying this problem at MIT and the Bangladesh University of Engineering and Technology. We deployed 50 sensors connected over a low-power wireless network to monitor a variety of soil chemistry and hydrological parameters in nine different locations. The base-station collected 26,000 measurements over a period of 12 days. A snapshot from the deployment process is shown in Fig. 2.1. The results gathered from this and other deployments are described in [58] and [59].

Our deployment, like others, was plagued with a number of unexpected events. On the first day in the field, the landowner informed us that our base-station would likely be stolen if we left it in the field over night. Our delay-tolerant networking layer was immediately put to work, and we received 91% of the data despite the base-station's absence for over half of the deployment duration. Without a networking layer that cached data locally until it was successfully received at the base-station, our system would have missed the diurnal activity, which took place in the early hours of the morning. We successfully addressed the networking issues with robust system design.

We were less prepared to address the sensor faults due to the unknown nature of the deployment environment. Our main problem was simply to identify faulty data in the field. Unfortunately, distinguishing between data that is faulty or simply unexpected is a challenge even in contexts where the environment is carefully characterized in advance. It was more difficult in Bangladesh, where our ambitiously

[1] Nithya Ramanathan, CENS/UCLA and Lorax Analytics.

Fig. 2.1 Photos from the deployment site in Bangladesh. The photo on the *left* shows a sensor node with the case open. The photo on the *right* shows part of the deployment process. The large metal cabinet houses the base-station and data collection server. Original images used with permission of Charlie Harvey, MIT Civil and Environmental Engineering

large network was deployed precisely because there was little a priori knowledge available about the chemistry of the rice paddy. For example, in one instance, a nitrate sensor reporting out-of-range values was miscalibrated, in another instance it was reporting an accurate, but unexpectedly low, concentration. Without additional information, it was impossible to determine at the time whether we should replace the sensor and throw out the data, or mark it as the most interesting sensor and data in the deployment. Timely interventions were necessary because the deployment was short-lived. But further analysis was required in many instances before we could determine the appropriate course of action.

Scientists handled the uncertainty by taking concrete actions in the field. Simply by being at the site when data was collected, the scientists were able to collect and incorporate contextual information when detecting and diagnosing faults that the system could not sense. For example, around day 7 of the deployment, moisture and chemical concentrations unexpectedly flattened out. Scientists in the field noted that the farmers had irrigated the field that day, and immediately made the connection between the two events. When simple observations could not explain unusual sensor readings, as occurred in our nitrate sensor example described above, a scientist extracted a water sample for later lab analysis. The scientist inserted a thin tube into the soil and syringed out several drops at a time. The sample was preserved to limit atmospheric exposure, and analyzed in a lab. This process is time- and labor-intensive, and was therefore avoided when possible. However, results from the lab analysis can verify anomalous sensor readings, or confirm suspicions of faulty hardware.

Such painstaking human actions and observations were necessary to interpret much of the data we collected in Bangladesh. Even the process to determine when further verification was required was done manually because the chemistry of the rice paddy was relatively unknown. It was not possible to use fully automated fault detection and remediation tools because in many instances even the scientists could

not conclusively determine when hardware was faulty without further analysis. However, real-time feedback on the health of the entire system would have aided us in focusing our limited resources on data problems that did require in-field validation and action. As our deployments have matured, we have aimed our efforts at designing this human-machine system as a whole to maximize the information returned from the system with limited burden on the user. We have designed a system that uses an automated model designed in advance, and incorporates feedback from the user at run-time in order to adapt to new environments gracefully. The system creates an automated model using a small set of features that groups similar sensor data in a low-dimensional space. This reduces fault detection and diagnosis to simple anomaly detection techniques, which are easily modifiable in real time. Our work has demonstrated that, if features are chosen well, this partially automated approach will scale to a large number of points, and adapt to new environments with little feedback from the user.

2.1.2 Peru Seismic Station Deployment, 2007[2]

The PERUSE Peru Subduction Experiment is an ongoing 5-year project to study the tectonic structure and behavior of the Nazca oceanic plate that is being subducted under the South American continental plate. The tectonic behavior in this region is unusual, and the data collected in the area prior to this project has not been sufficient to explain the observed phenomena. To address this, PERUSE will deploy a total of three linear transects of seismic sensors, to form three sides of a open rectangle facing the coastline.

Each transect will be 300–400 km long and will traverse areas that are mountainous and sparsely populated. The first two lines are currently populated with 50 broadband seismic sensors each placed at 6 km intervals; the third line will be constructed with 25 stations taken from each of the first two lines, so that for the final two years of the project, two lines of 25 stations will be operational concurrently with a third line of 50 stations.

These scales present problems for traditional broadband seismic stations. Remote seismic stations are typically deployed as standalone units with data storage for about 1 month of operation. This necessitates frequent visits by researchers to inspect operations and collect data. This is very time-consuming, and has high "mean time to notice failure", "mean time to repair", and "mean time to verify" cycles.

In collaboration with geophysicists at Caltech and UCLA, researchers at CENS have deployed and operated a large-scale seismic wireless research network across a 300 km transect crossing the Andes in southern Peru. We use an ad hoc WiFi (IEEE 802.11b) network to transport data across multiple hops from each of 50 sensor stations positioned roughly at 6 km intervals. Due to terrain and security concerns,

[2] Richard Guy, Technical Project Manager, CENS/UCLA.

Fig. 2.2 Peru seismic station. From *left* to *right*, node hardware internals showing the batteries, digitizer, and node hardware; overall view of the node installation site, including the seismic sensor itself inside the PVC tube; WiFi antenna installation. Original images used with permission of Ing. Victor Aguilar Puruhuaya

the interval between nodes varies somewhat and in many cases we use rather longer links – up to 50 km in some cases – to connect nodes that would otherwise be cut off from the network. Ultimately, the data arrives at an Internet-connected radio relay site where it is transported to UCLA via a conventional consumer DSL connection [52].

This work follows from a previous deployment in Mexico [38], and leverages much of the same hardware and software. The Mexico deployment provided opportunities to gain large amounts of experience with remote deployments, solar charging, and long range WiFi. Figure 2.2 shows some of the deployed hardware and the deployment environment. During the early parts of the Mexico deployment, the nodes required frequent servicing by research personnel while the bugs were worked out of the hardware and software. Several novel components were developed to address the needs of this system, including a Disruption Tolerant Shell [50] and a method for recovering timing information post-facto [51].

The Disruption-Tolerant Shell (DTSH) provides a way to issue broadcast and unicast maintenance commands to nodes many hops away from an Internet connection. Although in principle all of the nodes in the network are continually on-line, in practice intermittent network problems were commonplace. Network problems had a variety of causes, including antenna mis-alignment, bad weather, and upstream node failure. DTSH used a store-and-forward protocol to push requests out to the network and retrieve results [50].

A second development was motivated by intermittent problems with the GPS timing on the digitizers. Under some conditions, the device would record data that had large offsets in timing, caused by availability problems with GPS. Without tight time synchronization, this data was unusable for the application. To address this problem, an algorithm has been developed to use features of the data itself to correct the timing after the data is collected [51].

Since the Peru deployment was based on what had been learned and developed in the Mexico deployment, we expected that it would go smoothly. While this was generally true, some unexpected, difficult to debug problems still cropped up. One of these involved the use of a shared communications tower.

When possible, we used existing towers to mount our antennas, to avoid the cost of building our own much shorter and weaker masts. These towers typically were

in use for various types of telephony (e.g., microwave relays and cellular service), voice radio, or data relaying services, and we tried to learn in advance what frequencies were already in use so that we could both avoid interfering with existing services and hopefully avoid interference with our own services operating in the unlicensed 2.4 GHz band. This is relatively easy when a tower is alone on a hilltop and we have had the necessary preliminary conversations, but is more difficult when a number of towers from different enterprises are clustered in close proximity on a hilltop.

We recently began using a 70 m tower that is part of a cluster of five towers (all 50–100 m tall) spread across a 300 m diameter hilltop along a ridge that separates the large prosperous city of Arequipa from a poor rural region to the southwest. In casual ground-level (2 m elevation) testing, we knew that the site had spotty WiFi behavior in Channel 11, but based on our past experience operating this equipment in Peru and in Mexico, we were confident that we could overcome the problems with sufficient antenna height and perhaps a bit of RF amplification. We normally use 200 mW radios for up to 10 km links, but this can be augmented with a 1 W (or even 2 W) amplifier in extreme circumstances.

However, after installation, we noticed intermittent problems between this relay hub and its three client sites, which lay at different distances away, from 7 to 30 km. Sometimes we had very solid connections and multi-megabit throughput from the farthest node, but kilobit throughput from the nearest node. We could only get the throughput we expected for a few seconds per minute. The net result was connections that were unable to keep up with the aggregate 4 kB/s transfer rates we needed in order to upload the seismic data from upstream sensors.

After a frustrating day of fruitless troubleshooting on site, the team was scheduled to return to UCLA. Soon after their return, the relay site went offline completely. Our Peruvian geophysicist colleague traveled to the site to see if anything was visibly amiss. There he found that thieves had breached the coils of razor wire "protecting" the site, and had stolen the solar panel used to recharge the battery providing energy to our radios, along with two panels from a neighboring mast.

Serendipitously, the engineer who maintained the systems on the other tower was on site the same day to install a wind-powered generator atop his 30 m mast, and in the course of friendly conversation we learned that his communications system used a non-WiFi radio with a transmit frequency centered in the WiFi Channel 11 band, with target receivers located in the same line of sight as two of ours, and within a kilometer of our receiver locations. Since his system was not WiFi based, it did not appear in our channel surveys.

This serendipitous conversation avoided a complex debugging task. While surveys with 802.11 receivers would not reveal this type of interference, it could have been discovered using a portable spectrum analyzer. However, even the spectrum analyzer would only help when the interferer is operating. In this example, the site survey was complicated by directional antennas and by the intermittent nature of the interference.

2.1.3 WaterWise: Monitoring an Urban Water Distribution System (2008)[3]

WaterWise is a joint project between the MIT-Singapore Research and Technology Center and the Intellisys center at Nanyang Technical University (NTU) to develop, deploy, and test a system for monitoring the urban water distribution system in Singapore. The goal of the project is to develop a system that can provide more accurate hydrological modeling of the pipeline network, as well as detect, identify and locate pipe bursts in the system. This is an important capability because bursts in water pipes are often not discovered until many days after the event, during which time they can cause damage to other infrastructure.

This project is a continuation of work done at MIT on a similar system called PipeNet, that was deployed in Boston in 2006 for about 1 year. The initial proposal for WaterWise would deploy the PipeNet system in Singapore, on a larger scale. However, this proved to be impractical because, as is often the case with prototype implementations, many of the components used in the original system were either obsolete or not commercially available. Further, the algorithms that would be required to process the data and detect events were not established with certainty, because thus far the data sets collected from fielded nodes were relatively small.

Given that new hardware would need to be developed, and based on prior experience with the requirements of a new deployment, we proposed a conservative three-phase deployment plan in which the deployment could be scaled up over time. This process of scaling up served three purposes: (1) it made the early deployments more manageable, when the entire system was under development and in flux; (2) it meant that the cost of the initial version could be larger and less energy efficient, enabling it to be rapidly integrated from off-the-shelf modules without requiring a large investment in customization and optimization; and (3) it permitted the more "brute force" data collection strategy required to produce a data set to drive algorithm development and more efficient sampling regimes. Table 2.1 describes the three phase plan we proposed.

From the description of Phase 1, it might seem overly conservative. With only 5 nodes, few power constraints, and no complex networking strategy, where is the difficulty? However, despite the conservative nature of this strategy, we found that it was in fact achievable but difficult, and we also found that we needed to tune the plans along the way. Among the issues we encountered are the following:

- Although the first phase was slated for 5 nodes, we manufactured 10 boards so that we would have extras. However, even this oversupply has been insufficient because after seeing the data, our customer was anxious to deploy additional Phase 1 nodes to collect data at other locations. As a result we may fabricate a "Version 1.5" that addresses some bugs in in Version 1 but is otherwise substantially the same.

[3] Lewis Girod, MIT/CSAIL.

Table 2.1 Milestones, time-line and objectives of the three phase deployment plan for WaterWise. The schedule has slipped relative to the original proposal: by end 2009 only 8 nodes were deployed

Phase	Size	Objective
1 (2008)	5	In this phase, we will fabricate new prototype hardware and address many packaging, mechanical, and logistical issues. The system will support full time, full rate data collection, recorded to server via 3G; this data set will support algorithm development and validation. Controlled experiments can also be performed with full data collection. Power requirements are unconstrained, with the assumption that the node has batteries to last 1 week that are recharged by mains or solar.
2 (2009)	25	In this phase, we will refine the hardware and experiment with different platforms in a transition to a lower cost, lower power system. These nodes will be deployed in addition to the original set.
3 (2010)	100	In this phase we will validate the new design from phase 2 with a deployment at scale.

- In our original vision for Phase 1, the nodes were powered from light poles, with solar charging as a last resort. However, because getting power from the light poles turned out to be unexpectedly difficult to arrange, all of the Phase 1 nodes needed solar charging, using panels mounted only 10 feet off the ground. Getting the solar charging system to work required much iteration with panel size and location, negotiations with city authorities to trim foliage, and manual battery replacement. Two years into the project, we are now beginning to get access to light pole power.
- We spent a considerable amount of effort designing and fabricating watertight node packages so that the nodes could be located inside the manhole. However, power issues, hardware bugs, battery replacement, and ongoing software development have all required frequent access to the node hardware. We had expected this initially, and had intended the first nodes to be located above ground in a junction box, perhaps later moving underground. However they have remained above ground longer than expected and only now, in the second year, is the first node going underground. Figure 2.3 shows a node installation and hardware.
- Our original plans for 100 nodes were based on our best guess as to what was required. However, limitations in the number of tapping access points make larger scale deployments difficult. In addition, as we learn more about the application, it may be the case that a smaller number of carefully selected measurement points will work as well or better than a larger network.

We are presently (January 2010) entering Phase 2 of the project, having slipped about 1 year from the original time-line. Thus far the project has been quite successful, and even with the deviations from the original plan it is approximately on schedule. We are currently rolling out an intermediate version node to replenish our inventory and possibly to replace existing nodes where that is desirable, and we are beginning work on a new, lower cost, lower power version in parallel with ongoing work on algorithm development.

Fig. 2.3 The WaterWise System. From *left* to *right*: the node installation with solar panel; a view of the inside of the junction box, showing the batteries, solar charge controller, and the node itself in its watertight packaging; and a tapping point assembly installed inside a manhole, with a electromagnetic flow meter installed during calibration tests. Original images used with permission of Amitsur Preis, SMART Center

2.1.4 Discussion

These three examples provide snapshot views into deployment projects. In these examples we can see that application specifics are a driving factor in the eventual design, and that every deployment, no matter how well planned, encounters the unexpected. The complexity of the real-world environment in which these systems must operate is often a source for these complications, whether from thieves, weather, or bureaucrats.

Another common thread in these examples is that, while the eventual ideal design will require a clear understanding of the problem and may require optimization across many layers, this is not practical for prototyping. Because many WSN applications are breaking new ground, a prototyping phase cannot be avoided. This is in stark contrast with more mature areas such as IT or web services, where comprehensive application solutions exist commercially and there are many large consulting organizations with extensive experience solving applications that are likely to be very similar to yours.

These examples also demonstrate the importance of a broad blend of skills on the team. Including experts in the application domain is critical to ensure that the data produced by the system supports the application. Within the system design team, it is important to include those with both hardware and software expertise. If the design process focuses excessively on only part of the possible solution space (e.g. more hardware or software focused), this will often lead to systems that are more difficult to implement and support. In many cases a small hardware fix could also be implemented purely in software, but the software solution is much more complicated, or fundamentally less reliable; and the converse is often true as well. Teams with agility in both realms will be much more effective and the resulting projects will be much more practical.

2.2 WSN Design Strategies

The Smart Dust project that sparked the initial wave of WSN research field left researchers with a legacy of "must have" characteristics, namely that wireless nodes be small, low-power and disposable, and that networks of nodes be massively scalable, intelligent and self-configuring. In this section, we present the journey from the original Smart Dust vision to the WSNs being prototyped and deployed today, based on a number of examples from research and commercial implementations. We analyze typical constraints imposed on WSN systems and present a perspective on the utility of both practical and theoretical design.

2.2.1 From Smart Dust to Today's WSNs

WSNs began as a technology with the Smart Dust vision [40, 69], which promised to: (1) allow unprecedented detailed monitoring of phenomena, (2) capture their essence by revealing fine grained, complex dynamics and, (3) translate the findings into information-heavy field-phenomena or global representations. Such WSNs, coupled with models of the physical phenomena under investigation, would enable knowledge generation, provide the ability to design and integrate knowledge-based decision and actuation systems, and generate valuable and trustworthy predictions.

Thus, WSNs have been seen as the cornerstone for the fully integrated "sense-model-predict-act" systems of tomorrow in a variety of engineering and social contexts. However, after a decade of intensive research there are still a large number of challenges to be overcome before this grand vision becomes reality. Currently available commercial and research products do not yet achieve the Smart Dust vision of being invisible and maintenance free embedded systems that are easy to develop, deploy, and use.

In the Smart Dust vision, WSNs were envisioned to overcome application cost primarily through large scale, low effort deployment of low cost nodes. The large deployment scales and redundant measurements were intended to compensate for several inherent disadvantages stemming from the use of extremely low cost nodes; the economies of scale were presumed to drive a low unit cost and hence ensure fast industrial up-take of this disruptive technology. Furthermore, to contain node cost, the sensors would be low quality and poorly calibrated; the nodes themselves would be extremely resource-limited in order to reduce the power costs; the nodes would be one-use disposable devices, with little care taken in the physical deployment strategies (e.g. scatter them from the air). Given this proposal, early research challenges focused primarily on scalability and on software design for extremely resource constrained nodes. Research into scalable networks would leverage over-deployment for fault tolerance, self-healing network features, and efficient multi-hop protocols. Research that focused on resource-constrained nodes investigated small-footprint operating systems, low power communication, and distributed and collaborative processing with a goal of achieving years of functional life on a small battery.

While hundreds of research questions on the themes above were posed and answered in the first years of the new millennium, the majority of ideas and concepts contributed to the field could only be explored theoretically or validated through simulation. A minority of works have been tested experimentally and a small subset of these have been demonstrated in-situ, limiting the industrial impact of WSN systems.

Nonetheless, the contribution of in-situ demonstrations to the field has been invaluable. Most importantly, tackling real-life applications in real-life deployment environments has helped identify pressing research challenges not found in the Smart Dust agenda. In resolving a well-motivated real-life application, the developer of a prototype system can usually only afford a small number of relatively expensive sensor nodes. This leads to a range of considerations that run counter to the premises of the Smart Dust vision and which are discussed in more detail in later chapters:

- In order to satisfy the requirements of the application with a limited number of nodes, the developer must be concerned with the fidelity of the data that the nodes are sensing, and must develop a means of fulfilling the application even when this data is simply not continuously available (see Chap. 6).
- Many applications involve acquisition and communication of data at high rates, well beyond the capabilities and power available to most highly miniaturized devices, dictating the development of nodes with a large processing, power and physical footprint (see Chaps. 5 and 8).
- Keeping the system cost low implies sophisticated design and careful architecting of the WSN systems (see Chap. 7).
- Harsh deployment environments can hinder even small scale, carefully planned deployments (see Chap. 9).
- Working with the end user and complying with procedures and regulations in sensitive environments severely limits the WSN technological choices (see Chap. 10).
- Acquiring and delivering enough accurate data to gain insight into the researched phenomena is often surprisingly difficult to achieve, even when the system design is based closely on previously developed hardware and software which purports to solve the problem (see Chaps. 4, 6 and 9).

Many successful application solutions have indeed been developed in spite of (and often by explicitly ignoring) the tenets of the Smart Dust vision: sensor nodes do not need to be as small as possible, nor excessively resource-constrained; useful problems can be solved with small scale networks; more expensive, well-calibrated sensors are often the best choice; random deployment is neither generally necessary nor desirable.

An extreme, rather holistic example of how *one view does not fit all* comes from the *Plug* system developed by Lifton et al. [49]. Plug is intended for inhabited environments (such as the home and the workplace), is powered through a mains outlet (and thus has no energy-related concerns) and has a comparatively large size (20 cm × 7 cm × 12 cm) and weight (1 kg). However, despite its physical

Fig. 2.4 The VoxNet System. From *left* to *right*: the node hardware with the case open, showing the sensor interface board (the CPU modules are under the battery); one of the nodes, deployed in Colorado, USA in 2007; a VoxNet node locating a marmot. Original images used with permission of Lewis Girod, MIT/CSAIL

footprint, it is as unobtrusive as any other power bar, and supports multi-modal sensing, actuation capabilities and wireless communication. Its utility as a sensor network platform lies with the tight integration of the observed and the observer: the primary role of the Plug is to measure how the observer interacts with the environment, essentially setting itself in sharp contrast with the common view of sensor nodes hidden throughout the environment and not interacted with (the *deploy-and-forget* Smart Dust concept).

Another example comes from VoxNet [2] (presented in Chap. 5), a WSN system used in bio-acoustic sensing that requires sensor sampling rates on the order of several kHz and is exploratory in nature (i.e., the application requirements change as phenomena exploration experience is gathered). This type of application can be prototyped using powerful embedded computers, eliminating the need to optimize the system in the early stages of development (see Fig. 2.4). Although the node's power consumption is high and the battery lifetime is on the order of hours, this was not an immediate problem because the initial deployments could accommodate daily recharge. The development effort that would have gone to optimization was instead invested in making VoxNet a turn-key system that was easy for field biologists to use and adapt.

Experiences such as those above have contributed to the conclusion that the most useful research in WSNs is driven by finding successful solutions to problems posed from *outside* the WSNs research community and that the motivation for the WSN development needs to be driven by *realistic business models.* Only when the work is motivated by application specialists and beneficiaries of the technology, will WSN research move from a set of technological solutions in search of problems to become a widely-adopted, thriving technology. WSN application developers must formally define requirements, specify, design, implement and deploy while considering realistic field conditions and stakeholders (i.e., clients and/or application motivators, be they researchers in the dedicated application domain or commercial end-users). WSNs developed to provide research instruments that enable new insight into a variety of previously unquantified or unobserved phenomena should also conform with

the above. There is a long and costly development process from proof of concept through prototypes and pilot deployments, to finally providing the application researchers with a robust, repeatable, commercially available instrument which could become the norm in their branch of study.

In summary, practical WSN researchers have learned what are the essential ingredients to developing fit for purpose WSN systems: (1) the application dictates the choice of development methods, techniques and tools; (2) advances beyond the state of the art should be sought selectively in the context of any given application (i.e., not every application requires a minimal size and energy footprint and sophisticated in-network collaborative behavior); (3) self-posed questions and solutions bring marginal benefit to the field and no benefit to WSN adoption efforts; (4) letting the application and the application specialists drive and inform, respectively the WSN development is essential; (5) the specialists need to have strong business cases for WSN use, otherwise the work will remain at the stage of research feasibility studies.

2.2.2 Design Spaces and Design Views

The design space of WSN systems is complex, and often requires making trade offs that affect the design of both hardware and software components, while concurrently meeting the application requirements. This is further complicated by the shortage of pre-existing platforms and standards resulting from a limited commercial market. Römer [66], presented twelve qualitative metrics to describe the design space of existing, deployed WSN applications. The metrics are useful to help identify broad differences or commonalities between present and future WSN applications or application classes. The design space metrics can be categorized as follows:

- Node properties: mobility, size, cost, power source, communication modality
- Network properties: heterogeneity, deployment process, communication topology, coverage, connectivity, expected lifetime

The design process ranks these properties based on the application requirements, on the constraints imposed by hardware and software platform choices, and on the environment(s) in which the system must be deployed.

The authors here propose that there are three views on this design space that can drive the development process for WSN systems (in terms of both hardware and software). These are: the *application-centric* view, the *network-centric* view and the *device-centric* view.

The *application-centric* view maintains that the application's requirements dictate the software and hardware functionality that should be developed. Therefore, network and middleware protocols should be designed and implemented as needed. Several systems have successfully adopted this approach, for example the iterative development and deployment of a volcano monitoring project [70, 72] and the development of a WSN to monitor glaciers [55, 56, 60]. Both of these systems are discussed in detail in this book.

The strength of taking the application-centric view is that given a wide enough set of specific applications and deployment experiences, the services and protocols that are generically useful, reusable and important will naturally emerge. Those components will then become encapsulated as off-the-shelf software components and tools for WSNs. An example drawn from the literature is the Flooding Time Synchronization Protocol (FTSP) which was originally developed as part of a sniper localization system [47]. This approach has gone on to be the de facto time synchronization mechanism in mote-based WSN research, and is integrated into TinyOS 2.1 (the most recent version as of writing) by default.

The *network-centric* view focuses on directly designing generic components for building sensor networks as a first principle, so that arbitrary applications at arbitrary scales can be accommodated. Examples include MAC protocols [63, 77], multi-hop routing protocols [37, 39], localization algorithms [44, 57], data collection [32] and dissemination protocols [48]. Network-centric research forms the bulk of the early period of WSN research (from the late 1990s to early 2000s), as researchers tried to prepare for the reality of cheap generic devices that could be deployed in their thousands. Because network-centric work must formulate a model of application usage and network scalability, these approaches are typically evaluated through a combination of toy applications and network simulations. Although this presents a lower barrier to entry for researchers, the resulting work is often not readily applied in realistic deployment scenarios [1, 65]. Simulations are only as good as the assumptions that they make about the motivating application, the deployment environment and the behavior of the sensor nodes. Many research simulations employ assumptions about radio propagation that have been broadly disproved through empirical study [53, 74–76]. This casts doubt on all but the most carefully circumscribed wireless network simulation results. The end result is that if network-centric approaches are to be realistically used, they must also be evaluated realistically. While this trend is becoming more common, it is by no means the standard in WSN research.

Finally, the *device-centric* view builds WSN design choices around an existing hardware platform, meaning that the platform dictates the extent to which the application goals can be met as well as the type of protocols which can be implemented on the device. Thus, it is important to select a platform that makes a good fit to the target application. When real-life application functionality must be fitted to the capabilities of an unreasonably constrained platform, the application functionality must be stripped down to match platform capability, or else extra custom hardware must be added [43, 47]. Conversely, *under*-constrained platforms can pose logistical problems, in particular regarding battery lifetime. However, using an under-constrained system permits the designer to gain experience with the problem before optimizing the design, thus reducing the engineering effort required for a proof of concept.

The device-centric view is a popular option in computer science research as, by using commercially available devices with software support, researchers avoid a costly and often impractical process of hardware design and consruction. Highly constrained sensing devices like the Mica2 [23] and the TelosB [64] have been adopted as de-facto general-purpose WSN platforms. These platforms are optimized for small size and low energy usage by using low bandwidth radios, kilobytes of

RAM and program memory, and low processing power (microcontrollers instead of microprocessors). However, using these platforms it is difficult to implement applications that are more complex than simply sampling and forwarding data wirelessly. Whilst this type of optimization should not be ignored, these restrictions increase the difficulty of performing exploratory development of applications and related processing.

Some classic examples of platform capability heavily affecting the application performance have been in the range estimation and self-localization area of WSN research. Several efforts using the Mica2 for acoustic ranging and self-localization have had to work around the limited platform capabilities, trading off precision, range and accuracy for reduced processing and memory footprints [43, 67, 73, 74, 78]. It is interesting to note that highly accurate range estimation solutions using marginally more processing power or memory had been demonstrated in parallel to constrained systems research [31, 46, 61], adding further weight to the limitations of the *device-centric* view. It should also be noted that the platform constraint has lead to a novel set of self-localization techniques using interferometry [41, 42, 54] that would probably not be considered from an application-centric point of view.

To summarize, the *device-centric* view forces optimization before applications are fully realized when the chosen platform is overly resource constrained. The *network-centric* view tries to produce generic answers without fully understanding the differences between applications that prevent generality. Both of these approaches imply that device constraints take precedence over meeting the application requirements, or that a generic enough set of protocols or approach will be sufficient to describe any possible application. It is the authors' opinion that these views are flawed, and that in order to understand the realistic benefits of WSNs for a particular domain, it is important to adopt an *application-centric* view. In application-centric development, the requirements of the application are the over-riding priority. These requirements do not necessarily have to be met using a specific approach: it is far more important to provide a working system that meets the application goals than treating in-network or distributed processing as first principles.

Part of enabling wider adoption of WSNs is about being able to provide meaningful results from in-situ deployments. Once the application needs have been met, and the performance of the deployed system has been evaluated in-situ, the system and platforms can be optimized to meet other criteria in the design space, such as size and energy consumption. Similarly, many network features that are an a priori part of the Smart Dust vision, such as in-network processing and collaborative behavior, will only need to be implemented in an application-centric effort if they emerge from the application needs. One example is the development of the VoxNet acoustic sensing platform described in Chap. 5. When the system was initially deployed by biological researchers, all of the raw acoustic data was recorded and later uploaded: the system did not perform any local processing of the acoustic data. Subsequent analysis of the network data transfer rates from in-situ deployments revealed that in some cases, nodes could reduce their own data transmission times by processing some data locally and sending smaller amounts of data; the decision of when to do this could be made dynamically based on the current state of the network. Additionally, taking a network-wide approach to what data was important to the application

allowed for reducing the number of transmissions that needed to be made (regardless of local processing or not). Neither of these refinements were design choices in the initial development of the system, in which the aim was simply to meet the application goal of localizing an acoustic source in an on-line manner. However, local processing was later shown to be useful in reducing network usage and thus end-to-end localization latency.

2.2.3 Meeting Application Requirements

The importance of meeting application requirements is a unifying thread in the case studies presented in Chaps. 4–10. To best prepare the reader for that material, we distill some generic lessons and principles that apply to the examples in this book as well as future applications the reader may encounter. Application-centric designs typically encounter two key challenges: technological and development constraints, and translating requirements from the application domain into the WSN context.

Technological and development constraints include constraints on what is possible. No system is failure-proof, but wireless links, limited energy, and embedded hardware further fundamentally limit the reliability of most WSN systems. Unforeseen hardware and systems failures, and limited operational visibility impose limits on reliability in a deployed setting. Development constraints including time and budget constraints, environmental conditions, and hardware availability and capabilities, are also key limiting factors.

The second key challenge in developing a WSN system to support a specific application is ensuring that the sensing requirements are informed by domain experts. This process of translating the requirements from the application domain to those of a WSN system can be quite challenging and requires good communication with the domain experts. In this process, three key aspects should be considered:

Ensure that the right thing is sensed and that it is sensed correctly. To correctly measure a phenomenon, the sensor hardware must be properly configured and coupled to the phenomenon, and the sampling parameters must be adequate, including sample rate, sample resolution, and acceptable noise level. If the sample rate is below the Nyquist rate for the signal, ensure that proper filtering is implemented in the front end. When measurements are made at multiple points, ensure that the spatial density is adequate. Ensure that sample timing is recorded with sufficient accuracy, and that timestamps are sufficiently consistent across nodes. These considerations will affect how many nodes are required, the data rate that must be processed at each node, and will define system level requirements such as location discovery and time synchronization. For applications that are exploratory in nature, the sensing requirements may not be known a priori (i.e. existing domain measurements may not be adequate to make these estimates). In these cases, iterative deployment is necessary, where the application requirements become refined as more data is gathered and analyzed.

Ensure that the network performance matches the application. The wireless network must be capable of allowing data transfer that meets application requirements for timeliness. Because wireless communication performance is dynamic, sufficient provision must be made to provide headroom in supporting the minimum data transmission requirements of the application. Understanding how the relevant phenomena should be sensed (in space *and* time) will help to understand the volume and pattern of data transmission, as well as the expected network topology. If the size of the required network is small then complex multi-hop routing algorithms may not be required, or may be assumed to have a very limited maximum number of hops. Similarly, if the data generated is small and infrequent, on-node processing techniques may not be required. In choosing or designing communication protocols, it is important to be aware that simulation results often do not resemble observed behavior in the field and that this may drastically affect the performance of a fielded system.

Ensure that the system captures system performance data and application metadata in addition to the required application data. Application data must be captured and analyzed correctly, and must be correctly viewed by the target user. Correctness of data refers not only to sensor calibration, but also to whether values sensed are within expected ranges and thus are not faulty or unreasonable. Recording and analyzing additional application data and metadata can enable increased tolerance of the application to system or sensor failures. The system should be appropriately instrumented so that the systems developer can monitor its operation at run-time. Appropriate system instrumentation is important both to verify that the system is running correctly, as well as to collect data that will improve the performance of the system on subsequent iterations.

Successful WSN applications should be operable by non-developers, which means that they must be easy to operate and common functions must be readily accessible. Whilst this is in part a usability issue, it also points to a problem: what might seem an intuitive interaction for a computer scientist or engineer could be completely confusing for the intended user. Whilst a systems engineer may be interested in information which helps them understand and diagnose the state of the system, the domain expert is far more likely to be concerned with the correct presentation of sensor data gathered by the network to help them make inferences or even reconfigure the sensor network. It is only through working closely with domain experts that these issues can be addressed.

WSN developers concerned with implementation and deployment have taken the above to heart over the past few years and their efforts have pushed the boundaries of the domain repeatedly beyond the "sense and send" systems common till a few years back. Many of the field's theoretical advances, however, share little of these concerns, contributing instead proofs for sophisticated protocols to support large scale networks, constrained as envisioned by the Smart Dust concept.

When comparing the reported practical successes with the theoretical achievements – whether with regard to scale, physical size, complexity, yield or system life – a large gap between the two is apparent. The next section reasons around this issue, indicating the potential gains for the domain should the theoretical and practical communities join forces.

2.2.4 The Practical–Theoretical Divide: Open Research Questions and the Value of Deployment

Since its inception, the sensor network research community has incorporated (albeit disproportionately) both theoretical work and practical concerns, evidencing the uptake of WSNs by both the computer sciences and engineering communities. Conferences such as the ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN), for example, have offered dual tracks, one featuring work with theoretical groundings, the other intended to showcase tools, deployment experiences, and efforts grounded in the realities of deploying systems.

Unfortunately, while the theoretical and practical scientists frequently march under the same banner, there has been less exchange between them than might be optimal for the forward progress of the field as a whole. Sometimes the two communities seem to be talking past each other entirely. Theoreticians show little interest in the practical considerations of operating real systems in the field, or of even the practical implications of implementing and deploying the algorithms they themselves propose. As a result, they tend to produce approaches based on numerous assumptions that researchers who have deployed real systems know to be false, further damaging the chances of their ideas being incorporated into real systems.

On the other hand, practitioners shy away from exploiting the latest theoretical advances into their built systems, fearing that theoretically-grounded improvements will prove illusory in more realistic environments, or that complex algorithms will prove difficult to understand and debug once fielded. As a result, their overly-conservative approach coupled with the amount of time and resources necessary to build and field a real system, results in highly-incremental advances and generally slow progress.

As one example, the so-called “unit disc model” of wireless radio propagation was the basis for many early theoretical works on wireless ad-hoc networking. The unit disc model assumes that a node can communicate perfectly with all nodes within some fixed radius, and cannot communicate with any nodes outside of the disc. As one might expect, this assumption simplifies the radio propagation model and allows the behavior of complex algorithms running on top of ad-hoc networks to be quickly simulated, and in certain cases allows desirable properties of these algorithms to be proven. Unfortunately, as all practitioners know, the unit disc model does an extremely poor job of squaring with the complex reality of wireless wave propagation, where multi-path, fading, obstacles and many other complications challenge any simple model of wireless radio propagation. And for practitioners, the fact that the unit disc model does not match empirical reality is reason enough to discard anything that follows.

This dismissiveness comes at a price, since there are doubtless many theoretical results rooted in the unit disc model that would work reasonably well even with the assumptions of that model relaxed, or with simple workarounds to improve their robustness in real-life settings. But the assumptions that are necessary for theoreticians

to make in order to prove desirable properties become sufficient for practitioners to discard the entire effort. Along the same lines, theoreticians sometimes abandon algorithms and research efforts once they pass the point where it becomes hard to reason about their behavior in well-defined ways, which unfortunately is sometimes right at the point where their behavior might actually be useful. Properties of a system that can be guaranteed assuming a fully-connected topology become impossible to guarantee once that assumption is relaxed. It is possible that the system still has the desired properties, and this could be verified by simply implementing and deploying it, but the interest wanes for the theoretically-oriented researcher.

This fundamental inability to agree on which open questions qualify as research and on how to address them seems poised to widen the gap between the two groups over time, as theoretical work forges ahead less and less rooted in real systems challenges while systems builders react by almost entirely eschewing theoretical contributions when constructing their applications.

A focus on real-life applications and deployment (taken as a natural step in a WSN project), can provide a meeting point for the two communities and help address this widening gap. By addressing real application goals, both groups are forced to "get real": theoretical work must stand up to the vagaries of reality, while practitioners must build systems more complex than simple sense-and-send to accommodate demanding applications. However, this represents a difficult challenge for both groups, and it is as yet unclear to what extent the research and WSN user communities actively incentivize joint proceeds. Theorists are unlikely to need practical validation in order to publish papers. System builders can strengthen their arguments through successful deployment, but the emphasis on success pushes them away from complex algorithms or system designs and towards problems too simple to really require the sort of combined approach that would really benefit both communities.

The reality is that the grand vision of Smart Dust is still a long way off, and there are many way points to cross before we can build anything approximating the systems envisioned by early WSN researchers. Getting there will require concerted and combined efforts from both sides of the practical/theoretical divide, as well as progress in the commercial sphere to generate the volumes required to bring down hardware costs.

Ideally, deployments should incorporate the best efforts from both practical and theoretical advances; we show later that they have an important role to play in setting future research agendas that theorists and practitioners can work towards together, supported by commercial developers. The ultimate aim of all three communities is widespread adoption of the WSN technology, and as such it is not uncommon to see WSN researchers start spin-off companies in order to see their research endeavors become commercial success. Numerous examples of successful spin-offs exist, such as Sentilla, Dust Networks and Arch Rock (all of whom happen to be founded by alumni or faculty from the University of California, Berkeley); these companies have brought to market products that draw heavily from the founders' pioneering theoretical and practical research in networked sensing.

2.3 Starting Points for Development: Existing Platforms

In keeping with the potential for WSN technology to transform sensing and actuation, there are a range of WSN platforms and solutions available to purchase on the market today, comprising both sensor node hardware and sink-side software infrastructure. Some of these solutions are full, end-to-end products aimed at a particular application, some are generic solutions which can be configured to meet a variety of target applications, and some are fully customizable platforms that are suitable for experimental research. Each of these different classes of solution target a different part of the WSN market: application-specific solutions are targeted at commercial, industrial or home users who require a complete, end-to-end solution that enables a very specific application with minimal configuration; generic solutions provide a means to prototype WSN applications within a limited design space; and research platforms allow a researcher to experiment fully with WSN development.

For the researcher, the more application-specific solutions are often less expensive, but they may not be extensible enough to be applied to a particular research problem. It is therefore up to the user to be able to identify these different platforms and choose which is most appropriate for the application at hand. This section aims to support this choice by providing an overview of popular solutions from each category as well as their limitations.

Section 2.3.1 discusses end-to-end solutions, Sect. 2.3.2 discusses generic solutions and Sect. 2.3.3 discusses research platforms. Finally, Sect. 2.3.4 comments on the suitability of the state of the art to support both the researcher and the commercial markets these platforms aim to serve.

2.3.1 End to End WSN Solutions

End to end application solutions to wireless sensing are full hardware and software systems targeted at a specific application; as such they can be bought off the shelf and used out of the box. Application-specific solutions will either be wire-replacement in existing wired sensing systems or used for novel applications that can only be enabled through WSN technology. Typically, these systems consist of a number of sensing nodes with pre-installed, configurable software, a gateway device to connect the network to a desktop or server, and a server-side application suite that allows visualization or manipulation of gathered data. Other functionality may include the ability to set and receive warnings via text message or email when certain data threshold points are reached in the data being sensed. Table 2.2 shows a selection of several commercially available WSN-based systems, including the company that markets them, the wireless technology used and the target application. Most of the solutions shown transmit in the 2.4 GHz ISM band and use protocols built on top of 802.15.4 MAC and PHY standards, such as ZigBee or 6LowPan; only Grape Networks' solution does not. The first four rows in Table 2.2 are wire-replacement solutions, where the company has adapted its traditional

Table 2.2 End-to-end, application-specific WSN solutions, showing the company, wireless technology, target application and class

Company	Wireless Technology	Target application	Application class
PPM Technology	ZigBee	Indoor air-quality	Wire replacement
MicroStrain	802.15.4/FDMA	High data rate sensing	Wire replacement
Soil Instruments	2.4 GHz	Structural monitoring	Wire replacement
OnSet	802.15.4	Environmental monitoring	Wire replacement
Grape Networks	Custom/433 MHz	Microclimate monitoring	New application
ArchRock	6LowPan (IPv6)	Intelligent energy analytics	New application
Sentilla	802.15.4	Intelligent energy analytics	New application
SynapSense	802.15.4	Data center monitoring control	New application

wired sensing hardware to support wireless technology, improving deployment convenience and allowing remote data collection for users. Soil Instruments [20] enable wireless sensing for structural health monitoring with compatible sensors from their range; PPM Technology [17] provide indoor air-quality monitoring equipment for home and industrial settings; extensively used in environmental monitoring for agriculture and research, OnSet Corp [16] provide wirelessly enabled data loggers to which their smart sensors can be attached; and MicroStrain [14] provide a high data rate solution, where high rate data can be streamed wirelessly (although in node processing is not supported).

The remaining products shown in Table 2.2 are for novel applications enabled by WSN technology. First, Grape Networks [11] has a specific environmental monitoring product designed for monitoring vineyard conditions. The product's functionality is simple but robust: temperature, humidity and solar radiation are sampled at each node and this data is relayed back to a central server. Unlike many other available WSN platforms, the product uses a nonstandard communication protocol, which complicates integration with other platforms. The Grape Networks' Water Management platform enables fine-grained automated irrigation management; the operator can set thresholds via the Internet and register for alerts.

In keeping with the green, energy saving related WSN opportunities mentioned in Chap. 1, the final four rows in Table 2.2 show companies that have focused on very specific *energy-monitoring* applications. Using WSNs to monitor power usage of household or industrial devices can save money on energy bills and can enable greener, more efficient operation. Clearly, the climate issue is a strong motivator for these systems and there is a good case for widespread adoption if current trends continue. Sentilla [19] (originator of the TMote Sky platform) has focused its attention solely on *intelligent energy analytics*, a technology that allows end users to monitor energy usage using a WSN with a view to cost cutting and efficiency improvements. Arch Rock [7] has taken a similar approach with their Energy Optimizer solution based on their 6LowPan platform offerings. 6LowPan provides IPv6-based communication to nodes in the network, and simplifies integration of WSN data streams with existing enterprise infrastructure. Another company, SynapSense [21], offers a very specific solution for data center energy management/cooling control. This solution combines sensing with actuation, and creates a closed-loop system that

can be installed in data centers, dynamically applying cooling where required. The advantage of this technology to the end-user is that it is a proven solution: the system performs this task out of the box with minor configuration, requires no additional wiring and lasts for months or years between battery changes.

None of these systems are customizable outside of the limited tuning and configuration interfaces that the software provides. Limiting the number of configuration options focuses the functionality of the software, simplifying it such that the user can easily install and maintain it. However, this makes the products unsuitable for use outside of the anticipated application.

2.3.2 Generic Solutions

The next class of commercial WSN systems can be broadly classified as generic solutions. Compared with the application-specific solutions in Sect. 2.3.1, these systems are targeted at a broader class of applications. As such, these products do not aim to provide a comprehensive solution to a specific problem, instead providing software tools and hardware to enable a variety of applications to be prototyped and implemented. A strong focus here is on hardware and software that non-WSN expert vendors or original equipment manufacturers (OEMs) can use to integrate into systems that will be sold on to an end user.

In this class of WSN products, it is common to see *development kits*. These are starter packages, containing enough sensing hardware for a user to set-up a small network, collect and display some rudimentary data (such as temperature), as well as potentially reprogram the system with different applications. The typical contents of a development kit are one or more sensing (and/or routing) nodes, a gateway device and server-side software to collect and visualize data, as well as a library and associated API or integrated Software Development Kit (SDK) to allow the user to develop their own software. The extent of the system's reconfigurability varies greatly from product to product.

Table 2.3 shows a selection of generic solutions, the wireless technology they focus on and the product's target application class. Most of these products are targeted at home or industrial monitoring and automation, and all provide sensor nodes

Table 2.3 OEM WSN solutions, showing the company, wireless technology and application driver for the products

Company	Wireless Technology	Application driver
Dust Networks	WirelessHART	Multiple industry (WirelessHART)
Sensinode	6LowPan (IPv6)	Multiple industry (MBUS focus)
Millennial Net	2.4 GHz	Multiple industry
Jennic	ZigBee PRO	Multiple industry/home
Ember	ZigBee	Multiple industry
TI/Labview	802.15.4	Embedded control systems
EnOcean	868 MHz/315 MHz	Multiple industry/home (energy harvesting)

that boast long battery life and reliable multi-hop, mesh networking. Products are differentiated by the target application class, the type of network communication protocols they use and the level of reconfigurability provided through the API/SDK.

Dust Networks [8] provide WSN sensor nodes and gateway devices for industrial applications and automation, focusing on WirelessHART [6] and ISA100 standard compatibility and allowing existing HART systems to be easily adapted to WSNs. Dust Networks' main selling points for their sensing system are WirelessHART compatible nodes with low power consumption (a quoted 7–10 year lifetime), an adaptive, frequency-hopping, multi-hop mesh network, data security and 99.99% reliability in data transfer, as well as a set of development libraries and associated API which allow the user to write HART programs. EnOcean [10] focus specifically on an energy harvesting platform that can harvest and store all the energy required to operate, through solar, thermal or vibration energy harvesting circuitry. They provide a catalog of sensors from third parties that can be used directly with the hardware (that is they are approved for use). Ember [9] offer a product whose multi-hop networking components are based on the ZigBee PRO communication stack. This allows Ember's products to support the ZigBee PRO application profile set, allowing applications to be built for home automation, energy monitoring and industrial automation. Sensor nodes have ports for various types of sensors to be attached (although they do not provide lists of approved sensors), and a software API and SDK allows guided application development for applications within the ZigBee PRO application profile set. Texas Instruments/Labview [13] provides WSN modules that can be programmed within the LabView environment, allowing a WSN to be integrated into existing LabView software. This can enable those familiar with LabView style visual programming to work easily with WSN nodes. Sensinode [18], Millennial Net [15], and Jennic [12] are further examples of companies that provide development kits; of note here is Millennial Net's five *data models*, which are different data collection modes that their sensor nodes can be run in, to reflect the type of application the user desires: periodic sampling (sense and send), event driven (based on specific event), store and forward (sample then download), polling (request data centrally) or on-demand (grab data from nodes whilst in proximity). Finally, both Sentilla [19] and Arch Rock [7] provide development kits to allow vendors to prototype with similar hardware to that of their flagship end-to-end solutions (see Sect. 2.3.1).

This class of WSN systems provides a route for interested clients to try proof of concept wireless sensor networks, and afford enough reconfigurability to allow for many sensing applications. However, as with the application-specific WSN systems, the more targeted the usage models are, the more difficult it is for the user to adapt the system for more general or entirely different applications.

2.3.3 Research Platforms

The type of WSN systems that fall into the COTS WSN research platforms category are those that are highly customizable, are intended to be used by researchers that are

familiar (or will quickly become familiar) with them. Research platforms lower the barrier to entry for applied research: practitioners can carry out WSN research with known platforms within the research community without the need to develop their own platforms. The main distinction between the platforms discussed in this section and the generic solutions presented in the previous section is that, the research platforms are intended for researchers to experiment with the wireless aspects of the platform, for example MAC protocols and multi-hop routing algorithms. In these cases, the researcher may want to try various approaches through experimentation. The same is not true for the generic products that are oriented toward providing a wireless solution that a full system can be built upon, or a system to meet a specific application need.

Table 2.4 gives an overview of commercially available sensing platforms for WSN research (this does not include custom research platforms that are not publicly available), including the relative pricing of individual platforms. In general, WSN research platforms can be divided into two categories: mote class and microserver class platforms.

Mote platforms are typically based on 8 or 16-bit flash microcontrollers, with ∼1 kB of RAM, ∼100 kB Flash memory, and a wide variety of readily accessible I/O ports and built-in ADCs. Mote modules typically integrate a low power RF interface supporting 802.15.4, ZigBee or Bluetooth, or in some cases supporting a custom or user-defined MAC. These platforms are suitable for research into

Table 2.4 The reader should note that only commercially available platforms are included. It should also be noted that not all 802.15.4 compatible radios are ZigBee compliant (ZigBee is a stack which sits on top of the MAC and PHY layers defined by 802.15.4). 802.15.4 uses the 2.4 GHz part of the ISM band for communication

Platform	MCU/CPU	Comms	Node cost	Target apps
AmbioMote	MCU	custom 2.4 GHz	$200	SHM type
Arduino	MCU/8-bit	ZigBee or Bluetooth	€45–95	hobby/gadget
BTnode	MCU/8-bit	800 MHz and Bluetooth	€165	research
Cricket	MCU/8-bit	868 MHz	$195	localization
Iris	MCU/8-bit	802.15.4	$115	research
MicaZ	MCU/8-bit	802.15.4	$99	research
Mica2	MCU/8-bit	433 or 868 MHz	$99–125	research
SquidBee	MCU/8-bit	ZigBee	€130–150	hobby/gadget
TNode/KeyNode	MCU/8-bit	315–868 MHz	€65–99	research
Mulle	MCU/16-bit	802.15.4 or Bluetooth	€139–149	research
Pioneer	MCU/16-bit	802.15.4	$499	industrial
MSP430 (Scatterweb)	MCU/16-bit	433 or 868 MHz	€98	research/industrial
Shimmer	MCU/16-bit	802.15.4 and Bluetooth	€199	medical monitoring
TelosB/TMote	MCU/16-bit	802.15.4	$99	research
Gumstix Verdex/Overo	CPU/32-bit	Bluetooth	$129–219	hobby/gadget
IMote2	CPU/32-bit	802.15.4	$299	research
SunSPOT	CPU/32-bit	802.15.4	$750 / 2	hobby/gadget

communication protocols, low power operation and simple sensing applications; however they are not well-suited to applications that require substantial on-node processing at high rates.

Microserver class platforms provide 3–4 orders of magnitude greater computational and I/O capabilities, in exchange for larger energy requirements. These platforms often have a slightly larger form factor than a Mote, and are typically based on a 32-bit microprocessor, often a smartphone class application processor with advanced features such as multiple pipelines, large caches, frequency and voltage scaling, and SIMD or hardware floating point support. These platforms typically host megabytes of RAM and flash storage and a range of high-speed I/O interfaces such as MMC, SDIO, USB, and Ethernet. Many platforms support integrated RF interfaces such as Bluetooth or WiFi, with ZigBee being less common but relatively easy to integrate via a serial interface. These platforms often support built-in audio interfaces, and can readily support high data rate collection and processing. The relative pricing, capability, and form factor of platforms in not related to their functionality: a sensing platform with a 32-bit processor can be purchased for similar prices to 8 or 16-bit microcontroller-related devices, and can be found in a similar sized device. Some of these platforms cater for specific classes of application, for example the Shimmer platform is purposely developed to cater for mobility based applications: being lightweight, with small form factor, and having a suite of accelerometers, it lends itself well to a variety of medical/health-care related applications.

2.3.4 Discussion

This section has provided some insight to the aspects of WSN research which are currently considered to be commercially viable, notably, wire replacement, reliable data collection, low power operation and mesh networking. The problems that the solutions in Sect. 2.3.1 address are based around wire replacement and simple sensing and data archiving, with limited reconfigurability. True end-to-end solutions will typically address a specific problem, and provide a complete solution for the consumer. As such, these products are marketed in terms of their application-related benefit, rather than the specific wireless technology employed, or the specifications of the nodes; these products are not suitable for reconfiguring to meet a different application.

With the generic solutions presented in Sect. 2.3.2, there is a stronger focus on reconfigurability – allowing extra sensors to be added to node hardware, and providing an SDK to re-program sensor nodes to run user-configured applications. These systems still abstract away the wireless, mesh-networking side of the WSN, making them unsuitable for research related to new communications protocols, but does make them well-suited to OEMs who are producing or integrating wireless sensing into products. There is little emphasis on the areas which WSN research still finds challenging, most notably in-network processing and high data rate sampling. For most systems that are currently available, the on-node complexity is still low: nodes can sample data (usually at low rates) and forward it (potentially over multiple hops)

to a base station, where it can be displayed or archived. Finally, the research platform class of sensing hardware has the most customizable hardware and software potential, but is also the most complex to develop for.

As previously noted, many of the platforms surveyed in this section have made their way from the academic research labs to the market in relatively short periods of time. Successful technology transfer to the commercial sector has been enabled not only by the application of research but also by a deep understanding of the industrial community's needs and a clear identification of the barriers to be overcome in order for the WSN products to be adopted. The next section, whose title has been borrowed from one of (Dust Networks' CTO) Kris Pister's presentations on the subject [62], offers a wider perspective on the issues around the commercial perception of the WSN technology, and points to the perceived barriers to WSNs' wide adoption.

2.4 Who Is Taking Off: the WSNs or the Market Analysts?

There is no doubt that the drive provided by early, extremely ambitious visions of pervasive computing has left a mark on the WSN research domain by encouraging more theoretical advances than practical work.

Practical work is seen by many researchers as justifiable and essential only when the concepts and knowledge are ready to be translated into marketable products, and the industrial need for WSN products to date has been tepid. Realistically, advances in WSNs have always been characterized by a technology push rather than a market pull. Hence it is understandable that, in the absence of a firm market with specific and specified demands/requirements, much of the WSN research work has been either purely theoretical or limited to practical feasibility studies. However, the promise of great WSN products and great sales has always been on the horizon: prompted by a single obvious financial incentive of eliminating wiring costs encountered in existing measurement systems, enthusiastic analysts predicted inflated markets and raised both researchers' and investors' expectations since the early years of this millennia.

The authors maintain that real-life applications successes have a high impact on the technology's adoption prospects. Sturdy, demonstrable and sustainable solutions to clearly specified problems and the dissemination of those solutions through commercialization would contribute greatly to wider adoption of the WSN technology. Moreover, ensuring fast uptake would also imply that the said commercial WSN products would enable end-user, application specific customization without extensive developments or need for specialist involvement of WSN teams. Some of the off-the-shelf WSN systems available today permit customization, albeit at a basic data acquisition level (see for example the home monitoring solutions offered by ArchRock [7], EnOcean [10] and a handful of other companies). However, bridging the gaps between *data* and *information*, or *data* and *actuation*, for example, remain in the realm of the computer scientist rather than being facilitated for the end-user.

Thus, in between the concerns of the WSN research community and the technologists, sit the market for WSN systems and solutions and its stakeholders. Gartner's infamous Hype Cycles for Emerging Technologies [30] captured the commercial and user perception of WSN technology over the past few years and allow insight into the remaining barriers to adoption for WSNs.

2.4.1 WSN Forecasts and Gartner's Hype Cycle

Introduced in 1995 as a commentary on the common patter of human response to technology, Gartner's Hype Cycle provides a cross-industry perspective on the technologies and trends that managers should consider in developing emerging-technology portfolios. Gartner's Hype Cycle characterizes the typical progression of an emerging technology from *over enthusiasm* through a period of *disillusionment* to an eventual *understanding* of the technology's relevance and role in a market or a domain. Each phase is characterized by distinct indicators of market, investment and adoption activities [30].

The Gartner cycle aims to provide a model to discern hype from viability and to estimate when visionary claims will pay off, if at all. In this model, a technology's relevance to solving real business problems and exploiting new opportunities is encapsulated by the cycle's "time to maturity" prediction. Gartner's Hype Cycle [30] consists of five phases that can be used to categorize emerging technologies (shown in Fig. 2.5): *the technology trigger* represents the initial press hype in the technology,

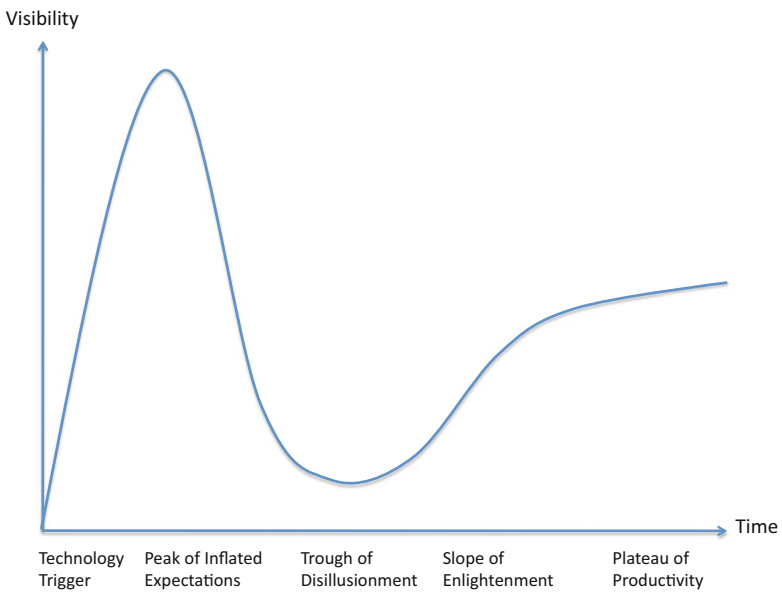


Fig. 2.5 Gartner's Hype Cycle (Image reproduced with permission)

the peak of inflated expectations represents the press-led unrealistic over-hyping over the technology, *the trough of disillusionment* represents the lack of hype as the technology fails to meet expectations, *the slope of enlightenment* represents a business-led resurgence in applied use of the technology, and the *plateau of productivity* represents the steadying state of use and acceptance of the technology in its particular market.

With the benefit of hindsight, Gartner's predictions for the WSN technology (denoted as Mesh Networks) featured in the 2005–2009 Hype Cycles [25–29] paint an accurate view of the domain's evolution as driven by research, with numerous attempts to break through commercially, in association with the emergence of standards and remaining high costs of WSN systems. The *Technology Trigger* for WSNs was clearly the Smart Dust concept that generated enthusiasm in many circles, from funding bodies to the popular press [40, 69].

2005 is taken, in what follows, as a discussion point, given that sensor networks were best positioned in the cycle at that time; WSNs were predicted to reach the *Plateau of Productivity* within 2–5 years, and hence see mainstream adoption and proven viability and technological pay-off, as shown in Fig. 2.6 [5, 33].

At the time, market analysts saw a wealth of opportunity for WSNs, particularly as wire replacements within the classical MEMS markets and smart metering. 16 million sensor nodes were predicted to be dispatched by 2009; the smart metering market was estimated at $1.6B for 2009, rising to $5–7B by 2014; more enthusiastic predictions were for a $50B total WSN market in 2014. To date, the market did not live up to these expectations. Much of these strong predictions came

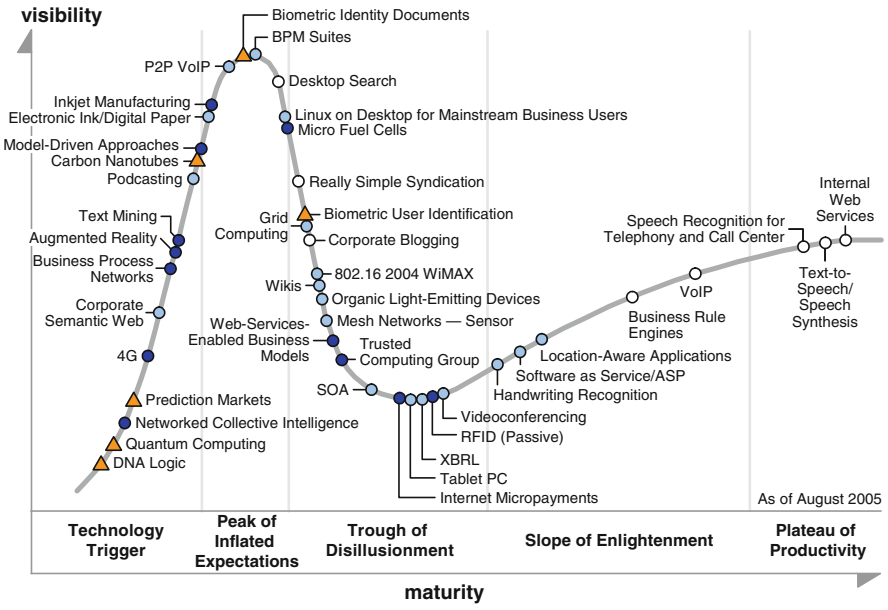


Fig. 2.6 Hype cycle 2005 [24] (Image reproduced with permission)

from pure analysis of wiring costs, overestimating the maturity of the technology at the time and underestimating the costs of applications development, deployment, maintenance and hardware. The WSN market reality in 2005 was $326 M, mostly within the manufacturing industry; examples include cargo and long-range asset tracking [3], building automation and industrial equipment monitoring [4].

Overestimating the technology maturity was also potentially prompted by the launch, in 2005, of many WSN products. 2005 was also probably the peak year in the number of research platforms produced by various research groups. Moreover, by 2005, standards for low-power, low-data rate communication stacks had become established: the initial 802.15.4 MAC and PHY later standards for low-power, low-data rate communication had existed since 2003, and the initial ZigBee specification that built routing and network formation on top of 802.15.4 had existed since 2004. Since 2005, ZigBee and 802.15.4 have seen significant revisions (in both 2006 and 2007) and new standards have arisen, such as 6LowPan (IPv6 over 802.15.4, released in 2007) and WirelessHART (a wireless, low power version of the HART systems control protocol) [6].

Following Hype Cycles were in comparison much more conservative, predicting technology maturity in 10 years (2006), and showing cycles of advance and regression between the *Slope of Enlightenment* and *Peak of Inflated Expectations* (2006, 2007, 2009). A parallel can be drawn here with the state of the art in research in 2006, for example, a year when many open questions were brought forth by difficult deployments resulting from projects started in 2003–2004 [22, 45, 68, 71]. This, together with the difficulties faced by commercial adoption pioneers (such as the deployed systems' unsuitability for harsher environments, the exaggerated maintenance needs, the user data overload, and the slow standardization processes) might explain the field's perceived stagnation and regression of expectations. As new potential markets opened, the predictions for WSNs continued to flourish, however, over the years. Of note are those related to the areas emerging from the global energy crisis, such as smart metering, demand response, HVAC control, and generic Home Area Networks [34–36].

In his thorough analysis of the discrepancy between the WSNs' real market value and the analysts views, Pister [62] highlights the barriers to adoption, as identified by various OnWorld surveys: reliability, standards, ease of use, power consumption, development cycles and node size, in this order of importance. Not surprisingly, the above are not the challenges set by the Smart Dust vision but the very challenges brought forth by the WSNs practical deployers community.

However, with installation, connection and commissioning costs expected to drop considerably through the use of mesh networking and the recent availability of long-life robust products, presently, the future looks brighter for the domain than ever since inception.

To conclude on the above, wide adoption of WSNs is yet to be witnessed, partly due to hype and inflated expectations, early adopters disappointment and the very real barriers of cost, reliability, standards and maintenance needs. These barriers are currently being lifted by better products making their way to the market, the emergence of some potential killer application for WSNs (discussed next) and

helped without doubt by frequent and sustained deployment of research products. Resolving in-situ, real-life applications can help considerably both in terms of the learning and experience gained by the developers but also by boosting confidence of buyers. Such systems will be practical proofs of technology viability and its applicability.

2.4.2 Current and Forthcoming WSN Research and Commercialization Opportunities

It is clear that, as with many emerging technologies, political shifts will drive WSN innovation, economic growth affects WSN penetration, and regulatory influences strongly affect markets. It is also commonly accepted that the class of "green issues" is presently at the forefront of the world's political agenda. Drawing from these, it is conceivable that the widespread use of WSNs could be, at least in the short to medium term, within the environmental monitoring area.

Environmental monitoring and the associated large class of "environmental" applications could, potentially be the *killer application* WSN researchers and vendors have been looking for. There are several key factors supporting the assertion:

1. *The economies of scale* – all environmental applications (be them to do with the outdoor natural environment monitoring or indoor monitoring) involve large geographical scale deployments and require hundreds, thousands or millions of instantiations; for the built environment for example, millions of buildings, residential and commercial have monitoring needs hence billions of sensors would need to be deployed;
2. *Political drive* – global warming is, and will likely continue to be, a high political agenda issue with a number of economic packages put in place to cater for the forthcoming "green" standards and policies. Examples here are the 2009 USA's "incentive package" for smart energy monitoring, the real-time energy grid, cleantech, etc. Policy will not only push towards rapid WSN development but has also brought both the WSN companies and the technology itself at the forefront of the research and commercial agenda. At the time of writing (January 2010), many WSN products vendors have already focused their WSN business to the "green" issue, delivering generic WSN environmental monitoring solutions;
3. *End user scale* – given that nearly 70% of the average household utility bill could be influenced by WSN applications to temperature and lighting control, household energy usage monitoring and interactive energy management might provide the largest class of WSN users - the world's population.

Whilst the above are likely to drive down the cost of hardware, encourage further emergence of standards and lead to increased reliability systems with easy-to-use functionality, the same application domain provides researchers with a large number of challenging opportunities: slow data rate sense-and-send solutions suitable for

rapid productization are welcomed within the environmental monitoring domain, as well as sophisticated high data rate systems (such as those dedicated to structural and seismic monitoring for example); both need to be initially catered for by the academic researchers, on the theoretical and practical camps. Further, environmental applications imply a need for long life, often beyond that offered by batteries and sophisticated energy management techniques alone. Research (followed by technological adoption) into energy harvesting technologies integration as well as user-driven information extraction strategies will need to be delivered by the research community. Cheaper, higher accuracy MEMS sensors and on-chip packages are needed. More reliable, scalable communications, web integration and integration into existing IP infrastructures are essential and need development. All above form a full research chain from the development of MEMS to the WWW.

Once adopted, the WSN technology is likely to influence strongly other economy sectors: the *Industrial Sector* is likely to undergo work-flow churn as WSN applications increase safety, reliability and efficiency of industrial facilities. Frequency agility requirements are driving evolution toward industrial standards. Automatic Meter Reading (AMR) management proves a very significant opportunity in the *Utilities Sector* and finally, the growth of personal and national security demand drives considerable growth in WSNs, in border control, access, and defense.

2.5 Summary of Strategic Recommendations

This chapter discussed the push-pull tensions between a variety of facets of WSN design and development. The tenets of the Smart Dust vision often contrast with the challenges faced by developers striving to provide solutions to real-life, clearly defined applications. Based on our investigation of deployment case studies, we can make several recommendations that we believe will improve the tractability and overall impact of WSN applications:

1. Collaborate with end-users to formally define application requirements and evaluation criteria.
2. Involve end-users throughout the development cycle, demonstrating end-to-end results at intermediate stages, and
3. Maintain a clear motivation for development, ideally based on a realistic business model.

Throughout this process, strong end user involvement and leadership is essential to motivate the need for the WSN solution and to define system requirements and performance criteria. Having the right technical blend in the development team is equally important, in order to support both hardware and software development.

A key part of a successful development strategy is the *application centric* approach to system design. We flesh out this strategy by stressing several essential aspects of development common to most WSN applications:

1. Ensuring that the right thing is sensed and that it is sensed correctly,
2. Ensuring that the network performance matches the application, and
3. Ensuring that the system captures system performance data and application metadata in addition to the required application data.

These, together with care for the system's usability and functional accessibility by the end-user form the starting point to a successful application.

Although generic solutions to implement and deploy the grand pervasive computing vision have yet to arise, the drivers for the WSN domain are stronger than ever and much has been learned within the past years. Such solutions need to be sought *now* and the uptake of the technology in a variety of domains will enable exactly that: finding the answers for the real-life problems which WSNs will impact most.

Acknowledgments We gratefully acknowledge the contributions of the authors of the "snapshot" case studies, and their collaborators on the projects described. The following paragraphs are an incomplete listing of those people who helped to make the referenced deployments happen.

Bangladesh Groundwater Monitoring: Eddie Kohler (UCLA); Jenny Jay (UCLA); Thomas Harmon, (UC Merced); Charlie Harvey (MIT); Borhan Badruzzaman (BUET); Deborah Estrin (CENS/UCLA); Nithya Ramanthan (CENS/UCLA and Lorax Analytics), and support from the NSF Center for Embedded Networked Sensing.

Peru Seismic Station Deployment: co-principal investigators Paul Davis and Deborah Estrin (CENS/UCLA); project manager Richard Guy (CENS/UCLA); Igor Stubailo, Matt Mayernik, Emily Foote (UCLA), Robert Clayton (Caltech), and support from the NSF Center for Embedded Networked Sensing, under NSF Cooperative Agreement #CCR-0120778.

WaterWise system: co-principal investigators Andrew Whittle (SMART/MIT) and Hock Beng Lim (Intellisys/NTU); researchers Mike Allen (SMART), Lewis Girod (CSAIL/MIT), Mudasser Iqbal (Intellisys), Frederick Kim (MIT), Ami Preis (SMART), Steven Wong (NTU), and support from the Singapore Public Utilities Board and Singapore National Research Foundation.

References

1. Ali M, Saif U, Dunkels A, Voigt T, Römer K, Langendoen K, Polastre J, Uzmi ZA (2006) Medium access control issues in sensor networks. SIGCOMM Comput Commun Rev 36(2):33–36, DOI http://doi.acm.org/10.1145/1129582.1129592
2. Allen M, Girod L, Newton R, Madden S, Blumstein DT, Estrin D (2008) Voxnet: An interactive, rapidly-deployable acoustic monitoring platform. In: IPSN '08: Proceedings of the seventh international conference on Information processing in sensor networks, ACM Press, New York, NY, USA
3. Anonymous (2005a) Active rfid - a profitable business. Tech. rep., IDTechEx, URL http://www.idtechex.com/research/articles/active_rfid_a_profitable_business_00000396.asp
4. Anonymous (2005b) Dust networks (from the internet archive). URL http://web.archive.org/web/20050510004341/www.dustnetworks.com/applications/main.html
5. Anonymous (2006) Zigbee and 802.15.4 wireless networks - market and ic forecasts, applications and technology analysis. Tech. Rep. RR-ZIG, URL http://www.abiresearch.com/research/1001132-Wireless+Sensor+Networks

6. Anonymous (2007) Why wirelesshart? HART Communication Foundation White Paper, URL www.hartcomm2.org/hart_protocol/applications/white_papers/why_wirelesshart.html
7. Anonymous (2008) Archrock corporation. URL http://www.archrock.com
8. Anonymous (2009a) Dust networks. URL http://www.dustnetworks.com
9. Anonymous (2009b) Ember corporation. URL http://www.ember.com/
10. Anonymous (2009c) Enocean gmbh. URL http://www.enocean.com
11. Anonymous (2009d) Grape networks incorporated. URL http://www.grapenetworks.com/
12. Anonymous (2009e) Jennic ltd. URL http://www.jennic.com/
13. Anonymous (2009f) Labview pioneer module. URL http://sine.ni.com/nips/cds/view/p/lang/en/nid/207289
14. Anonymous (2009g) Microstrain incorporated. URL http://www.microstrain.com/
15. Anonymous (2009h) Millenial net. URL http://www.millennial.net/
16. Anonymous (2009i) Onset computer corporation. URL http://www.onsetcomp.com/
17. Anonymous (2009j) Ppm technology. URL http://www.ppm-technology.com/
18. Anonymous (2009k) Sensinode. URL http://www.sensinode.com
19. Anonymous (2009l) Sentilla corporation. URL http://www.sentilla.com
20. Anonymous (2009m) Soil instruments ltd. URL http://www.soil.co.uk/
21. Anonymous (2009n) Synapsense corporation. URL http://www.synapsense.com/go/index.cfm
22. Barrenetxea G, Ingelrest F, Schaefer G, Vetterli M (2008) The hitchhiker's guide to successful wireless sensor network deployments. In: SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems, ACM, New York, NY, USA, pp 43–56, DOI http://doi.acm.org/10.1145/1460412.1460418
23. Crossbow mica2 (2003) URL http://www.tinyos.net/scoop/special/hardware/#mica2
24. Fenn J, Linden A (2005) Gartner's hype cycle special report for 2005. Tech. Rep. G00130115, Gartner, also available as www.gartner.com/resources/130100/130115/gartners_hype_c.pdf
25. Gartner (2005) Gartner highlights key emerging technologies in 2005 hype cycle. URL http://www.gartner.com/press_releases/asset_134460_11.html
26. Gartner (2006) Gartner's 2006 emerging technologies hype cycle highlights key technology themes. URL http://www.gartner.com/it/page.jsp?id=495475
27. Gartner (2007) Watch mobile, user interface and web 2.0 innovations, says gartner. URL http://www.gartner.com/it/page.jsp?id=511347
28. Gartner (2008) Gartner highlights 27 technologies in the 2008 hype cycle for emerging technologies. URL http://www.gartner.com/it/page.jsp?id=739613
29. Gartner (2009a) Gartner's 2009 hype cycle special report evaluates maturity of 1,650 technologies. URL http://www.gartner.com/it/page.jsp?id=1124212
30. Gartner (2009b) Understanding hype cycles. URL http://www.gartner.com/pages/story.php.id.8795.s.8.jsp
31. Girod L, Estrin D (2001) Robust range estimation using acoustic and multimodal sensing. In: International Conference on Intelligent Robots and Systems
32. Gnawali O, Fonseca R, Jamieson K, Moss D, Levis P (2009) Collection Tree Protocol. In: Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys'09)
33. Hatler M, Chi C (2005) Wireless sensor networks: Growing markets, accelerating demand. Tech. rep., On World, URL http://www.onworld.com/wsn/wirelesssensors.htm
34. Hatler M, Gurganious D, Chi C, Ritter M (2007) Wsn for smart industries - a market dynamics report. Tech. rep., URL http://www.onworld.com/smartindustries/index.html
35. Hatler M, Gurganious D, Chi C (2009a) Energy smart home area networks (hans) - a market dynamics report. Tech. rep., On World, URL http://onworld.com/HAN/index.html
36. Hatler M, Gurganious D, Chi C (2009b) Global smart metering - a market dynamics report. Tech. rep., On World, URL http://onworld.com/smartmeter/index.html
37. Heinzelman WR, Chandrakasan A, Balakrishnan H (2000) Energy-efficient communication protocol for wireless microsensor networks. In: HICSS '00: Proceedings of the 33rd Hawaii International Conference on System Sciences-Volume 8, IEEE Computer Society, Washington, DC, USA, p 8020

38. Husker A, Stubailo I, Lukac M, Naik V, Guy R, Davis P, Estrin D (2008) Wilson: The wirelessly linked seismological network and its application in the middle american subduction experiment. Seismological Research Letters 79(3)
39. Intanagonwiwat C, Govindan R, Estrin D (2000) Directed diffusion: a scalable and robust communication paradigm for sensor networks. In: MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking, ACM Press, New York, NY, USA, pp 56–67, DOI http://doi.acm.org/10.1145/345910.345920
40. Kahn JM, Katz RH, Pister KSJ (1999) Next century challenges: Mobile networking for smart dust. In: Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom 99), ACM, N.Y.
41. Kusy B, Ledeczi A, Koutsoukos X (2007a) Tracking mobile nodes using rf doppler shifts. In: SenSys '07: Proceedings of the 5th international conference on Embedded networked sensor systems, ACM, New York, NY, USA, pp 29–42, DOI http://doi.acm.org/10.1145/1322263.1322267
42. Kusy B, Sallai J, Balogh G, Ledeczi A, Protopopescu V, Tolliver J, DeNap F, Parang M (2007b) Radio interferometric tracking of mobile wireless nodes. In: MobiSys '07: Proceedings of the 5th international conference on Mobile systems, applications and services, ACM, New York, NY, USA, pp 139–151, DOI http://doi.acm.org/10.1145/1247660.1247678
43. Kwon Y, Mechitov K, Sundresh S, Kim W, Agha G (2005) Resilient localization for sensor networks in outdoor environments. In: IEEE International Conference on Distributed Computing Systems (ICDCS05), pp 643–652
44. Langendoen K, Reijers N (2003) Distributed localization in wireless sensor networks: a quantitative comparison. Computer Networks, special issue on Wireless Sensor Networks 43:499–518
45. Langendoen K, Baggio A, Visser O (2006) Murphy loves potatoes: experiences from a pilot sensor network deployment in precision agriculture. In: IEEE International Parallel and Distributed Processing Symposium (IPDPS '06)
46. Lanzisera S, Lin DT, Pister KS (2006) Rf time of flight ranging for wireless sensor network localization. In: Fourth Workshop on Intelligent Solutions in Embedded Systems, pp 1–12
47. Ledeczi A, Nadas A, Volgyesi P, Balogh G, Kusy B, Sallai J, Pap G, Dora S, Molnar K, Maroti M, Simon G (2005) Countersniper system for urban warfare. ACM Transactions on Sensor Networks 1(2):153–177
48. Levis P, Patel N, Culler DE, Shenker S (2004) Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In: Proceedings of 1st Symposium on Networked Systems Design and Implementation (NSDI 2004), March 29-31, 2004, San Francisco, California
49. Lifton J, Feldmeier M, Ono Y, Lewis C, Paradiso JA (2007) A platform for ubiquitous sensor deployment in occupational and domestic environments. In: IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks, ACM, New York, NY, USA, pp 119–127, DOI http://doi.acm.org/10.1145/1236360.1236377
50. Lukac M, Girod L, Estrin D (2006) Disruption tolerant shell. In: CHANTS '06: Proceedings of the 2006 SIGCOMM workshop on Challenged networks, ACM, New York, NY, USA, pp 189–196, DOI http://doi.acm.org/10.1145/1162654.1162655
51. Lukac M, Davis P, Clayton R, Estrin D (2009a) Recovering temporal integrity with data driven time synchronization. In: Proceedings of the 8th international conference on Information Processing in Sensor Networks (IPSN '09)
52. Lukac M, Stubailo I, Guy R, Davis P, Puruhuaya VA, Clayton R, Estrin D (2009b) First-class meta-data: a step towards a highly reliable wireless seismic network in peru. In: Workshop on Sensor Networks for Earth and Space Science Applications: ESSA 2009
53. Lymberopoulos D, Lindsey Q, Savvides A (2006) An empirical characterization of radio signal strength variability in 3-d ieee 802.15.4 networks using monopole antennas. In: Proceedings of the Third European Workshop on Wireless Sensor Networks, EWSN (EWSN '06), pp 326–341
54. Maróti M, Völgyesi P, Dóra S, Kusy B, Nádas A, Lédeczi Á, Balogh G, Molnár K (2005) Radio interferometric geolocation. In: SenSys, pp 1–12

55. Martinez K, Padhy P, Elsaify A, Zou G, Riddoch A, Hart JK, Ong HLR (2006) Deploying a sensor network in an extreme environment. In: SUTC (1), pp 186–193
56. Martinez K, Hart JK, Ong R (2009) Deploying a wireless sensor network in iceland. In: GSN, pp 131–137
57. Nagpal R, Shrobe HE, Bachrach J (2003) Organizing a global coordinate system from local information on an ad hoc sensor network. In: Proceedings of the Second ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN '03), pp 333–348
58. Neumann RB, Polizzotto ML, Badruzzaman ABM, Ali MA, Zhang Z, Harvey CF (2009) Hydrology of a groundwater-irrigated rice field in Bangladesh: Seasonal and daily mechanisms of infiltration. Water Resources Research 45(1):9412, DOI 10.1029/2008WR007542
59. Neumann RB, Ashfaque KN, Badruzzaman ABM, Ali MA, Shoemaker JK, Harvey CF (2010) Anthropogenic influences on groundwater arsenic concentrations in Bangladesh. Nature Geoscience pp 46–52
60. Padhy P, Dash RK, Martinez K, Jennings NR (2006) A utility-based sensing and communication model for a glacial sensor network. In: EUMAS
61. Peng C, Shen G, Zhang Y, Li Y, Tan K (2007) Beepbeep: a high accuracy acoustic ranging system using cots mobile devices. In: Proceedings of the 5th international conference on Embedded networked sensor systems (SenSys '07), pp 1–14
62. Pister K (2007) The future of wireless sensor networks. In: BEARS '07: The Berkeley EECS Annual Research Symposium, URL http://www.eecs.berkeley.edu/BEARS/presentations/07/Pister%20BEARS070215e.ppt
63. Polastre J, Hill J, Culler D (2004) Versatile low power media access for wireless sensor networks. In: SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems, ACM Press, New York, NY, USA, pp 95–107, DOI http://doi.acm.org/10.1145/1031495.1031508
64. Polastre J, Szewczyk R, Culler DE (2005) Telos: enabling ultra-low power wireless research. In: Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN '05), pp 364–369
65. Raman B, Chebrolu K (2008) Censor networks: A critique of sensor networks from a systems perspective. ACM SIGCOMM Computer Communication Review (CCR 2008) pp 75–78
66. Römer K (2005) Time synchronization and localization in sensor networks. PhD thesis, ETH Zurich, Switzerland
67. Sallai J, Balogh G, Maroti M, Ledeczi A, Kusy B (2004) Acoustic ranging in resource-constrained sensor networks. Tech. Rep. ISIS-04-504, Institute for Software Integrated Systems
68. Tolle G, Polastre J, Szewczyk R, Turner N, Tu K, Buonadonna P, Burgess S, Gay D, Hong W, Dawson T, Culler D (2005) A macroscope in the redwoods. In: Proceedings of the Third ACM Conference on Embedded Networked Sensor Systems (SenSys)
69. Warneke B, Last M, Liebowitz B, Pister KSJ (2001) Smart dust: Communicating with a cubic-millimeter computer. Computer 34(1):44–51, DOI http://dx.doi.org/10.1109/2.963443
70. Werner-Allen G, Lorincz K, Johnson J, Lees J, Welsh M (2006a) Fidelity and yield in a volcano monitoring sensor network. In: 7th USENIX Symposium on Operating Systems Design and Implementation 2006, pp 381–396
71. Werner-Allen G, Lorincz K, Welsh M, Marcillo O, Johnson J, Ruiz M, Lees J (2006b) Deploying a wireless sensor network on an active volcano. IEEE Internet Computing 10(2):18–25, DOI http://dx.doi.org/10.1109/MIC.2006.26
72. Werner-Allen G, Dawson-Haggerty S, Welsh M (2008) Lance: optimizing high-resolution signal collection in wireless sensor networks. In: Proceedings of the 6th ACM conference on Embedded network sensor systems (SenSys '08), pp 169–182, DOI http://doi.acm.org/10.1145/1460412.1460430
73. Whitehouse K (2002) The design of calamari: an ad-hoc localization system for sensor networks. Master's thesis, The University of California, Berkeley
74. Whitehouse K, Culler DE (2006) A robustness analysis of multi-hop ranging-based localization approximations. In: Proceedings of the Ffith ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN '06), pp 317–325

75. Whitehouse K, Jiang F, Karlof C, Woo A, Culler D (2004) Sensor field localization: a deployment and empirical analysis. Tech. rep., The University of California, Berkeley
76. Whitehouse K, Karlof C, Woo A, Jiang F, Culler DE (2005) The effects of ranging noise on multihop localization: an empirical study. In: Proceedings of the Fourth ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN '05), pp 73–80
77. Ye W, Heidemann J, Estrin D (2002) An energy-efficient MAC protocol for wireless sensor networks. In: IEEE Conference on Computer Communications (INFOCOM), URL citeseer.nj.nec.com/ye02energyefficient.html
78. Zhang J, Yan T, Stankovi JA, Son SH (2007) Thunder: towards practical, zero cost acoustic localization for outdoor wireless sensor networks. SIGMOBILE Mob Comput Commun Rev 11(1):15–28, DOI http://doi.acm.org/10.1145/1234822.1234827

Chapter 3
Designing for Deployment

Michael Allen, Geoffrey Challen, James Brusey, Lewis Girod, and Elena Gaura

Abstract This chapter introduces the *design for deployment* process, emphasizing the importance of gathering application requirements, applying a sound testing strategy and debriefing post-deployment. A set of key WSN design parameters are identified and explained in the context of the real deployments described in Part II. A case is made for deploying *iteratively*, as a method of progressively refining the system to best meet requirements. Finally, a selection of key lessons are drawn from the experience represented by the deployment case studies.

3.1 Introduction

Negotiating the design space for WSNs can be daunting even for the experienced designer. Apparently straightforward application requirements give rise to complex tensions within the design space and lead to a difficult optimization process in terms of hardware choices, software support, network topology, deployment planning, and maintenance strategy.

This chapter provides a starting point for designers engaging in the complex decision-making process towards a successful deployment. It starts, in Sect. 2, with a description of the stages of the *design for deployment* process, emphasizing the importance of certain stages such as gathering requirements, and testing at different levels of abstraction, along with the need to iterate over the process several times. Following this, Sect. 3 provides an analysis of several key design parameters in the context of the Part II chapters. Section 4 focuses on iteration, particularly deployment iteration, and shows how it was an important part of several successful deployments described in Part II. The final section provides a brief summary of key lessons drawn from these deployments. Scattered throughout the chapter, illustrative quotes from later chapters provide a link between the theoretical discussion

M. Allen (✉)
Singapore MIT Alliance for Research and Technology, S16-06-10, 3 Science Drive 2, Singapore 117543
e-mail: michael@smart.mit.edu

E. Gaura et al. (eds.), *Wireless Sensor Networks: Deployments and Design Frameworks*,
DOI 10.1007/978-1-4419-5834-1_3,

here and real deployment experience. Note that these quotes are provided as a taster; the associated chapter should be consulted for the full context.

3.2 The Design for Deployment Process

The following *design for deployment* process was derived partly drawing from the extensive experience represented by the later deployment chapters and partly by reflecting on the authors' collective experience with WSN development. Key differences between the design for deployment process and a more traditional software development life cycle are: the need for several distinct forms of testing; the focus on deployment and associated evaluative debrief; and the importance of iteration, particularly over the deployment phases.

As with other software, WSN development begins with gathering requirements and initial code implementation, continues with testing in more and more realistic environments, and finally produces a system to deploy. Figure 3.1 provides an overview of the *design for development* process. Lessons learned at each stage can be incorporated as the development process continues. Learning may also result in a return to an earlier point in the process if, for example, problems discovered during testing or deployment and/or consultation with the applications scientist or end-user produce a need to reimplement portions of the system or reconsider the design requirements. Iteration can continue until application requirements are met. Alternatively, iteration allows for new application goals to be incorporated over the

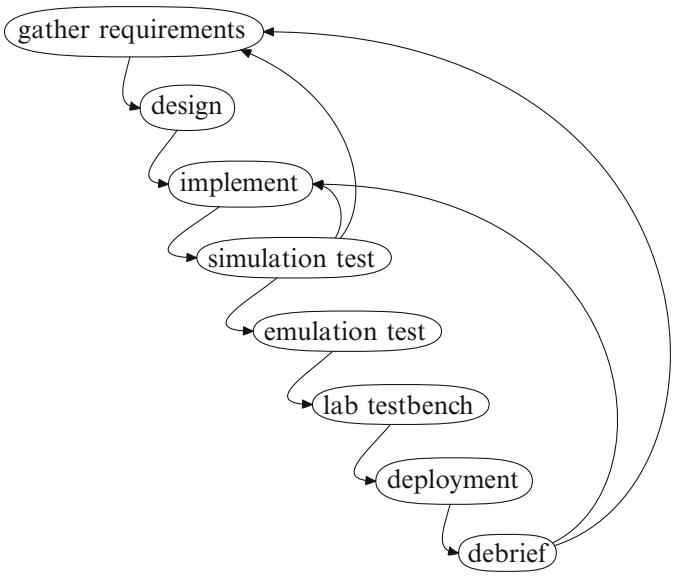


Fig. 3.1 The *design for deployment process* model: example including potential back loops

lifetime of the project, producing a lineage of systems meeting similar but different needs within the same application space.

While a great deal of attention is frequently paid to later stages of the *design for deployment* process, such as *implementation*, *testing* and *deployment*, carefully *gathering application requirements* early in the WSN design process is key to later success. This is demonstrated by several of the deployments discussed in Part II of the book; see particularly Chaps. 4, 9, and 10. It can also be particularly helpful to return to the application requirements following the development of the prototype system and initial deployment; at that point, developers and end-users alike have become more familiar with the capabilities of the technology and the challenges unique to the deployment environment will have made themselves apparent.

Post deployment, the debriefing stage evaluates how successful the deployment actually was. Perhaps data was gathered successfully but on analysis, the data is unusable. In some cases, the system may have performed poorly against requirements suggesting a redesign or changes to the implementation; in other cases, the problem may be the requirements themselves and they must therefore be revisited. In several of the later chapters, analysis of early deployment results led to devising additional systems and software to improve the quality of the data gathered (particular examples are Lance and Vigilance in Chaps. 4 and 6, respectively).

Since WSN systems tend to be based on display-less, low-power computers, it is extremely difficult to find software bugs or diagnose their cause, either in the laboratory or in the field, while that software is running on the target platform. As much as possible should be done to eliminate bugs, therefore, prior to running on the target and prior to deployment in the field. For example, communication *simulation* tools can be used to test protocols prior to implementation. Once implemented, an *emulation* environment or test harness allows further testing without the restrictions of the target platform.

Finally, a laboratory *testbed* provides a final testing stage before deployment, using the target platform (typically with multiple nodes). In the ExScal deployment (Chap. 8) several memory leaks and illegal access exceptions were only discovered at this final testing stage. In principle, however, bugs identified at later testing stages should trigger the generation of regression tests; that is, the emulation or simulation should be revised so that a similar problem would be found with less effort in the future.

> *A Stargate could achieve up to 44 kHz sampling rate, which was more than enough for our system. However, it could only process about 5% of the inputs sampled at 22 kHz in our initial implementation because of its slow floating point emulation.* – Chap. 7

The cost associated with testing and/or not discovering bugs until the system is deployed strongly encourages the use of off-the-shelf hardware and software since such systems tend to be more mature and more robust than custom developed ones. The SMART system described in Chap. 10 gives several examples of using commercial products instead of custom ones motivated in part by the stringent requirements of medical environments. The

GlacsWeb project in Chap. 9 uses custom hardware, due to the lack of a suitable off-the-shelf platform, but reduces the complexity of software debugging by mostly using high-level scripting languages.

In summary, a phased testing approach, with each phase increasing in realism, can help to ensure successful deployment first time. Using commercial or mature systems can help but does not necessarily ensure success. In the end, iterating the development process several times, possibly even revising the original requirements, may be necessary to develop a WSN system that can cope with the deployment environment and deliver the desired results. The process as a whole, and in particular, the emphasis on iteration is designed to respond to the difficult nature of deploying wireless sensor networks and to ensure that the end result meets the application goals.

The next section focuses on the design phase and in particular, points to several key design parameters that fundamentally affect the overall WSN design.

3.3 Key Design Parameters

The motivating application establishes what functionality is required of the WSN system as well as dictating the physical environment in which it will be deployed. The importance of formally specifying application requirements is evident throughout every chapter in this book – each of the chapters in Part II have concrete motivating applications, informed by domain specialists, that have led to successful systems design and deployment.

When considering how to respond to requirements, certain design parameters are particularly influential on the overall design and should thus be considered first. A selection of these key parameters are examined below.

3.3.1 Sampling Rate and Data Rate

One of the most important design parameters to consider is *data rate*. Roughly speaking, the data rate for any sensor node corresponds to the sampling rate times the bits per sample, although the exact relationship may be more complex if there are a variety of sensor types and not all sensors are sampled at once. Assuming a fixed available wireless network bandwidth and a certain number of nodes sharing the medium, the *transmitted* data rate for any node is limited. Therefore, beyond a certain data rate, it becomes impractical to transmit every bit that is sensed.

Instead, the data must be filtered, summarized, compressed, or somehow reduced prior to transmission. Processing on-node (or *in-network*) is often desirable because, even though processing power tends to be limited, the energy *saving* from transmitting smaller or less frequent packets typically outweighs the energy *cost* associated with performing the processing. The deployments in Chaps. 4, 5, and 7 all faced the problem of too high a data rate for the limited bandwidth available

(see below for the different methods applied). Note that a high sampling rate has other design consequences; for example, it means that the node will be awake more often, thus reducing its effective lifetime.

3.3.1.1 Dealing with a High Data Rate

> *In our case, the scientific objectives were known in advance. This allowed us to design a system based on intensive data processing at the sensor nodes, rather than data recording.* – Chap. 7

Applications that sample at a higher data rate than the available transmission bandwidth need to either increase bandwidth (by running cables, say) or reduce the amount of data transmitted. Data reduction techniques generally fall into three categories:

- *Event detection*, where transmissions are only made when something interesting occurs (e.g., VoxNet, Chap. 5);
- *Data compression*, where the data is compressed before transmitting (e.g., Cane Toad Monitoring, Chap. 7); and
- *Filtering*, where data is reduced to a summary before transmitting (e.g., Lance, Chap. 4).

In addition to transmitting the reduced versions, original data might be stored locally in case it is later required (e.g., Chaps. 4, and 5).

> *It is important not to lose data that may be useful to the domain expert – this is the reason that the spill to disk functionality exists.* – Chap. 5

Adding data reduction mechanisms usually complicates the design of a sensor system. The system architecture and software must be more complex and becomes more difficult to test. If the complete raw data set is not being saved for future analysis, the correctness of the data reduction critically affects the quality of the data gathered, and additional ground truth data must be collected in order to validate the system. In addition, processing *in-network* typically increases the complexity of the network protocols and topology. This is especially true of data reduction schemes that leverage sensing redundancy across multiple nodes (e.g., see Chap. 7).

When using event detection or filtering, it is important to understand both the phenomena and the behavior of the sensors. To create good filters and event detectors, it is usually necessary to first acquire a sizable sample data set – preferably based on the intended sensors, fitted as they would be in the final system. A prototype *sense-and-send* wireless system or even a data logger might be used to gather this data set. Extra sensors can be used at this stage to help validate the system and characterize the sensors. For example, a duplicate sensor could be used at the same location to check the accuracy of a base sensor.

3.3.2 Cost

The cost of a system is a key design parameter that guides the choice of hardware platform used. However it is not just the choice of hardware that influences the cost of a deployed WSN system. Instead, it is important to consider the whole lifetime costs including purchase, development, deployment, and maintenance of the final system. These costs need to be weighed against the value of the information obtained from the system.

Design choices may change where costs are incurred in the lifecycle of the system. The use of a commercial off-the-shelf system (or COTS) may significantly reduce development costs while possibly increasing purchase costs. In Chap. 10, for example, the sensing and wireless hardware used in patient health monitoring is required to meet electrical safety standards requirements and thus an additional cost factor for a custom system is the certification cost. In contrast, Chap. 7 gives the example of a network to monitor cane-toads composed entirely of Stargate nodes. This solution satisfied the application goals, but was too expensive to create sufficiently large, low-maintenance networks. Subsequently, a two-tier solution using cheaper motes was employed to increase network lifetime and reduce cost. There was a penalty to bear here, in terms of the development effort required to tune the algorithms to run on the more constrained platform, but this was offset against the reduced equipment cost.

> *Sadly, 2006 was the year when the whole of the front part of the glacier broke off and all of the equipment was lost.* – Chap. 9

When deployment is properly considered, cost becomes an important motivator to ensure that sufficient effort is spent on development and testing. With many real applications, deployment costs are high even if the installation is successful first time. Deployments often take place in remote locations (such as the volcano deployment in Ecuador in Chap. 4, or the glacier deployments in Norway and Iceland in Chap. 9) and installation may be laborious (consider the 1,000-node network deployment in Chap. 8). For this reason, it is important to put extra effort into development and testing to avoid failure at the deployment stage.

> *We completed marking all locations (983 XSMs and 203 XSSs) in 27 working hours with a team of 8 people. Laying out the equipment took longer: about 24 working hours with 14 people.* – Chap. 8

Finally, long-term WSN deployments will typically require periodic maintenance, the cost of which should be accounted for in the overall design. Common examples include battery replacement and repair of sensors and nodes, but more subtle examples might include obstruction mitigation, such as cutting foliage growth that has covered a solar panel or blocked communication paths. Some sensors, in particular reactive chemical sensors, have difficult maintenance requirements. For example, the water quality sensors used in a WSN project in Bangladesh required frequent recalibration (see Chap. 2).

> *... the cost of maintenance (which includes the monetary cost, as well as the cost of environmental disturbance) may be too high to pay.* – Chap. 6

The implications of a system's maintenance needs go beyond financial cost. The need for maintenance can greatly restrict the way in which sensors can be used in the context of long-term deployments, and in some cases may make a proposed system untenable. For deployments in remote or harsh environments, access to perform maintenance may be limited. Thus opportunities for maintenance need to be carefully exploited. For the glacier deployment discussed in Chap. 9, the glacier is physically inaccessible for the majority of the year and thus maintenance trips must be carefully planned. If any failure occurs, repair or replacement cannot be performed until the next maintenance period, risking the loss of valuable data in the interim.

Maintenance cost can become a fundamental design constraint in its own right. In Chap. 6, a maintenance support tool called Vigilance was developed to help minimize the data loss caused through environmental disturbance when sensors are repaired.

In summary, when considering the implications of cost as a design parameter, it is important to consider the costs occuring over the whole lifecycle. Also, costs should be weighed against the benefits brought by the system and the information it provides.

3.3.3 Network Size and Density

> *Rather than mask the presence of failed sensors, and the effect of missing data, tools are needed to both help compensate for the loss of data, as well as to estimate the amount of uncertainty introduced by missing data.* – Chap. 6

The third key parameter is network size and/or density. Large networks that employ large numbers of nodes tend to be based on small, low-power, inexpensive motes at the lowest tier. For large systems, it may be worthwhile to custom build rather than use COTS, in order to contain cost and to allow for integration of application specific sensors. Typically, use of inexpensive sensors leads to lower accuracy and resolution of the resulting data compared to that produced by more expensive options. Inexpensive sensors tend to be less reliable also. These problems can be offset somewhat by the use of *spatial redundancy*, where correlation in data from sensors close to one another can be exploited to improve the overall accuracy of the system. For large networks, particularly where density is high, it may be possible to use low-power radio transceivers and multi-hop protocols to reduce the energy requirements associated with transmitting data.

Both small and large network deployments are described in Part II. For example, project ExScal, presented in Chap. 8, is the first thousand-node network deployed in a real environment. Given the limited sensing range of each node, even a uniform arrangement of the nodes across the deployment area could not guarantee full coverage. This meant that the density had to be increased at the boundary of the network to initially detect intruders accurately, whilst providing coarser tracking within the network. Post deployment, the developers point out the importance of choosing sensors with wide individual coverage to contain sensor density needs.

The grid topology not only enabled us to provide coverage in a cost efficient manner but it also enabled us to design energy efficient data transport protocols, such as logical grid routing ... – Chap. 8

A contrasting limitation to sensing range that also affects network size and density is radio communication range. For example, the deployment density of sensor nodes for VoxNet in Chap. 5 is limited by the need to keep each node within range of two neighbors to ensure that time synchronization is highly accurate. In that case, given the radio range for the IEEE 802.11b radio used, a uniform arrangement with average inter-node spacing of 50 m is used to ensure this.

3.3.4 Deployment Environment

The deployment environment affects many aspects of the design of a successful system, from the choice of radio, through to the packaging of each mote, and is thus a key design parameter. Wireless communications is often heavily affected by environmental conditions, and most importantly, it is often difficult to assess the impact of the environment prior to deployment. Furthermore, even after a system has been deployed, there may be subsequent impact on communications as the environment changes over time, causing periods where wireless communication is disrupted.

For example, in populated environments, radios operating in the same frequency bands as sensor networks can cause unexpected losses in connectivity and data rate, as discussed in the short case study on the Peru seismic network in Chap. 2. In the glacier deployment described in Chap. 9, the effect of ice formation drastically affected the usable frequency spectrum for data transmission from buried nodes.

... only 33.7% of packets are delivered on average. The causes included unreliable wireless links and high degree of channel contention ... – Chap. 8

In addition to affecting wireless communication performance, the environment can also affect the placement of sensors. For example, in the soil sensor deployment described in Chap. 6, temperature sensors deployed below the soil could not be deployed at the chosen sites due to a layer of rock just below the surface. This emphasizes the

Reasons for sensors to become Byzantine were waving tall grass, extreme heat, or rain. – Chap. 8

importance of understanding the practical problems brought by the environment, and making sure the site is adequately surveyed prior to deployment.

Finally, packaging is often critical to ensure that sensors and processors are kept dry, well protected, and able to be tested, debugged, or replaced. It may take several deployment iterations, however, before the packaging fully meets all of these requirements. An example of this iterative process is given in Chap. 9.

3.3.5 Deployment Duration

The intended deployment duration can influence the type and number of batteries required by nodes, and the power conservation techniques that need to be employed. In short-term sensor network deployments, battery consumption does not have to be considered a primary concern. For example, in Chap. 5, the VoxNet platform is only required to be deployed for eight hours at a time, in an attended context. This means that nodes can operate continuously without power management considerations, sampling data at tens of kHz.

> *... batteries were exhausted, and various wires were frayed and broken, which led to either no measurements reported, or erroneous measurements, both of which produce a gap in the dataset.* – Chap. 6

However, in deployments that have target lifetimes of weeks, months or even years, it is important to have suitable strategies to prioritize data collection and conserve node energy. In Chap. 4, the deployment duration needs to be in the order of months. To enable this, the authors developed the Lance framework for data collection. Lance uses the intrinsic value of the data being collected, the current battery levels of the sensor nodes, and the known cost of transmitting the data in order to determine which data should be collected to optimize both the network lifetime and the application goals. In Chap. 9, the sensor network is also required to function for several months without maintenance, hence per-node data sampling rate is kept very low (order of minutes), and radio communication is kept to a minimum.

> *... the time window in which to monitor marmots meant that a non-trivial software bug causing a return to the accommodation could effectively hamper the day's work.* – Chap. 5

3.3.6 Target Audience and Interaction Model

The final design parameter considered here is that of the target audience and interaction model. Many of the deployed systems described in this book are aimed at domain scientists and are deployed by the original system developers. For these systems, usability has not been the focus. Rather the focus tends to be towards ensuring data quality.

> *It is critical that interactive use not impact any ongoing data collections that the system is tasked with.* – Chap. 6

As WSN technology becomes mainstream, applications will be aimed at non-expert users. For such systems, visualization of the system status and the data being gathered will become more important. The system should be robust enough to be able to deal with network formation, reconfiguration and data transfer in a way that is transparent to the user. For systems aimed at non-technical users, it becomes more important to design in features to support not simply deployment but also day-to-day operation, maintenance and reconfiguration.

In Chap. 5, for example, VoxNet's usage model expects the scientist to: (1) use the network as an interactive, exploratory tool and (2) be in close proximity to the network during deployment. VoxNet's design therefore must support on-line interaction, enabled through: (1) a control console, which allows the user to address the network from a central point, issuing commands and new software during the deployment, and (2) a spill to disk service, which logs all raw data gathered during deployment. Thus, the user could request to view the acquired raw data at the control console during the deployment and tune event detection parameters.

> *The involvement of a domain scientist during deployment was a critical part of the process.* – Chap. 5

To summarize, this section discussed several key parameters to consider when designing a real-world, deployable WSN application. The design process is complex, and it is often difficult to understand all of the issues affecting the design without going through a deployment cycle. In the next section, the importance of deploying iteratively – gradually improving the design based on deployment experience – is discussed further.

3.4 Iterative Deployment

> *... designing and building for a long time then deploying a perfect system was not feasible ... mainly due to the unknown nature of the environment.* – Chap. 9

Iteration is not just important during the design, implementation and testing phases of WSN development, but also central to successful deployment. As Fred Brooks once famously stated, "plan to throw one away; you will anyhow" [1]. Some form of failure in any early deployment iteration is to be expected. However much can be learned from these early deployments. Environmental effects can be more adequately assessed. Software bugs that only occur during deployment can be uncovered. Through deployment, both the system developers and the end-user gain a better understanding of the real requirements. Finally, a trial deployment can reveal much about ability of the sensors and sensor systems to accurately measure the phenomena under investigation.

Nonetheless, iterative deployment also brings a new set of challenges when compared with a single outing. While repeated deployment increases the probability of deployment success, experience suggests that success is not guaranteed. While it is usually possible, on each iteration, to solve problems that occurred previously, experience is so sparse and the domain so challenging that the chance of finding new problems on each iteration is still high.

Unfortunately, distinguishing between data that is faulty or simply unexpected is a challenge even in contexts where the environment is carefully characterized in advance. – Chap. 2

3.4.1 The First Deployment Iteration

The first iteration of a series of deployments is different from a single, unrepeated deployment. Sometimes a single deployment is so ambitious or lengthy that it is too difficult or costly to repeat. Both ExScal and SMART projects, described in Chaps. 8 and 10, respectively, present good examples of this type of deployment. In contrast, a first iteration deployment will tend to be lighter, faster, and with limited goals.

Although lab testing was useful in the early stages of development, it was the deployment at remote sites that forced us to think about what tools and features were still lacking in Hyper's design. – Chap. 6

Two basic goals that should be included are: to assess the challenges presented by the environment; and to check the requirements with the end-users. The system should be kept simple with a focus on quick solutions that have well-understood properties. Complex hardware or software is generally not desirable at this stage; first, because a complex solution may be more sophisticated than the application requires; and, second, because any increase in complexity increases the risk of bugs. Both the volcano and cane toad monitoring projects, explored in Chaps. 4 and 7, respectively, began with a simple proof-of-concept or pilot deployment and leveraged the lessons learned to design a more mature follow-on.

Based on our experience ... we were wary of reprogramming the network unless absolutely necessary. – Chap. 4

Prioritizing simplicity and fast development for a first deployment leads to preferences for off-the-shelf hardware, existing software components, and simple system design. Fast development is often essential because deployment schedules are pre-determined by the phenomena under investigation. Development of custom hardware and software should be avoided if there

Unlike a purely laboratory-based system, once the fieldwork logistics started in spring, the deadlines were fixed and unmovable. – Chap. 9

are existing solutions. Planning to "throw one away" may be a good strategy, but constructing a quick and simple version allows the replacement to be done in stages rather than en masse. This can significantly improve the design process for future deployments, since rather than starting from scratch each time, the last artifact can be used as a starting point and development confined to well-defined areas of the system.

Finally, speed and simplicity reduce the amount of effort invested in any one feature, making such a feature easier to abandon post-deployment if it proves unsuccessful, unimportant or uninteresting.

3.4.2 The Second Deployment Iteration

> *Most deployments have reported issues with data quality, and dealing with the resulting data loss or corruption is inevitable during the analysis phase of almost all deployments.* – Chap. 6

The second deployment iteration aims to resolve issues uncovered during the first deployment. In most cases, the first system was probably unable to meet the application requirements, and results and experience from the first deployment can be used to identify and address the most crippling weakness of the original system. The initial volcano monitoring prototype (Chap. 4) delivered data of a quality that did not meet the scientific requirements, resulting in a renewed focus on data quality during the design of the second system. Often, rapid reaction to exposed weaknesses can even occur during the deployment itself. The VoxNet project, described in Chap. 5, shows an example of a single deployment during which many important issues were addressed while in the field.

> *Starting with more powerful computing platforms allowed the development of robust detection and classification systems that were immediately useful to ...[the domain scientists]..., and then allowed us to focus on how to migrate such functionality into cost-effective low-power computing platforms.* – Chap. 7

The second deployment may be an ideal time to design and deploy hardware solutions more appropriate for, or tailored to, the specific application. This can slow the process of producing a second system but if done properly the hardware platform may be able to last for several deployment iterations, thus justifying the associated development cost.

Developers should avoid the temptation to begin completely anew and discard all of the software developed for the initial deployment. Instead, a form of triage is required; components or features will be found to be either: unrecoverable (needing to be scrapped), still feasible but needing attention, or meeting requirements

sufficiently well that they can be left alone. Methodical post-deployment assessment is critical to ensuring that future development efforts are not wasted.

Finally, in contrast with the first deployment, more effort is required to ensure that the resulting system can be evaluated in terms of meeting the application goals. Designing the evaluation strategy for a deployed system can be challenging and cannot be left until after the deployment is completed.

> *Post deployment analysis of system log data is a vital part of fixing or recreating problems in controlled environments [but] it is not always clear exactly what logging data will become useful during post analysis.* – Chap. 5

Debugging and logging tools become critical in this context. Building debugging infrastructure into the system early in the design process, as suggested above, speeds the development process initially and can also be critically important in the field as well. If properly designed, debugging tools can be used to record useful information about the system during the deployment, information that can both aid in debugging potential problems and support a post hoc evaluation of the system's performance after the deployment ends. The overhead for collecting log data needs to be considered as far as its effect on the other system goals, since transmitting status or other non-essential information over the radio during the deployment consumes energy and can reduce the system's lifetime.

3.4.3 Subsequent Deployments

Once a first proof-of-concept and second more substantial deployment have been completed, iterative deployment moves into a new, more stable phase. Continuing to deploy real systems can be an exciting opportunity if the engagement with the problem and understanding of the application deepens on each successive iteration. Multiple future iterations may also give the development team the opportunity to test various different approaches or sub-applications in a realistic context, meaning that each individual deployment can be tailored to a particular specific goal and perhaps not need to tackle as much at once as the second deployment.

However, continued iteration runs into challenges. If deployment experiences become repetitive and little new knowledge is gained then it may be difficult to justify the expense and stress of returning to the field. If the developers find that they have met the initial application requirements, system development can begin to isolate and iterate on other potential goals. Nodes can be made smaller or their cost reduced; the network lifetime can be extended; the size and extent of the network can be increased. Part II contains examples of systems that adjusted their expectations and goals once an initial system was judged successful. For example, the cane toad monitoring project described in Chap. 7 illustrates how a shift in focus can occur after initial project goals are met. The revised goal was to develop a new system that was more cost-effective by replacing portions of the original infrastructure with

the resource-poor, miniaturized devices. The volcano monitoring system described in Chap. 4 also proceeded from initial deployments that established the feasibility of the sensing platform into the development of a generic framework for optimizing high-resolution signal collection.

> *The experience from successive deployments provided clear lessons, which, if heeded, can drastically improve the performance and reliability of future systems.* – Chap. 9

While deployments can be exciting, motivating experiences for all involved, their cost has to be justified with regard to the overall project goals. Well-planned, well-executed deployments can provide insight that no laboratory experiment can match. Poorly-planned, sloppy deployments will consume resources without contributing much to the understanding of the application problem or WSN design in general. If each deployment is treated with the care and rigor due an expensive and time-consuming experiment, then these field experiences will inspire innovative, rewarding and successful WSN installations.

3.5 Lessons from the Field

The deployment case studies in the Part II chapters provide a number of lessons that can roughly be divided into: lessons about development, about performing a deployment, and about learning from the deployment experience. The most important and general lessons drawn from Part II are summarized here for the reader. This is intended to provide a focused checklist to consider when embarking on development for deployment.

3.5.1 Development Lessons

Try not to re-invent the wheel – use existing software and hardware where possible. When considering the software or hardware design, it is important to perform a survey of current solutions to determine what is currently available, and whether significant custom development is actually required.

If suitable existing components or solutions already exist and can be integrated, significant time and effort can be saved in the overall system development. As a note of caution, when considering commercially or freely available components, it is important to test their functionality experimentally, and ensure that the integration work does not outweigh the time and effort that would be expended in developing custom components to the same standard.

When developing custom components, there is a natural desire to develop generic electronics and systems that can be used to support multiple applications. However, high levels of optimization (for power consumption, resolution, accuracy, etc) and

genericity are goals often at odds with each other and create a design tension. Of the two, optimizing for a particular application is probably more manageable.

Simulate first. When developing and testing algorithms that will be used in a WSN system, it is vital they are verified with some form of simulation before deployment on actual hardware. This separates any platform-related implementation issues from any fundamental algorithmic problems. Simulation is usually speedy and cheap; at the preliminary/proofing stage, significant amounts of hardware do not have to be available.

On the other hand, be aware that accurate simulation is complex and that it is generally necessary to use a special language to describe algorithms. An alternative is to use some form of emulator or test harness. An emulator differs from a simulator in that it allows the final program code to be tested directly.

Build in support, to ease the deployment process. A basic concept when building a prototype circuit board is to include test points. The same idea applies to software: it is important to be able to monitor and verify the WSN functionality on-site, during deployment. A good example here is the Hyper routing service used in Chap. 6, which supports easy grafting into the network of a mobile sink, allowing the deployer to quickly verify that the data collection and transmission process is working as expected.

Visual cues are easy to implement and can be helpful in verifying that each node is operating as expected. For example, a status LED can be used to provide evidence to the deployer that the equipment is functioning, is correctly configured, and able to communicate with the network. For situations where the LED would be rarely used and the power budget is limited, a useful trick is to add a mechanical switch to turn off LEDs (as discussed in Chap. 9).

Instrument the system with logging for debugging and optimizing system performance. During the deployment lifetime, it is important to be able to characterize the system's performance and determine its weaknesses. Performance of the system could be monitored for example by logging data transfer times, times of event occurrences, battery levels, multi-hop routes as well as any debugging-related software output.

It is important to relate system logs to some global time source, so that the overall state of the network can be understood during post-hoc analysis. Problems relating to network protocols or transient environmental events become almost impossible to diagnose unless some form of local trace or log has been kept by each node and that entries in the log have a time-stamp that can be related to other node timestamps. A sophisticated time synchronization protocol may not be needed, of course. Sending a message to the base station with the local time offset may be sufficient.

Failure to recognize the importance of logging data with global timestamps can cause significant problems. For example, early deployments of VoxNet (Chap. 5) suffered from a lack of time-stamping in certain system logs, making it practically impossible to analyze the data traces after the deployment.

3.5.2 Deployment Lessons

Even deployments that appear to be simple will take a lot of planning and effort, and may require more time than expected. Each deployment presents new challenges that have little to do with the software and hardware development issues.

Allocate adequate time for deployment and actual system operation. It is important to build in some slack to the deployment schedule, in order to allow for emergencies and unexpected problems. It will take time to iron out any problems that occur in field, such as difficulties with the placement of sensors or delays in ensuring correct system operation. In Chap. 5, several problems arose in field due to equipment being unavailable for testing before deployment. This prompted several days of integration that ended up taking time away from in-situ deployment.

Prepare an equipment checklist and include spares. The one piece of laboratory equipment left behind (e.g., multi-meter, soldering iron, etc) will be the one needed. Spare components, nodes, cables, and laptops help to minimize the risk of a deployment failing due to a minor hardware fault.

Be prepared for devices and software, which were working in the laboratory, to malfunction in the field. An example of this sort of problem is shown in Chap. 9, where radios were not able to communicate over the expected distances due to the mountainous terrain.

Focus on data quality. Ensure that the data being gathered is *correct*. The sensor network's job is not done by simply sensing and delivering data – the developer must be concerned with the quality of the data that is coming out of the network, especially if the sensor network is being used as a science instrument, as many of the examples in this book are.

3.5.3 Learning from the Deployment Experience

The implication of iterative deployment is that improvement is sought with each new deployment. Two important, closely related lessons are given below, to be considered in the context of system refinement and iteration.

Optimize last and only optimize as needed. The primary goal for a deployed sensor network is to meet the base requirements of the application. Only after these requirements are met should the developer consider refinements that improve the efficiency, cost-effectiveness, or complexity of the system. Chaps. 4 and 7 provide several excellent examples of iteration being applied to meet different constraints after the basic application requirements were met.

Let the application requirements drive the iteration process. Most often, the data generated from the first (possibly exploratory) deployment of a system enables new observations, and points to processing or decision making that can be automated by the WSN system. Iterative development and deployment should be driven by the value added in meeting the application goals, and reflect the evolution of these goals as a result of the successful first iteration.

3.6 Summary

Applying Wireless Sensor Networks to real-world problems is both hugely exciting and greatly challenging. The challenges come partly from interfacing computers with the real world and partly from the uncertainties associated with distribution and wireless communication. Despite these challenges, as the case studies given in the rest of the book show, real-world WSN systems *are* possible and they *can* be both enormously rewarding and successful.

Reference

1. Brooks FP Jr (1995) The mythical man-month (anniversary ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA

Part II
Wireless Sensor Network Applications Case Studies

Chapter 4
Volcano Monitoring: Addressing Data Quality Through Iterative Deployment

Geoffrey Challen and Matt Welsh

Abstract Deploying wireless sensor networks to support geophysics presents an interesting challenge. High data-rates required by geophysical instrumentation preclude continuous data collection from even moderately-sized networks. However, geoscientists are used to working directly with complete signals, and therefore uncomfortable with in-network processing that could reduce bandwidth by reporting data products.

Over five years of working with seismologists we have developed a lineage of solutions driven by their scientific goals. Three field deployments have provided valuable lessons and helped drive each successive design iteration. We began by addressing datum quality, encompassing per-sample resolution, accuracy, and time synchronization. Later deployments focused on holistic data quality, which requires considering constraints limiting full data collection in order to maximize the value of the limited data retrieved. This chapter uses our three deployments to demonstrate the benefits of iteration. The first two illustrate our work on datum quality, while the last presents a new approach to optimizing overall dataset quality.

Keywords Volcano · Sensor network · Data quality · Optimization · Deployment · Time synchronization · Lance

4.1 Introduction

Beginning in 2004, computer scientists from Harvard University joined forces with seismologists from the University of New Hampshire, the University of North Carolina, and Instituto Geofísico, Escuela Politécnica Nacional, Ecuador, to begin a collaboration aimed at using sensor networks to further the study of active volcanoes. As of early 2009 our collaboration has spanned three successful deployments,

G. Challen (✉)
Harvard School of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02139, USA
e-mail: challen@eecs.harvard.edu

E. Gaura et al. (eds.), *Wireless Sensor Networks: Deployments and Design Frameworks*,
DOI 10.1007/978-1-4419-5834-1_4,

multiple scientific publications and generated a large number of interesting ideas to explore. We have been fortunate to be a part of this long-running partnership.

From a simple starting point – a handful of nodes streaming continuous data from a single sensor per node – we have developed a sophisticated resource-aware architecture carefully balancing the value of the data to the application against the network-wide cost of extraction. These design changes were motivated by the science goals, responsive to changing hardware platforms, and driven by experience gained deploying prior iterations. Because each of the three deployments is already well-documented, one of our goals is to illuminate the design process by linking successive artifacts together while maintaining the thread of data quality as an application driver.

From the beginning of our work, providing high quality data has remained a part of our research agenda. This focus emerged out of both the scientific goals, and constraints of the devices we have deployed. Because seismologists are used to processing high-resolution data from multiple stations, wireless sensor networks – while considerable less burdensome than existing instrumentation – must provide data of similar quality before they can be used for scientific study. The scale promised by augmenting existing seismological instrumentation with wireless sensor network hardware is new, but the existence of data processing techniques means that the requirements are already firmly in place.

Designing wireless sensor network applications in this space has required work to meet some of the data quality requirements while finding ways of creatively relaxing others. Specifically, we have found it necessary to deliver high-resolution data meeting strict timing requirements, but found flexibility in terms of providing a complete data set from every node covering all moments of time.

4.1.1 Overview of Seismoacoustic Monitoring

Volcanic monitoring has a wide range of goals, related to both scientific studies and hazard monitoring. Figure 4.1 displays an overview of several instruments that might be used, the signals that they collect and example configurations used during deployments. The type and configuration of the instrumentation depends on the goals of a particular study. Traditionally, dispersed networks of seismographs, which record ground-propagating elastic energy, are utilized to locate, determine the size of, and assess focal mechanisms (source motions) of earthquakes occurring within a volcanic edifice [2]. At least four spatially-distributed seismographs are required to constrain hypocentral (3D) source location and origin time of an earthquake, though using more seismic elements enhances hypocenter resolution and the understanding of source mechanisms. Understanding spatial and temporal changes in the character of volcanic earthquakes is essential for tracking volcanic activity, as well as predicting eruptions and paroxysmal events [9].

Another use of seismic networks is the imaging of the internal structure of a volcano through tomographic inversion. Earthquakes recorded by spatially-distributed

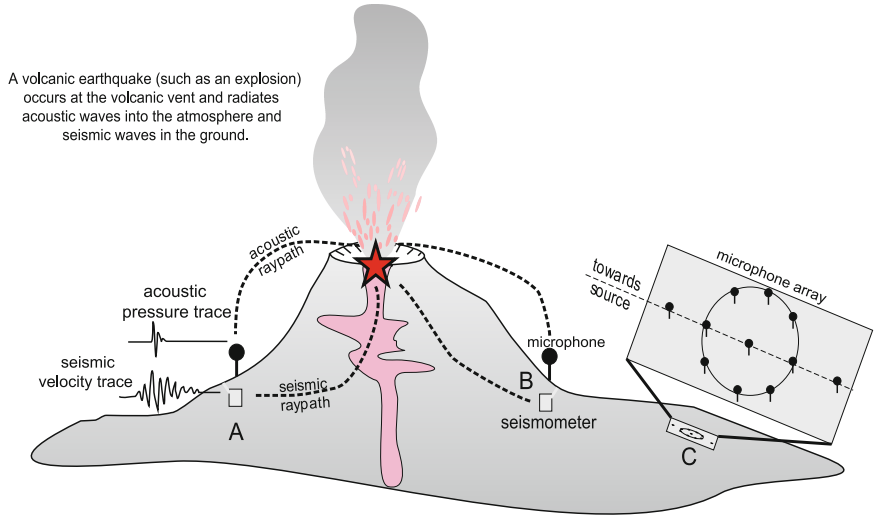


Fig. 4.1 Sensor arrays for volcanic monitoring

seismometers provide information about propagation velocities between a particular source and receiver. A seismically-active volcano thus allows for three-dimensional imaging of the volcano's velocity structure [1, 16]. The velocity structure can then be related to material properties of the volcano, which may be used to determine the existence of a magma chamber [6, 10]. Dense array configurations, with as many as several dozen seismographs, are also an important focus of volcanic research [3, 13]. Correlated seismic body and surface wave phases can be tracked as they cross the array elements, enabling particle motion and wavefield analysis, source back-azimuth calculations, and enhanced signal-to-noise recovery.

4.1.2 Opportunities for Wireless Sensor Networks

Networks of spatially-distributed sensors are commonly used to monitor volcanic activity, both for hazard monitoring and scientific research [17]. Typical types of sensing instruments include seismic, acoustic, GPS, tilt-meter, optical thermal, and gas flux. Volcanic sensors range from widely dispersed instrument networks to more confined sensor arrays. An individual sensor station could consist of a single sensor (e.g., seismometer or tilt sensor), or an array of several closely-spaced (10^2 to 10^3 m aperture) wired sensors, perhaps of different types. Multiple stations may be integrated into a larger network installed over an extended azimuthal distribution and radial distance (10^2 to 10^4 m) from the volcanic vent. Data from various stations may be either recorded continuously or as triggered events and the acquisition

bandwidth depends upon the specific data stream. For instance, seismic data is often acquired at 24-bit resolution at 100 Hz, while tilt data may be recorded with 12-bit resolution at 1 Hz or less.

Unfortunately, the number of deployed sensors at a given volcano is usually limited by a variety of factors, including: monetary expenses such as sensor, communication, and power costs; logistical concerns related to time and access issues; and archival and telemetry bandwidth constraints. Due to their small size, light weight, and relatively low cost, wireless sensor nodes have an important role to play in augmenting and extending existing seismic instrumentation, providing the increased spatial resolution necessary to support seismic applications like tomography.

Sensor data at a station may be recorded locally or transmitted over long-distance radio or telephone links to an observatory located tens of kilometers from the volcano. At the receiving site, data is displayed on revolving paper helicorders for rapid general interpretation and simultaneously digitized for further processing. However, due to the expense and bandwidth constraints of radio telemetry, high-quality, multi-channel data acquisition at a particular volcano is often limited. These analog systems also suffer from signal degradation and communication interference.

As a result, many scientific experiments use a stand-alone data acquisition system at each recording station. The digitizer performs high-resolution analog-to-digital conversion from the wired sensors and stores data on a hard drive or Compact Flash card. However, these systems are cumbersome, power hungry ($\approx$10 watts), and require data to be manually retrieved from the station prior to processing. Depending on the size of the recording media, a station may record several days or weeks' worth of data before it must be serviced. Deploying a wireless sensor network with telemetry to the base station allows real time data collection, network monitoring and retasking not possible with untelemetered systems.

4.1.3 Overview of Three Deployments

In total, we have performed three deployments of iterations of our system at active volcanoes in Ecuador. Figure 4.2 shows the location and layout of the second and third deployment. All three are summarized below:

1. *July, 2004, Volcán Tungurahua*: We deployed three infrasonic monitoring nodes continuously transmitting at 102 Hz to a central aggregator node, which relayed the data over a wireless link to the observatory approximately 9 km away. Our network was active from July 20–22, 2004, and collected over 54 h of infrasonic signals.
2. *August, 2005, Volcán Reventador*: This deployment featured a larger, more capable network consisting of sixteen nodes fitted with seismoacoustic sensors deployed in a 3 km linear array. Collected data was routed over a multi-hop network and over a long-distance radio link to a logging laptop located at the observatory 9 km away from deployment site. Over three weeks the network captured 230 volcanic events.

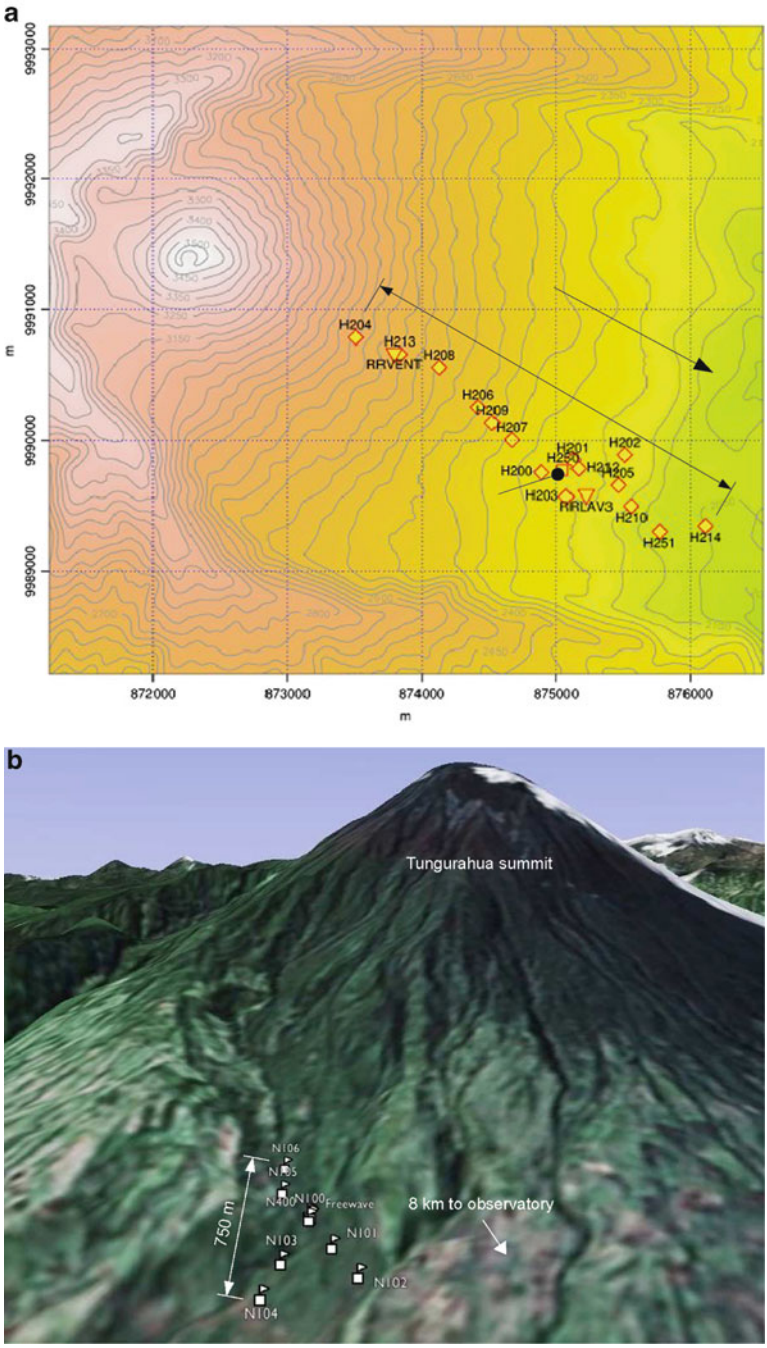


Fig. 4.2 *Deployment locations.* (**a**) and (**b**) show the locations of our second and third volcano deployments, in 2005 and 2007. (**a**) shows 17 nodes deployed on Reventador Volcano; (**b**) shows eight nodes deployed on Tungurahua Volcano

3. *July, 2007, Volcán Tungurahua*: We returned to Tungurahua Volcano in 2007 and deployed eight sensor nodes in order to test Lance, a framework for optimizing high-resolution signal collection (described Sect. 4.7). The network was operational for a total of 71 h, during which time we downloaded 77 MB of raw data.

4.1.4 Datum v Dataset Quality

Reviewing our previous work, we have found it useful to divide our focus on data quality into two separate concerns, *datum* quality and *dataset* quality. The former encompasses the quality of any one data point, and requires addressing sampling rates, resolution, fidelity, and accurate timestamping. Datum quality does not consider broader measures such as the value of the data to the application, coverage or latency. Such higher-level metrics are incorporated in the idea of *dataset* quality, which considers the entire corpus of data presented to the application or end user. This distinction is important because each requires separate techniques to address.

Placed in this context, our first two deployments served to push the datum quality to a level acceptable to the domain scientists. Through the experience that came with iteration, at the end we were able to convince ourselves that we had built a system capable of meeting the science *datum quality* goals [20]. Because the application we chose happened to have fairly well-established datum quality requirements, this work was iterating towards a fixed target.

Our interest in dataset quality reflects the inherent limitations of sensor network devices. Due to storage, bandwidth or power limitations, at some point as the size of the network grows or the target lifetime increases, it becomes infeasible to collect all signals from all nodes during the entire deployment. Thus the question emerges: what data should be collected in real time and what data should not? If the network is provisioned with adequate storage and the deployment is of fixed length, data not delivered in real-time may be eventually recovered manually; otherwise it will be lost.

Returning a partial dataset to the end user also requires sorting the wheat from the chaff. Either some of the data has to be interesting enough to justify eliding a great deal of the rest, or some of the data has to be uninteresting enough to justify dropping it entirely. Our application, volcano monitoring, falls more into the first category, since seismologists would never concede that any signal is, prima facie, uninteresting. However, they do have metrics allowing the value of a signal to be estimated and compared against others, which allows system resources to be directed at the most valuable signals. Because seismological applications may benefit sufficiently from the increased network resolution made possible by small, low-power sensors, the discarded data that these networks imply is tolerable.

The high sample rates and per-sample resolution of seismic signals meant that it only took a medium-size network to make full-signal streaming data collection infeasible. Given that the problem worsens as the network size grows and target

lifetimes increase, and our long-term goal is to deploy a perpetually-powered network of several hundred nodes – an order of magnitude larger than any of our efforts to date – addressing this problem remains central to our ongoing research.

4.1.5 Structure of this Chapter

We begin by describing the key changes made between our initial deployment at Tungurahua Volcano in 2004 and our second more significant effort at Reventador Volcano in 2005. Between these two deployments we made a large number of changes addressing both datum quality – including hardware board development and time synchronization rectification – and dataset quality – including reliable transport and event-driven data collection. We discuss iterative improvements to the interface board and time synchronization in Sects. 4.2 and 4.3 below, and outline the event triggered approach in Sect. 4.4.

After beginning to address the datum quality requirements, we focused on developing techniques that would advance the science goals by allowing the size of the network or the duration of our deployments to be increased. This meant addressing the dataset quality component. Our research in this area has been a great deal more speculative since our seismologists are not used to working with incomplete data sets, and while some of the flaws in earlier systems drove our initial research direction, work in this area became largely driven by the constraints inherent in wireless sensor network nodes: storage, bandwidth, and power. Our work in this area culminated in Lance [21], an architectural framework for optimizing dataset quality in the presence of constraints such as storage, bandwidth and energy. Sections 4.5 and 4.7 describe successive iterations of this framework, which emerged as part of our third deployment in 2007, while Sect. 4.6 describes the policy module component common to both.

Finally, addressing either datum or dataset quality in isolation, even while holding one of them fixed, does not address overall data quality or application-driven quality metrics. Collected data is expected to be put to use in service of some broader scientific goal, which likely has complex data quality requirements not easily distilled into separate datum and dataset goals. We outline future directions intended to bridge this gap and conclude in Sect. 4.8.

4.2 Sensor Interface Board

Of any system component, the hardware interface board received the most significant internal redesign on the way to our second deployment in 2005. Our proof-of-concept deployment deployed nodes fitted with a simple hardware interface board interfacing the Mica2 motes at use at the time with a single microphone. The Mica2's 10 bit ADCs were used to collect acoustic signals at 100 Hz, the

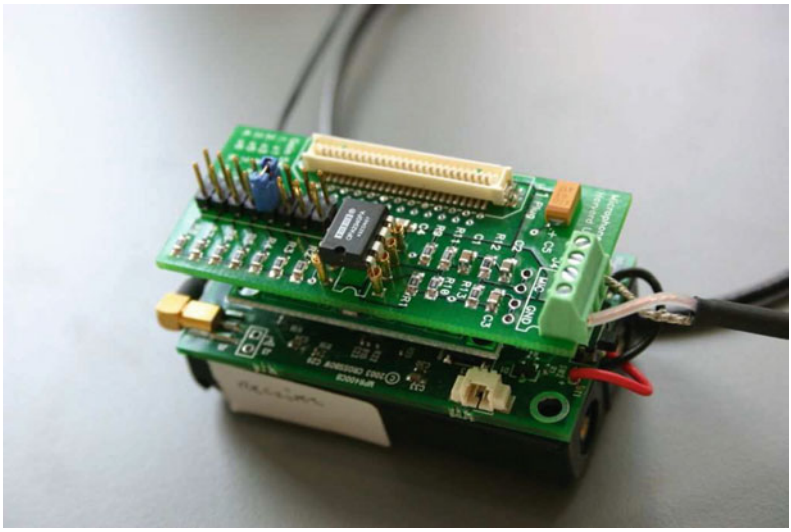


Fig. 4.3 *2004 Volcano Mote.* A Mica2 mote with simple sensor interface board allowing a single microphone to be attached as shown

sampling rate necessary to capture volcanic infrasound. Sampling was driven by the Mica2's onboard oscillator and the standard TinyOS `Timer` component. Figure 4.3 shows a Mica2 mote with the hardware interface board attached.

In particular, the following set of limitations of the original board had to be addressed in order to provide the datum quality required by the scientists:

1. *Resolution*: Scientific analysis required a per-sample resolution of at least 16 bits, with greater than 20 being ideal. The Mica2 motes used for the original deployment provided only 10 bits of resolution and later designs, such as the TelosB, provided only 16.
2. *Filtering*: The ideal filter for the seismic and infrasonic data our nodes were collecting is a hard cutoff at 50 Hz. Unfortunately, upon review the original sampling board had a passive filter with a 3 dB point of 5 Hz which meant that desirable components of the signal were being attenuated.
3. *Number of Channels*: Fitting each node with both seismometers and microphones, as we wanted to do for the second deployment, required allowing multiple sensors to be attached to each node. Our original board was built to interface a single sensor.
4. *Timing*: We expected that providing the high-fidelity timing necessary to track fast-moving pressure waves across the network would require both a stable timing source driving the per-node sampling and a way of interpreting local times across multiple nodes. Multi-hop time synchronization is discussed separately below, but the timing process necessitated a stable local source that would produce clean per-sample intervals.

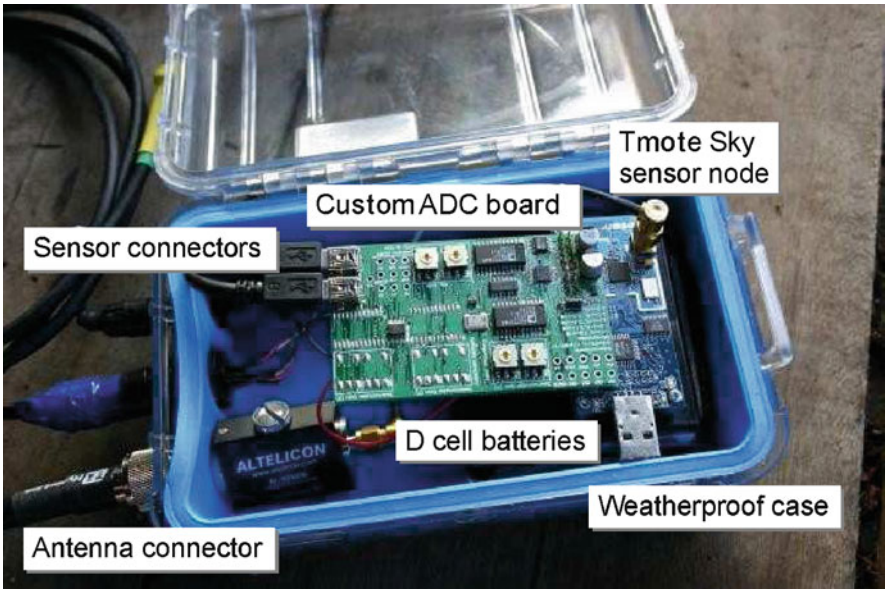


Fig. 4.4 The second iteration of our volcano monitoring sensor node, fielded at Reventador in 2005 and Tungurahua in 2007

5. *Isolation*: When testing our Tungurahua deployment before fielding it we noticed that, due to poor isolation between multiple chips on the Mica2, radio transmissions were interfering with signal collection. The large amount of current required to drive the CC1000 radio was flooding the ground plane and causing the onboard ADC voltage reference to shift. While our original system worked around this in software, providing clean data for scientists would require isolating the sampling components from sources of potential interference.

4.2.1 2005 Board Redesign

Initially, we considered using the TMote Sky's onboard ADCs. The TMote had a single 16 bit ADC – which we believed to be well-isolated from other circuitry on the node – and the ability to perform DMA sampling from the ADC into onboard RAM while leaving the CPU in a low-power state. However, several limitations of the TMote led us towards a standalone board. First, the 16 bit resolution of the onboard ADC was not quite enough to support our application. Had this been the sole limitation we may have settled for 16 bits, but a separate board meant we could select higher-resolution ADCs. In addition, the TMote had a *single* 16 bit ADC, and

we were worried about multiplexing this between multiple channels. Second, we were concerned with providing a stable filter centered at 50 Hz. Building a passive filter at that low of a frequency is difficult, so we concluded that the filtering would have to be done by an oversampling ADC. Performing the filtering on the TMote would have required oversampling at rates it was not capable of.

For these reasons we moved to a standalone sensor interface board providing its own reference voltage (ensuring isolation), ADCs (allowing us to choose the resolution and number of channels and implement digital filtering), and clock source (ensuring a highly-accurate sampling rate). Offloading this much functionality to a hardware board was risky since it left us little room for error in the design and fabrication, but it significantly simplified the code running on the nodes themselves. As it turned out, the fully built-out application we deployed at Reventador Volcano consumed almost all of the TMote Sky's 48 kB of program memory, leaving little room for the functions we offloaded to the interface board.

As deployed in 2005 we were quite happy with the performance of our external sensor interface board. The analysis in [20] revealed no significant deficiencies in the board design. There was some initial confusion about the sampling rate which, due to the precision of the oscillator and the settings on the ADC, turned out to be slightly less than exactly 100.000 Hz, but it was consistent across multiple nodes meaning that data collected from several stations could be lined up and processed together. We designed software allowing us to modify the ADC settings on-the-fly and this came in handy during the deployment when we realized that the hardware gain settings were too low. We were able to modify them in software and address the problem without returning to the deployment site.

4.2.2 Performance and Future Designs

The external interface board was not without its drawbacks. In particular, the oversampling ADCs we chose consumed a large amount of power, around 8 mA per ADC. When combined with other board components, such as the reference voltage, 3 to 5 V conversion necessary to power certain components, and external oscillator, the sensor interface board ended up consuming around 60 mA of current, or around 3 times more than the TMote with radio and CPU fully active. As mentioned previously, there is no way to power down the interface board remotely, nor would we be able to without immediately losing data since the ADCs are sampling at tens of kHz. For the Reventador deployment we provisioned around this high power consumption by deploying larger (D cell) batteries and replacing them several times. This can be seen as a tradeoff between datum quality – which necessitates the high-power external sensor interface board – and a reduction in dataset quality through shorter system lifetimes or higher-duty cycles due to increased power consumption.

It is likely that the next iteration of this board will take on several additional challenges. First, we will strive to lower the power consumption while maintaining

high fidelity through the use of newer ADCs which can hold down their current consumption even while performing the several factors of oversampling necessary to perform digital filtering. Second, we are increasingly interested in the ability to support multiple applications. Thus any future sensor board design, either built in-house or purchased as-is, will be expected to support multiple applications and sensor types. Our current board is, in its choice of ADCs and hardware-filtering, somewhat tailored to the volcano application. We'd like to move away from this if possible.

These two future goals are in many ways at odds with each other, since the challenge of reducing the power consumption of a board designed for a very specific purpose is different and potentially more manageable than the challenge of designing a general-purpose yet low power board. This is a design tension that we expect to continue to play out in future hardware revisions.

4.3 Time Synchronization

Unlike the sensor interface board, in which a simple prototype led directly to a more successful second effort, achieving high precision time synchronization between nodes is a battle we have continued to fight through multiple design iterations. Indeed, at present we are not yet certain a suitable solution has truly been found, or whether this challenge will emerge again while preparing the next deployment. The problem of accurate timing is one shared by multiple applications and deployment efforts, and significant effort has taken place in this area.

The need for accurate timestamping arises directly from the analyses performed on seismic and infrasonic volcano data, with the required precision dependent on the intended use and other aspects of datum quality such as the sampling rate. Seismic signals can move across a deployed array at thousands of meters per second, with acoustic signals moving at the speed of sound, (roughly 300 m/s). If two neighboring stations are deployed 100 m apart (possible with good line of sight and powerful antennas) a seismic signal can cross that gap in tens of milliseconds and an acoustic signal in hundreds of milliseconds. At a typical seismological sampling rate of 100 Hz this means that seismic wave arrival times at the two stations might only differ by one or two samples, necessitating precise time synchronization if the propagation of these waves is to be accurately captured. Thus our target accuracy for timestamping has typically been 10 milliseconds: a single sample interval.

Wired seismic instrumentation frequently deploys a single GPS receiver at each station, with the power required to operate GPS-driven timestamping a less significant component of the station's power budget. While we considered this approach (as described below), we ultimately rejected it due to its prohibitive power consumption.

4.3.1 Single-Hop Time Synchronization

Our first deployed system made use of a simple time synchronization approach appropriate in a single-hop environment. A single node was attached to a Garmin GPS "puck", which provides a highly-accurate (within 1 microsecond) pulse-per second output. This was trapped by an interrupt pin and, when triggered, that node sent out a broadcast radio message that should reach all other nodes. Upon receiving the message, each sampling node marked the sample that it was in the process of collecting. Beginning with this mapping between some of the samples (roughly one out of every 100) and the GPS per-second pulse, an accurate timestamp can be assigned to each sample via linear interpolation. (A more complete description of this protocol is contained in previously-published [18] work.)

This simple approach has many desirable properties, particularly when the two enemies of time synchronization in wireless sensor networks – skew and drift – are considered. Skew reflects the fact that all oscillators are not created equal, and that two oscillators sold as identical will, in fact, differ by some small amount. This is particularly true of the less expensive crystals used on low-cost wireless sensor network nodes. Drift names the tendency of the true rate of any particular oscillator to change over time, due to environmental changes such as temperature and humidity. Thus even if two oscillators start out perfectly synchronized changes to their local environments would cause them to drift apart slowly over time. Because the GPS PPS is guaranteed to be accurate and displays no drift, the accuracy of the broadcast message rate can be guaranteed. And, even in the presence of skew and drift on the receivers, as long as the drift rates are bounded the interpolation between neighboring PPS signals should yield accurate results. However, the single-hop approach obviously does not work in a multi-hop environment where the multiple sampling nodes cannot all hear the GPS broadcast.

4.3.2 Adaptation to Multi-Hop Using FTSP

During the three deployments we have performed, a single GPS node was deployed near a large power source (car battery) that also powered other pieces of critical infrastructure (the root of the spanning tree and long-distance point-to-point serial communication linking the deployment site to the volcano observatory). Provisioning multiple GPS nodes with the power necessary to enable acceptable system lifetimes would have greatly increased our deployment burden, and so a software solution was sought.

We ended up deploying a new wireless sensor network protocol called FTSP (Flooding Time-Synchronization Protocol) [8], which was released around the time that we completed our deployment at Tungurahua. FTSP allows a network of nodes deployed into a multi-hop topology to share a single global clock by providing

mappings between a local timestamp on any node and the global timebase. Our plan was to timestamp our data by performing two mappings: the first would map the local time on each node into the global FTSP time; the second would map the global FTSP time into the GPS time. We would rely on FTSP to perform the first mapping and deploy a single node with GPS to allow us to perform the second.

4.3.3 Observed FTSP Instabilities

During the testing that preceded our 2005 deployment at Reventador a number of problems were seen with FTSP in a lab setting. Sometimes the global time would become wildly inaccurate for a period of time before settling back to being quite accurate. We were unable to track down the source of this instability, although we did make multiple changes to the protocol in attempts to harden it and tailor it for our particular application. However, the faults we observed in the lab were all temporary in nature, and we believed that although FTSP did not seem to be always accurate it was stable and able to correct itself when it got off track.

The behavior we observed upon deploying the system was radically different. We did see, periodically, the small, correctable bits of instability that we had observed in the lab. However, we also noticed longer stretches of instability that seemed uncorrectable by FTSP itself. The only solution to rectify the timing on nodes that entered into this state was to reboot them, which forced them to resynchronize upon protocol restart. While we eventually built monitoring and automatic reboots into the system driver software running at the base station, allowing automatic reboots of nodes with troubled timing, the stretches of timing outages frustrated our attempts to collect clean, well-timestamped data.

Figure 4.5 shows an example of the FTSP instability observed in the field. The global time reported by two nodes suddenly jumps off by several hours, and the nodes do not resynchronize until rebooted 4.5 h later. It turns out that two bugs combined to cause this problem. First, it was discovered that the TinyOS clock driver would occasionally return bogus local timestamps.[1] Second, FTSP does not check the validity of synchronization messages, so a node reading an incorrect value for its local clock can corrupt the state of other nodes, throwing off the global time calculation.

The failures of the time synchronization protocol made establishing the correct GPS-based timestamp for each data sample extremely challenging. To do so, we developed a *time rectification* approach which filters and remaps recorded timestamps to accurately recover timing despite these failures. Figure 4.6 shows an overview of the process. The first step is to *filter* the global timestamps recorded by each node, discarding bogus data. Second, we build a model mapping the local time on each

[1] This bug was fixed in February 2006, several months after our deployment.

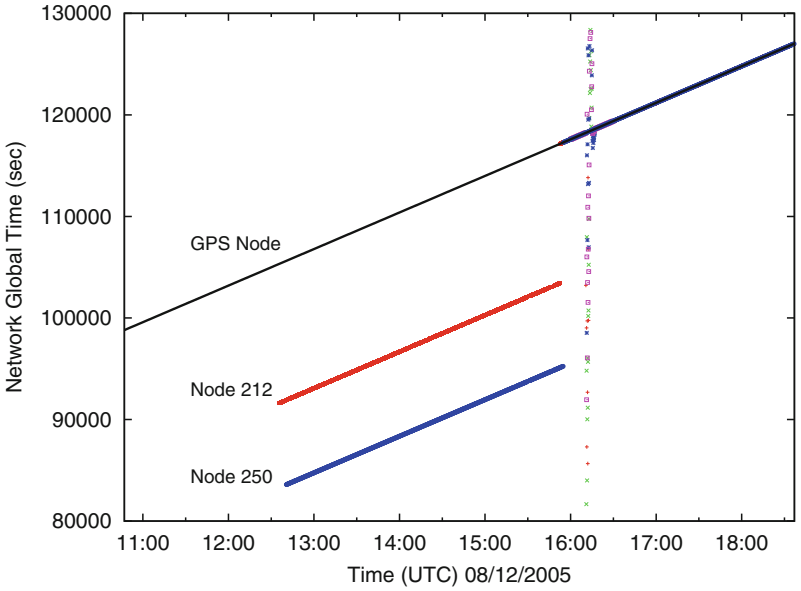


Fig. 4.5 *Example of FTSP instability observed during field deployment.* The global time value reported by sensor nodes and the GPS node is plotted against the time that the base station received the corresponding status messages. All nodes are initially synchronized, but starting at 1,230 GMT, nodes 212 and 250 report incorrect global times for the next 4.5 h. When the nodes eventually resynchronize, the global timestamps of other nodes initially experience some instability

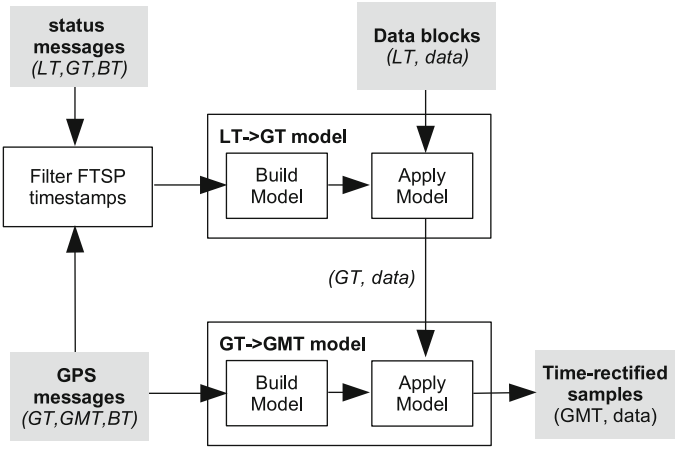


Fig. 4.6 Time rectification process overview

node to FTSP-based global time. Third, we use the GPS timestamp information to build a second model mapping FTSP time to GMT. Finally, both models are applied to the timestamps recorded in each data block producing a GMT time for each sample.

4.3.3.1 Timestamp Filtering

We begin by filtering out status messages appearing to contain incorrect global timestamps. To do this, we correlate global timestamps from each node against a common reference timebase and reject those that differ by more than some threshold. For this, we use the base station laptop's local time, which is *only* used for filtering FTSP timestamps, not for establishing the correct timing. The filtering process in is many ways similar to prior work [14, 15] on detecting adjustments in network-synchronized clocks.

We use the following abbreviations: *LT* is the local time of a node; *GT* is the FTSP global time; *BT* is the base station's local time; and *GMT* is the true GMT from the GPS signal. The single GPS node periodically sends a message logged by the base station consisting of the triple *(GT, GMT, BT)*. We use linear regression on this data to produce a reference timebase mapping *BT* to *GT*.[2] Nodes periodically report their status through a heartbeat message, which includes their local (LT) and global (GT) times, and for each node status message logged by the laptop *(LT, GT, BT)*, we map *BT* to the expected GT_{ref} using the reference timebase. If $|GT_{ref} - GT| > \delta$, we discard the status message from further consideration. We use a threshold of $\delta = 1$ sec. Although radio message propagation and delays on the base station can affect the *BT* for each status message, a small rejection threshold δ makes it unlikely that any truly incorrect FTSP timestamps pass the filter. Indeed, of the 7.8% of timestamps filtered out, the median *GT* error was 8.1 h.

4.3.3.2 Timestamp Rectification

The goal of *time rectification* is to assign a GMT timestamp to each sample in the recorded data. In order to do so, we build two models: one mapping a node's local time to global time, and another mapping global time to GMT. Figure 4.7 shows an example of this process bridging a small local timing instability of the kind described previously.

From those status messages that pass the filter, we build a piecewise linear model mapping *LT* to *GT* using a series of linear regressions. Models are constructed for each node separately, since local times vary significantly between nodes. Each regression spans up to 5 minutes of data and we initiate a new regression if the gap between subsequent *(LT, GT)* pairs exceeds 5 minutes. Each interval must contain at least two valid status messages to construct the model. We take the *LT* value stored in each data block and use this model to recover the corresponding *GT* value.

The next step is to map global time to GMT. Each of the GPS node's status messages contain a *(GT, GMT)* pair. As above, we build a piecewise linear model mapping *GT* to *GMT*, and apply this model to the *GT* values for each data block. Finally, we assign a GMT value to each sample contained in the block, using linear

[2] We assume that the global time reported by the GPS node is always correct; indeed, the definition of "global time" is the FTSP time reported by the GPS node.

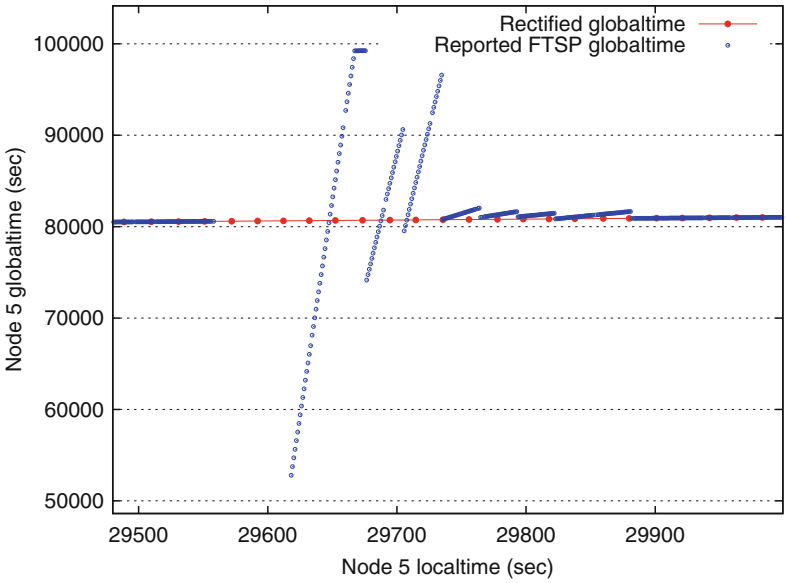


Fig. 4.7 *Time rectification example.* The raw (LT, GT) pairs collected from the node show that it experiences a period of FTSP instability. The time rectification process removes the errant timestamps creating an accurate mapping between LT and GT created using a linear regression on the remaining timestamps

interpolation between the GMT values assigned to the first sample in each block. This process makes no assumptions about sampling rate, which varies slightly from node to node due to clock drift.

4.3.4 Evaluation

Evaluating our time rectification process proved difficult, primarily because we had no ground truth for the timing of the signals recorded in the field. However, by reproducing the deployment conditions in the lab, we were able to measure the accuracy of the recovered timing in a controlled setting. In addition, as described earlier, two GPS-synchronized data loggers were colocated with our sensor network, providing us the opportunity to directly compare our time-rectified signals with those recorded by conventional instrumentation.

4.3.4.1 Lab Experiments

Our first validation took place in the lab. Feeding the output of a signal generator to both a miniature version of our sensor network and to a Reftek 130 data logger allowed us to directly compare the data between both systems. The miniature

	Raw error	Rectified error
1 hop, 50th percentile	1.52 ms	1.42 ms
1 hop, 90th percentile	9.86 ms	6.77 ms
6 hops, 50th percentile	2.63 ms	2.18 ms
6 hops, 90th percentile	13.5 ms	6.8 ms

Fig. 4.8 *Timestamp errors in a 6-hop lab testbed.* This table shows the 50th and 90th-percentile timing errors on both the raw FTSP timestamps, and rectified timestamps

network consisted of a single sensor node, routing gateway, and GPS receiver node. The same software was used as in the field deployment. The Reftek 130 logs data to a flash memory card and timestamps each sample using its own GPS receiver.

The results showed a consistent 15 ms offset between the time-rectified signals recorded by the sensor node and the Reftek data logger. We discovered that this offset was due to delays introduced by the digital filtering performed by the ADC on our sensor board (see Sect. 4.2.1). Adjusting for this delay resulted in an indiscernible offset between the sensor node and Reftek signals. While this experiment does not reproduce the full complexity of our deployed network, it does serve as a baseline for validation.

In the second lab experiment, we set up a network of 7 sensor nodes in a 6-hop linear topology. The topology is enforced by software, but all nodes are within radio range of each other, making it possible to stimulate all nodes simultaneously with a radio message. Each node samples data and sends status messages using the same software as the field deployment. The FTSP root node periodically transmits a beacon message. On reception of the beacon, each node records the FTSP global timestamp of the message reception time (note that reception of the beacon message is not limited by the software-induced topology). Because we expect all nodes to receive this message at the same instant (modulo interrupt latency jitter) we expect the FTSP time recorded at each node to be nearly identical. The FTSP root also records the time that the beacon was transmitted, accounting for MAC delay. The experiment ran for 34 h, during which time FTSP experienced instabilities similar to those seen during our deployment.

This allows us to compare the *true* global time of each beacon message transmission and the *apparent* global time on each receiving node, both before and after subjecting the data to our time rectification process. We call the difference between the true and apparent times the *timestamp error*. Figure 4.8 shows the results for nodes one and six hops away from the FTSP root. After rectification, 99.9% of the errors for the one-hop node and 93.1% of the errors for the six-hop node fall within our 10 ms error envelope.

4.3.4.2 Comparison with Broadband Station

Although time rectification works well in the laboratory, it is also necessary to evaluate its accuracy on the data collected during the field deployment. For this

purpose, we made use of one of the broadband seismometer stations colocated with our sensor network. The RVEN (for "Reventador vent") station was located 56 m from sensor node 213 (See Fig. 4.2b). Given their proximity, we would expect the seismic waveforms captured by both RVEN and node 213 to be well correlated. Some time shift between the two signals would be expected: a seismic wave passing each station could be as slow as 1.5 km/sec, so the time lag between the signals could be as high as 37 ms. However, due to differences in the seismometers and the placement and ground coupling of the sensors, we would not expect perfectly correlated signals in every case.

We identified 28 events recorded by both RVEN and node 213. The data for node 213 was time rectified as described earlier, and the RVEN data was timestamped by the Reftek's internal GPS receiver. We applied a bandpass filter of 6–8 Hz to each signal to reduce sensor-specific artifacts. The cross-correlation between the signals produces a set of of *lag times* indicating possible time shifts between the two signals. Due to the periodic nature of the signals, this results in several lag times at multiples of the dominant signal period. For each lag time, we visually inspected how well the time-shifted signals overlapped and picked the best match by hand.

Figure 4.9 shows an example of this process that demonstrates excellent correlation between the RVEN and node 213 signals with a 29 ms time shift. Figure 4.10 shows a scatter plot of the best lag times for all 28 events. Of these, only 5 events fall outside of a $+/-$ 47 ms window defined by the distance between the stations

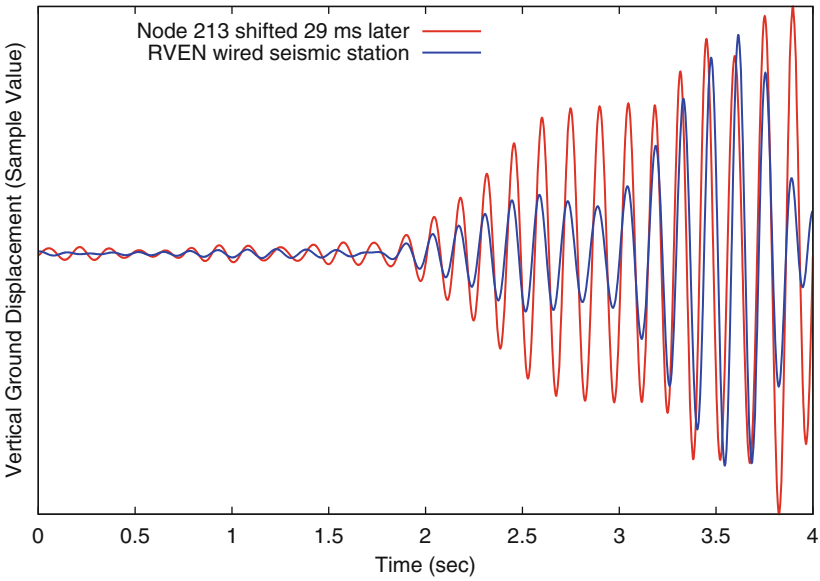


Fig. 4.9 *Comparison of RVEN and node 213 signals.* This figure shows two seismic waves recorded by sensor node 213 and a broadband seismometer located 56 m away. After time rectification, a 29 ms time shift produces an excellent match

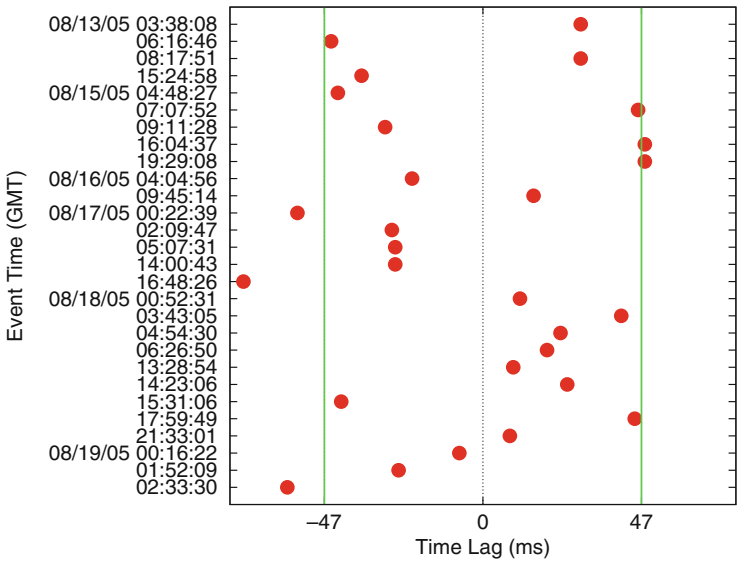


Fig. 4.10 *Lag times between Node 213 and RVEN.* The best lag time between the two stations is shown for 28 events. best time lag between the two stations is shown. Most time shifts into the $+/-$ 47 ms window that we would expect given the distance between the two stations and up to 10 ms of timing error

($+/-$ 37 ms) and our acceptable sampling error (10 ms). We have high confidence that our time rectification process was able to recover accurate timing despite failures of the FTSP protocol.

4.3.5 Lessons Learned

Overall there were many lessons we took away from our experience with data timestamping. First, due to the fact that development of the time rectification technique took several months after the system was dismantled, we missed a critical window during which the scientific interest in our data peaked. Arriving in the field with an end-to-end solution ready to take the output of our network and mold it into a form suitable for scientific inspection would have been ideal, and is highly advisable for deployments where scientist have clear datum quality expectations going in, as ours did.

In addition to being unprepared for the issues we observed in the field, we also did not think through the impacts of presenting half-baked data to the seismologists we were working with. Particularly given our worries about other parts of the system that did end up performing satisfactorily, like the sampling board and driver (Sect. 4.2), bulk data transfer protocol and event detection mechanism, we were excited when any signals at all showed up at the base station. In our rush to deploy

the system we had not prepared an adequate data analysis and visualization environment, and much of this was developed on-the-fly as the deployment progressed. Due to this late and rushed development, none of the initial tools did any of the post-hoc timing rectification that we eventually had to perform in order to make the data suitable for scientific study. Instead, we devised primitive tools allowing data from multiple stations to be plotted together.

Unfortunately, these stacked plots were interpreted very differently by the computer scientists and seismologists. To the computer scientists these were evidence that the network was sampling data, detecting events and successfully retrieving data using our bulk data-transfer protocol, namely things were *working*. Less concerned with the inner workings of our system the seismologists appreciated little of this. To them, these poorly-synchronized signals were instead evidence that our signal timing was extremely broken, a fear that persisted for some time after the deployment as we worked hard together to rectify and validate the timestamps. The exposure of this intermediate data product to consumers unprepared for it should serve as a cautionary tale about how much of the internal engineering of a deployed sensor network application to reveal externally.

Validating the timestamps on the data we did collect was also quite frustrating and illustrates to what degree we were unprepared for the severity of the timing challenge. The several wired data loggers deployed alongside our network all had attached GPS units providing extremely accurate signal timing. Had we thought to attach a single sensor both to one of our wireless stations and, by simply splitting the output signal, to one of the wired stations we could have easily cross-checked the timing with a known ground truth (i.e. the identical signal). In a remarkable oversight we never thought to do this, meaning that we had no two identical inputs available to reveal any inaccuracies in the timing information for the signals we collected. It would also have been useful to spend more time hardening FTSP before the deployment, and to have built in more logging and visibility into its operation to help us at least have an idea, in situ, of how well it was functioning.

While further work in this area would have been possible, our interests after the 2005 deployment tended more in the direction of improving dataset quality. Given our techniques to rectify the timestamps provided by FTSP were already in place going forward, little additional work was done in this area. In addition, FTSP has continued to be maintained and was canonized as an official mainline part of TinyOS in version 2.1.1 Given the importance of accurate timestamping to many sensor network applications, certainly ones in the scientific space, it is extremely encouraging that the sensor network community has indicated its willingness to continue to test and maintain this critical component.

4.4 Event Detection

While our proof-of-concept deployment had each node attempting to stream a continuous signal over a single radio hop to the base station, this approach did not scale to more nodes deployed into a multi-hop topology. In fact, it didn't even work that

well with the small number of nodes we deployed in 2004. Analysis of our data showed many dropouts and periods of missing data caused by simple packet delivery failures. Some nodes were impacted more than others, but all deployed nodes displayed this weakness.

Going into the 2005 deployment we knew that we needed a more scalable approach. The one that we developed was, in its own way, a precursor to the more intensive dataset work that would develop into a separate data-collection framework. What we decided to do was exploit the network's monitoring capability to help decide what data was interesting, and capture that data at the expense of other signals.

The way that this worked was as follows. Nodes were programmed to locally detect interesting seismic events and transmit event reports to the base station. If enough nodes triggered in a short time interval, the base station attempted to download the last 60 seconds of data from each node. The download window of 60 s was chosen to capture the bulk of the eruptive and earthquake events, although many volcanic events can exceed this window (sometimes lasting minutes or hours). To validate our network against existing scientific instrumentation, our network was designed for high-resolution signal collection rather than extensive in-network processing.

Nodes run an *event detection algorithm* that computes two exponentially-weighted moving averages (EWMA) over the input signal with different gain settings. When the ratio between the two EWMAs exceeds a threshold – indicating that the signal's short term average has exceeded its long-term average by a large amount – the node transmits an event report to the base station. If the base station receives triggers from 30 of the active nodes within a 10 s window, it considers the event to be well-correlated and initiates data collection.

The bulk-transfer phase operated as follows. The base station waits for 30 s following an event before iterating through all nodes in the network. The base sends each node a command to temporarily stop sampling, ensuring the event will not be overwritten by subsequent samples. For each of the 206 blocks in the 60 s window, the base sends a *block request* to the node. The node reads the requested block from flash and transmits the data as a series of 8 packets. After a short timeout the base will issue a repair request to fill in any missing packets from the block. Once all blocks have been received or a timeout occurs, the base station sends the node a command to resume sampling and proceeds to download data from the next node.

At Reventador this event detection approach proved difficult to properly calibrate. We discovered that it detected a small percentage of the seismic events located by one of our domain scientists during a particular window of time. The reason for this was probably the parameters chosen for the EWMA algorithm, which produced a system that was not sensitive enough. Calibrating the system a priori using data collected from the targeted volcano would have been ideal, but was difficult to do given the difference in frequency response between our instruments and the ones that had been deployed at the volcano previously. Other weaknesses of the event-triggered approach to data collection led us to move away from it in subsequent deployments.

4.5 Addressing Storage and Bandwidth Limitations

The simple event-based triggering we deployed at Reventador volcano in 2005 displayed a number of weaknesses that had to be addressed in order to build a more scalable system. In particular, these limitations led to significant loss of data and the overall approach would not have scaled as the size of the network or the lifetime target increased. After analyzing the performance of the 2005 network we identified three key weaknesses:

1. *Lack of Data Prioritization*: Our event-triggered system attempted to download data corresponding to well-correlated seismic events. However, the event detector operated in a binary fashion, leaving it unable to prefer certain events over others. Because we required each node to make an individual, local decision as to what constituted an "interesting" event, the efficacy of the system as a whole was largely dependent on this parameter. After analyzing the data collected at Reventador we determined that the original threshold was likely set far too low, causing our network to trigger on less than 5% of the actual seismic events observed by wired stations during the deployment. Without the ability to prioritize events setting a threshold means either risking underutilization if the threshold is too high or being unable to distinguish between extremely-interesting and less-interesting triggers if it is too low.
2. *FIFO Storage Management*: Due to the limited flash storage available on each TMote Sky, each node could only store around 20 minutes of continuous sensor data. When an interesting event occurred, to avoid overwriting data about to be requested for download we disabled sampling on each node until the data corresponding to the event in question had been downloaded. Due to the high download latency imposed by reliable transfer over multiple lossy links, and the high event frequency, a large portion of the network was offline for a significant amount of time after each triggered seismic event. This led to data loss.
3. *FIFO, Non-Preemptive Download Policy*: Following each triggered event we downloaded signals from each node in turn until the entire event was captured. As previously mentioned, this download process took a significant amount of time during which many nodes were not sampling. This meant that smaller, less interesting events could prevent the detection of larger, potentially more interesting ones if the small event occurred slightly earlier in time, since the network would still be busy downloading the small event when the large event occurred. Given that many large eruptions are preceded by small precursor earthquakes, this meant that many such large events failed to be recorded at all.

As an initial response to these challenges we developed a utility-driven architecture for optimizing the value of downloaded data in the face of storage and bandwidth constraints. Putting aside the datum quality issues this architecture attempts to address dataset quality directly by allowing the application to express preferences for some data over others and having the system attempt to maximize the value of the downloaded data while meeting constraints on storage, bandwidth,

energy and target lifetime. Based on experiences with earlier systems the design of a new system, Lance, was guided by several overarching design principles:

1. *Decouple mechanism from policy.* When possible we wanted to begin to abstract the development of our approaches to dataset problem away from the volcano application context, making Lance suitable for use in multiple application domains.
2. *Simplicity through centralized control.* As previously described, we had significantly simplified the design of our volcano-monitoring network by aggregating functionality on the base station. Lance continues this approach, again treating nodes as slave devices.
3. *Low overhead for maintenance traffic.* The drawback of a centralized solution can be high overhead for the associated control and maintenance traffic. We wanted to avoid this if possible by dividing decision making between local per-node components and global components running at the base station and optimizing the communication patterns between the two.

This architecture developed through two iterations, the second of which was published as Lance [21]. The first and second version of this system share a policy module architecture allowing applications to tailor the data collection strategy. The two iterations differ, however, in significant ways, with most of the changes resulting from our 2007 deployment at Tungurahua Volcano. The first system presents an *ordinal* (ordering-only) conception of utility, addresses storage and bandwidth as constraints, and attempts to maximize the value of the collected data without considering cost. The second moves to a *cardinal* (ordering and exchange) conception of utility, focuses on energy and bandwidth as constraints, and develops and deploys a cost model as part of the maximization process. This and the two sections that follow place these efforts into the broader context of an approach to dataset quality.

4.5.1 Overview of Lance

Figure 4.11 provides an overview of the Lance architecture. Sensor nodes sample sensor data, storing the data to local flash storage. Each application data unit (ADU) consists of some amount of raw sensor data, a unique *ADU identifier*, and a *timestamp* indicating the time that the first sample in the ADU was sampled. ADU timestamps can either be based on local clocks at each node, or tied to a global timebase using a time synchronization protocol such as FTSP [8]. The size of an ADU should be chosen to balance the granularity of data storage and download with the overhead for maintaining the per-ADU metadata. In the applications we have studied, an ADU stores several seconds or minutes of sensor data, not an individual sample. ADUs are stored locally in flash, which is treated as a circular buffer.

Ideally, nodes would be able to compute the value v_i of an ADU locally, as the data is sampled. However, since the value might depend on factors other than the ADU's data, such as data computed at other nodes, Lance assigns values v_i at the base station based on global knowledge of the state of the network. However, this

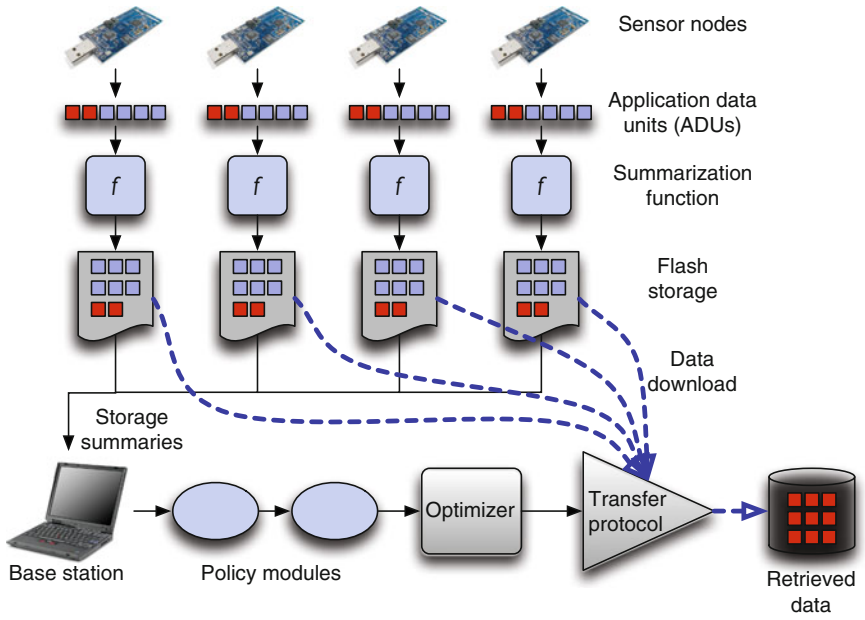


Fig. 4.11 *Lance architecture.* The architecture of both the priority- and utility-driven versions of Lance is quite similar. The interpretation of the assigned utility values differs, but the major architectural components are the same

requires nodes to communicate some low-bandwidth information on the ADU contents to the base station. For this purpose, each node applies an application-supplied *summarization function*, computing a concise summary s_i of the contents of the ADU as it is sampled. Nodes periodically send *ADU summary* messages to the base station, providing information on the ADUs they have sampled, their summaries, timestamps, and other metadata. As a special case, if a node is able to assign the ADU's initial value directly, this is used as the summary.

The Lance *controller* receives ADU summaries from the network. The controller also estimates the download cost $\bar{c}_i$ for each ADU, based on information on network topology as well as a model of energy consumption for download operations. The ADU summaries and cost are passed through a series of *policy modules*, which provide application-specific logic to assign the value v_i to each ADU. The resulting values are passed to the Lance *download manager* which is responsible for performing downloads, using a reliable data-collection protocol, such as Flush [5].

4.5.2 Cardinal v Ordinal Utilities

One significant difference between the first and second versions of Lance is the meaning of the utility values assigned to each ADU. Utility values are intended

to reflect the value of that signal to the application, with higher utilities indicating more valuable data. Accurate and efficient utility assignment – while a difficult and application-specific challenge – is crucial to the Lance approach.

The first version used *ordinal* utilities (or priorities), meaning that the assigned utilities established an order among available ADUs but did not reflect the relative importance of one ADU versus another. Given priorities, an ADU assigned priority 100 can only be assumed to me more valuable that one assigned an ADU 99. It might be 100 times more valuable. It might be only 2 times more valuable. The fact that ordinal utilities produce a strict ordering and nothing else simplified the policies of the first system. The storage manager could easily prioritize available flash on each node, and the download manager simply downloaded the available ADU with the highest utility, regardless of the resources necessary to do so. Since ordinal utilities do not allow the value of an ADU to be weighed against the cost (in system resources) required to download it, they render the notion of cost unnecessary.

The second iteration of Lance used *cardinal* utilities, which not only produce an ordering but also imply relative value between ADUs. So an ADU assigned utility 100 is assumed to be twice as valuable as one assigned utility 50. This formulation allows the incorporation of cost into the optimization framework, as described later in Sect. 4.7.

4.5.3 Utility Functions

Implicit in Lance is the idea that, when addressing dataset quality, wheat must be separated from chaff. We rely on a utility function to do so, regardless of whether the utility is interpreted as a cardinal or ordinal one. In general utility functions can be quite complex, depending not only on the data acquired at a particular node but also on the distribution of data across the network, availability of storage, bandwidth, energy and other resources, and so forth. To help designers construct complete systems *without* starting from scratch, Lance attempts to cleanly separate the *policies* governing how to divide data into interesting and uninteresting piles from the *mechanisms* governing the collection of data after the division is performed.

To leverage global knowledge while reducing the resource consumption on each node, we separate local knowledge from global by effectively dividing the utility assignments between node-specific (the utility calculator that runs on each node) and network-wide (the policy module chain described in the next section) components. The node-level utility calculator operates only based on inputs available on a single node and must run on a resource-constrained sensor network node, limiting the amount of computation it can perform. In contrast, the network-wide policy modules running on the powered base station have access to a global view of the network and significantly augmented resources. As such they can use global, historical and out-of-band information to further refine the initial utility assignment performed on the node itself.

Communication between the node-level and network-wide portions of the utility function is overhead with respect to the application goals, and so a practical requirement limiting the usage of Lance is keeping the traffic between these two components to a minimum. The utility functions we have developed in the context of the volcano application have typically limited this overhead by having the output of the node-level utility function be a single scalar value, which can be efficiently transmitted to the base station. There is no requirement that the node-level utility function produce such simple outputs, but it does aid in reducing the overhead inherent in the utility assignment process.

As a concrete example tying Lance to previous work, the distributed event detector deployed at Reventador in 2005 could be easily reimplemented in Lance. The node-level utility calculator uses the ratio of two EWMAs to assign a binary utility, either one or zero, to each ADU. At the network level, a policy module aggregates these assignments and, when enough overlap within a particular time window, it marks that entire segment of data with a unit utility value (all other time windows receive utility zero). Because each "interesting" set of signals receives the same utility value, Lance does not distinguish between them and will download them in a LIFO fashion.

4.5.4 2007 Deployment

To evaluate the performance of Lance in a real field setting, we undertook a one week deployment of eight sensor nodes at Tungurahua Volcano, Ecuador, in August 2007. The priority-driven version of Lance was used to manage the bandwidth resources of the sensor network, as described below. Time and budget constraints prevented us from deploying a larger network for longer period of time. Our primary goal was to validate Lance's operation in a field campaign, as well as to identify challenges that only arise in real deployments. Our experiences during this deployment led to further refinements of Lance described in the next section.

The sensor network was operational for a total of 71 h, out of which the Lance download manager ran for a total of 56 h. During this time, Lance successfully downloaded 1,232 ADUs, or 77 MB of raw data. An additional 308 downloads failed due to timeout or stale summary information, for an overall success rate of 80%. 11,012 unique ADU summaries were received from the network, representing an aggregate of 688 MB of sampled data. Lance therefore downloaded approximately 11% of the data produced by the network. Figure 4.12 summarizes the number of ADUs downloaded and the mean throughput for each node.

4.5.4.1 RSAM v EWMA Node Level Utility Calculator

The system as initially deployed computed the RSAM [12] (Reduced Seismic Amplitude Measurement) as the value for each ADU. This approach was intended

Fig. 4.12 Download performance during the deployment

Node	ADUs downloaded	Mean throughput
100	311	651.0 B/s
101	131	446.8 B/s
102	262	445.8 B/s
103	292	424.4 B/s
104	150	256.8 B/s
105	66	453.7 B/s
106	20	253.4 B/s
Total	1,232	431.5 B/s

to prioritize data based on the overall level of seismic activity. We experienced two problems as soon as the system was fielded. First, the RSAM calculation was sensitive to DC bias in the seismometer signal, causing Lance to generally prefer downloading ADUs from one or two nodes (those with the largest positive bias). We were able to work around this problem using *policy modules*, described in the next section.

The second problem with the RSAM utility calculator was caused by the uncharacteristically low level of seismicity at the volcano throughout the deployment. We observed only about 20 volcano-tectonic earthquakes and *no* clear explosions, whereas the previous week, Tungurahua exhibited dozens of earthquakes each day. As a result, the RSAM utility calculator was generally unable to distinguish between actual seismic activity and noise. Given the low level of volcanic activity, after the first 25 h of the deployment we chose to reprogram the network to use a different utility calculator designed to pick out small earthquakes from background noise. This function computes the maximum ratio of two EWMA filters over the seismic signal; it is similar to that described in [20]. Due to code size limitations on the motes, it was necessary to manually reprogram each node with the new utility calculator, which took two teams about 4 h.

We return to the 2007 deployment later in this chapter in two other contexts. First, we discuss *policy modules*, a useful feature that spanned both the priority- and utility-driven versions of Lance. We illustrate their use with examples from our field deployment. Finally we present the mature, utility-driven Lance, and also include an evaluation of its performance based on data collected during our 2007 deployment.

4.6 Policy Modules

Policy modules provide an interface through which applications can tune the operation of the download manager. As previously described, along with the node-level utility assignment they make up the utility function responsible for providing input to the download manager. Lance requires that policy modules be efficient in that they can process the stream of ADU summaries received from the network in real time. In practice this is not difficult to accomplish, as the rate of ADU summary

reception is modest, and the base station (typically a PC or laptop) is assumed to have adequate resources. For example, a 100-node network with an ADU size of 60 s would receive an ADU summary every 600 ms. Typical policy modules take a small fraction of this time to run.

One of the main benefits of policy modules is that they permit significant changes to the network's behavior *without requiring changes to the node-level utility assignment*. Changing the latter would typically involve reprogramming sensor nodes. Based on our experience at Reventador we were wary of reprogramming the network unless absolutely necessary. Although systems such as Deluge [4] permit over-the-air reprogramming, any changes to the sensor node software could result in unexpected failures that can be very difficult to debug without manual intervention. On the other hand, introducing new policy modules at the base station is relatively straightforward, and can be quickly reversed without risking sensor node failures.

4.6.1 Example Policy Modules

Policy modules can be used to encapsulate a wide range of data collection goals, and make it easy to customize Lance's behavior for specific applications. While we provide a standard toolkit of general-purpose policy modules, application developers are free to implement their own modules as well. By composing modules in a linear chain, it is easy to implement various behaviors without requiring a general-purpose "policy language."

Priority thresholding: `filter` is perhaps the simplest example of a policy module that filters out ADUs with a utility below a given threshold T. This type of filtering can be used to force a drop of low-valued data. Conversely, the `boost` utility module sets the utility for an ADU above a given threshold to a very high value, ensuring that it will be downloaded next.

Priority adjustment and noise removal: Policy modules can be used to remove the effects of noise or correct for node-level utility bias, for example, based on poor sensor calibration or differences in site response. Moreover, since each node computes the initial ADU utility based only on local sensor data, it may be necessary to normalize the ADU utilities in order to compare utilities across nodes.

`adjust` adds or subtracts a node-specific offset to each ADU utility in order to correct for differences in sensor calibration. `smooth` applies a simple low-pass filter on the raw utility values to remove spikes caused by spurious sensor noise. Likewise, `debias` is intended to remove sensor-specific DC bias in the utility values assigned by the node's prioritization function. `debias` computes the median utility value for a given node over a given time window. It then subtracts the median from each ADU utility before passing it along to the next module in the chain.

Likewise, when a sensor network contains multiple sensors with varying sensitivity, it is natural to prioritize data from more sensitive instruments. In cases where networks are deployed to monitor fixed physical phenomena, it may be desirable

to prioritize data from nodes located close to the phenomena being observed. The `adjust` module can be used to scale raw utilities based on a sensor's location, signal-to-noise ratio, or other attributes.

Priority dilation: Another useful policy is to dilate a high (or low) utility value observed in one ADU across different ADUs sampled at different times or different nodes. This can be used to achieve greater spatial or temporal coverage of an interesting signal observed at one or more nodes. The `timespread` detects ADUs with a utility above some threshold T, and assigns the same utility to those ADUs sampled just before and just after.

Likewise, the `spacespread` module groups ADUs from across multiple nodes into time windows and assigns the maximum utility value to all ADUs in that window. Define a window $W(t, \delta)$ as the set of ADUs such that $t - \delta \leq t_i \leq t + \delta$ where t represents the center of the window and δ the window size. `spacespread` determines the maximum ADU in the window $p^* = \arg_{i \in W} \max p_i$ and sets $p'_i = p^*$ for each ADU in W.

Correlated event detection: The `correlated` module is used to select ADUs that appear to represent a correlated event observed across the entire sensor network. `correlated` counts the number of ADUs within a time window $W(t, \delta)$ with a nonzero utility value. If at least k ADUs meet this criterion, we assume that there is a correlated stimulus, and the utility values for all ADUs in the set are passed through. Otherwise, we filter out the ADUs in the window by setting $p'_i = 0$ for each ADU in W.

As an example of composing policy modules to implement an interesting behavior, consider the chain

$$\mathit{filter}(T) \rightarrow \mathit{correlated}(k) \rightarrow \mathit{spacespread}$$

This policy filters incoming utilities, rejects time-correlated sets with fewer than k ADUs above the threshold, and assigns the maximum utility across the set to all ADUs. This can be useful in systems that wish to perform collection of time-correlated data, but avoid spurious high-utility data from just a few nodes. This policy is equivalent to the volcano earthquake detector used in our previous work [20], expressed as a simple policy module chain, and demonstrates, as mentioned in the previous section, that our distributed event detector can be reimplemented as a Lance utility function with both node- and network-level components.

4.6.2 Evaluation and Use at Tungurahua

We evaluated the usefulness of the policy module architecture through testbed experiments, as well as during our field deployment in 2007. For the testbed experiment, we use a distribution of ADU data values based on a 6-h seismic trace collected at Reventador Volcano, Ecuador in 2005 [20]. The raw seismic data is divided into ADUs of 36 kB and ADU values v_i are assigned by computing the ratio of two EWMA filters on the data, which assigns greater value to ADUs that contain

earthquakes. For each node in a 25 node topology, the ADU values from the seismic trace are attenuated based on a hypothetical signal source and assigned to each of the 25 nodes based on their location with respect to the signal source. We then enable a policy module chain that assigns higher priority to ADUs that correspond to correlated seismic activity across the network.

Figure 4.13 shows the result of this experiment running on the MoteLab testbed. The upper portion of the figure shows the ADU values over time; the middle portion,

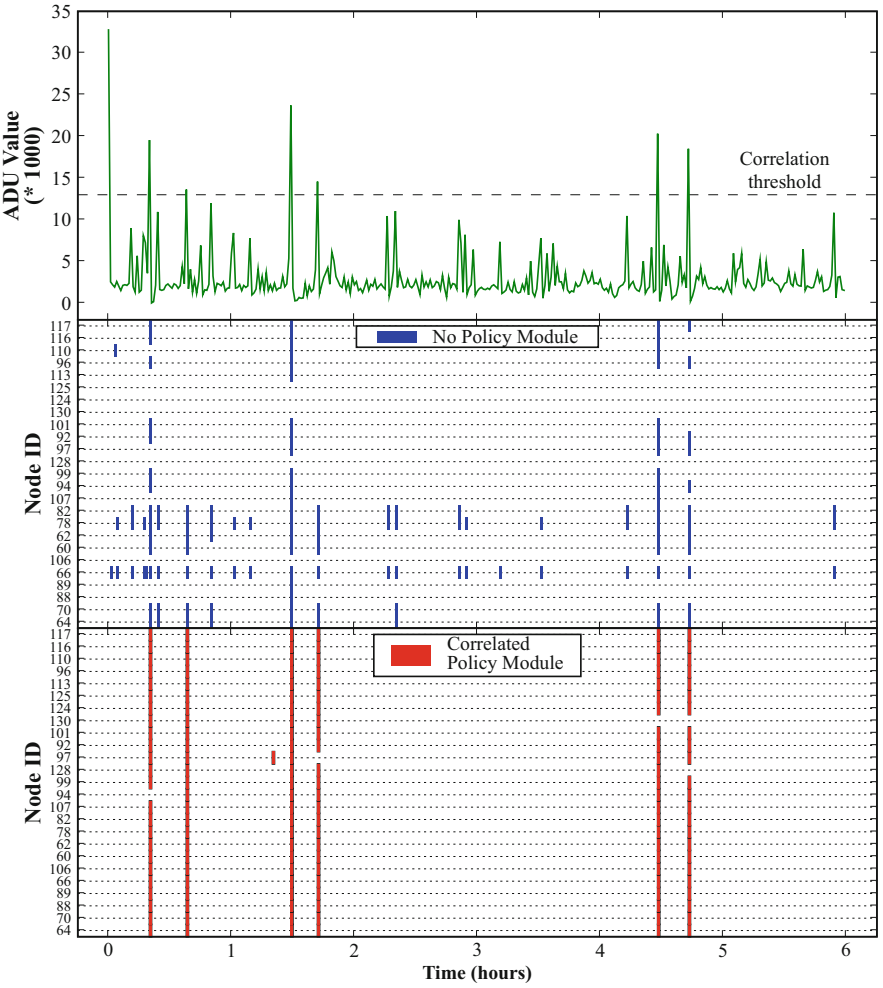


Fig. 4.13 *Usage of policy modules to affect download distribution.* Here we illustrate the use of policy modules. The graph compares the download behavior of the system with and without a policy module chain which assigns greater values to ADUs corresponding to correlated seismic activity. The graph is colored at a particular timestamp and node ID if we downloaded that signal from that node. The top graph shows the ADU values over time, with the threshold for the `filter` component of the policy module chain indicated

the set of ADUs downloaded by the system with no policy modules in use; and the lower portion, the ADUs downloaded with the policy module chain in use. As the figure shows, the policy modules cause the network to prefer correlated seismic events and download an ADU from all nodes in the network when such an event is detected. Gaps in the set of ADUs downloaded are due to download timeouts. In one case, a single ADU is downloaded spuriously due to an incorrect value being reported by that node to the base station. This use of policy modules shows the drastic change in the system behavior that can be achieved without reprogramming the sensor nodes themselves.

Our deployment at Tungurahua Volcano allowed us to evaluate the ability to change download policies at the base station without reprogramming nodes, one of the significant advantages policy modules provide. As described in Sect. 4.5.4.1, the RSAM utility calculator initially deployed at Tungurahua was sensitive to DC bias, which caused Lance to continuously download signals from one or two nodes with large DC biases.

This problem was easily corrected, without any node software changes, by introducing a policy module at the base station to process the raw RSAM values received from each node and filter out the DC bias. This was achieved by computing the median RSAM value over each 30-minute window of raw RSAM values on each node, and subtracting the median from the RSAM. Figure 4.14 shows the result of the application of the filter, with the debiasing effect clearly visible. The ability to change and correct download behavior from the base station without modifying sensor node code proved extremely useful in this case, particularly in that it permitted the rapid iteration necessary to properly craft the appropriate filter.

4.7 Optimizing for Energy and Bandwidth Usage

After completing our 2007 deployment at Tungurahua we redesigned our original Lance system and shifted its focus. Instead of optimizing for storage and bandwidth, we instead chose to optimize primarily for energy, with bandwidth as a secondary concern. Several changes to the hardware and software environment explain this shift in emphasis.

First, storage management began to seem less pressing as the promise of sensor network nodes with large attached flash memories seemed to be becoming a reality. The TMote Sky deployed at Reventador had only 1 Mbit of flash, meaning we could only store around 20 min of signal (from two channels, 24 bits per sample at 100 Hz). In contrast, the SHIMMER mote developed for medical monitoring can be fitted with a flash card allowing it to store as much as 2 GB of data, meaning that deployed in support of the volcano application it could conceivable store over 41 *days* of full-resolution sampled data (assuming 2 data channels sampled at 100 Hz with 3 bytes per sample). The idea behind managing storage was that it was necessary in order to preserve the highest-utility ADUs until the network controller would get around to

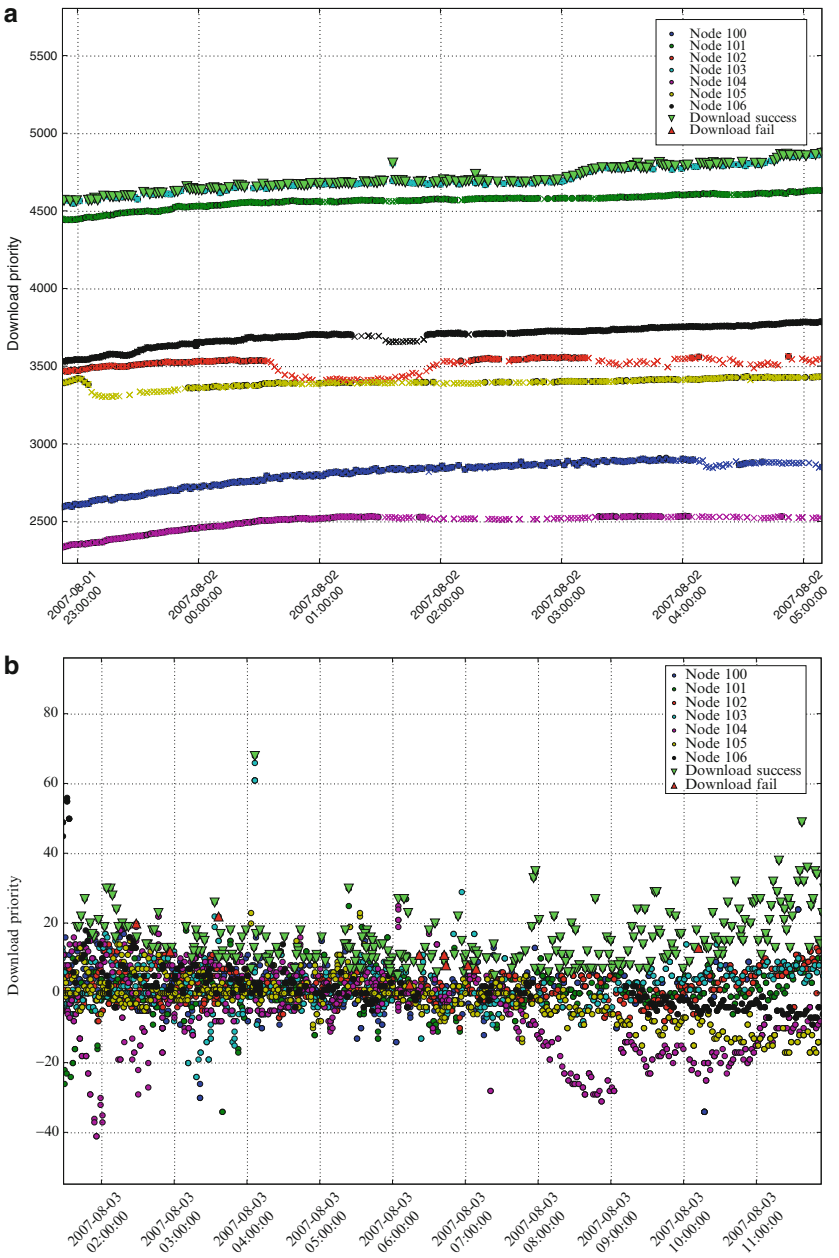


Fig. 4.14 *Effect of DC bias on RSAM summarization function.* Each point represents the ADU value received at the base station, and the triangles indicate those ADUs that were downloaded by Lance. In (**a**), because nodes' RSAM values are offset significantly from each other, Lance prefers downloading from the node with the largest positive bias. However, (**b**) shows what happens after applying a simple debiasing policy module at the base station. This filter removes the DC bias causing Lance to download from multiple nodes

downloading them. Once storage sizes swelled it seemed like a simple FIFO storage policy would allow plenty of time for interesting data to be downloaded.

Second, support for radio duty cycling entered the TinyOS tree and began to be used by our group. This shifted our focus from thinking about bandwidth as a constraint to thinking about the *energy* associated with operating the radio in order to download data as a constraint. In contrast to the early Lance system, which deployed a "greedy" download manager which strove to download as much data from the network as possible, once the radio could be easily disabled when not in use it became natural to want to consider the cost of downloading signals and not necessarily run the downloads in a greedy fashion.

4.7.1 Refocusing on Energy Usage

Shifting the system's concern to energy usage required several changes to the Lance architecture: a move from ordinal to cardinal utilities, and the development of a cost model.

Replacing the ordinal utilities with cardinal ones allowed us to compare the relative value of different "baskets" of ADUs in a meaningful way. Two ADUs, each with a utility of 50 were considered equivalent to the application as a single one with utility 100. This meant that instead of simply downloading the highest-utility ADU available in the network at a given moment, we needed a way of determining which was the best ADU in terms of both the value but also the system's ability to extract it.

Given our high data rates, we were already restricted by bandwidth alone to downloading only a portion of all of the data sampled by every node in the network. Once we began using low-power listening (LPL) [11] to duty cycle the radio extracting data also had an impact on energy consumption as well. Thus, achieving a target lifetime could be accomplished by downloading data at slower rates, allowing node radios to be powered off when not in use. Either bandwidth or energy could have driven the cost model, but given the dependence of energy consumption on bandwidth usage (and lack of a similar relationship in the other direction) we chose to represent the cost necessary to download an ADU as the *energy* required to retrieve it.

Compared with our original ordinal utility efforts, the new version of Lance contributed several ideas. First was a way of estimating, a priori, the energy necessary to extract an ADU from the network. This was necessary so that both the cost and value of each ADU could be considered before download decisions were made. For value, we leverage the same two-part utility-assignment framework described earlier. The cost estimation component is described below.

In addition, once the cost and value had been assigned to each ADU, the problem of deciding which to download can be framed as an optimization problem. We described a way of producing an optimality bound by mapping an offline version of

this problem to a well-studied optimization problem, and use this optimality bound to evaluate several online algorithms suitable for real-time decision making. These contributions are also described in further detail below.

4.7.2 Cost Estimation

Each ADU has an associated cost $\bar{c}_i$ that represents the energy requirement to download the ADU from the network. $\bar{c}_i$ is a vector $\{c_i^1, c_i^2, \ldots, c_i^n\}$ where c_i^j represents the estimated energy expenditure of node j when ADU i is retrieved. The key idea is that we explicitly model both the energy cost for downloading the ADU from its "host" node n_i and the energy cost for each node along the routing path from n_i to the base station which must forward packets during the transfer. In addition, we also model the energy cost to nodes that overhear transmissions by nodes participating in the transfer. This energy cost on intermediate nodes is non-negligible, since reliable transfer protocols involve a potentially large number of retransmission. However, the overhearing cost is typically small, since modern low-power MAC protocols quickly return to sleep when overhearing transmissions to another node. The cost vector $\bar{c}_i$ therefore depends on the network topology.

Lance estimates the download energy cost vector $\bar{c}_i$ for each ADU sampled by the network. We assume that nodes are organized into a spanning tree topology rooted at the base station. The cost is a function of many factors, including the reliable transport protocol, each node's position in the routing tree, radio link quality characteristics, and the MAC protocol.

Given the complex dynamics that can arise during a sensor network's operation, we opt to use a simple conservative estimate of the energy cost to download an ADU from a node. Our approach is based on an empirical model that captures three primitive energy costs involved in downloading an ADU. The first, E_d, represents the energy used to download an ADU from a given node which includes the energy cost for reading data from flash and sending multiple radio packets (including any retransmissions) to the next hop in the routing tree. The second, E_r, represents the energy cost at intermediate nodes to forward messages during the ADU transfer. The third, E_o, represents the energy cost to nodes that overhear transmissions during a transfer. For simplicity, we assume ADUs of fixed size and compute E_d, E_r, and E_o based on the time necessary to download an ADU from the target node.

Using this simple model, we set the elements of the cost vector $\bar{c}_i$ as follows. $c_i^n = E_d$ for the node n hosting the ADU, and $c_i^m = E_r$ for nodes m along the routing path from n to the base station. We set $c_i^o = E_o$ for nodes that are assumed to be within one radio hop of any of the nodes involved in the transfer. Estimating $\bar{c}_i$ therefore requires knowledge of the current routing topology. This information is readily available: the periodic summary messages, sent to the base station by every node, include the node's radio neighbors and parent in the routing tree. Cost vectors can be easily recomputed whenever the routing topology changes.

To ensure that all nodes meet the lifetime target L, Lance models the energy availability at each node using a token bucket with depth D and fill rate C/L,

corresponding to the mean discharge rate. D is determined by the target lifetime L, the battery capacity B and the background drain rate R. In general, $D = B - L \times R$, so D represents the energy remaining after the node reserves enough to ensure it can meet its target lifetime at the background level.

4.7.3 Lance Optimizer

The Lance optimizer is responsible for scheduling ADUs for download, based on knowledge of the set of ADUs currently stored by the network, their associated values, and costs. In our design, Lance attempts to download a single ADU at a time, in order to prevent network congestion, although it may be possible to download multiple ADUs simultaneously, depending on the network topology. A download completes either when the entire ADU has been received or a timeout occurs.

Lance's optimization process attempts to maximize the value of the ADUs retrieved while adhering to the lifetime target L. In essence, we seek a greedy heuristic approximation of the multidimensional knapsack solution that would be used by an oracle with complete knowledge of the ADUs sampled by the network over all time. The optimizer first excludes ADUs that would involve nodes without enough energy to perform a download. That is, if the token bucket for a given node m has $E(m)$ joules, ADUs for which $E(m) < c_i^m$ are excluded from consideration. Note that as the bucket fills, the ADU may become available for download at a later time. We call these ADUs *infeasible*, and the remaining *feasible*.

To determine the next ADU to download, the optimizer considers the value v_i of each ADU and its associated cost $\bar{c}_i$. We consider three *scoring functions* that assign a download score to each feasible ADU; the ADU with the highest download score is downloaded next. In the case of ties, an arbitrary ADU is chosen.

The first scoring function, *value-only*, simply downloads the feasible ADU with the highest value v_i. Note that *value-only* will meet the network's lifetime target (since only feasible ADUs are considered) but does not rank ADUs according to cost. The second scoring function, *cost-total*, assigns the score $\hat{v}_i$ by scaling the value of the ADU by its total cost: $\hat{v}_i = v_i / \sum_j c_i^j$. The feasible ADU with the highest score is then downloaded from the network. This approach penalizes ADUs stored deep in the routing tree, which have a higher overall cost than those located near the base station.

The third scoring function, *cost-bottleneck*, scales the ADU value v_i by the cost to the node that is an energy bottleneck for downloading this ADU. That is, let b represent the node with the minimum value of $E(b)$ such that $c_i^b > 0$. *cost-bottleneck* sets the score $\hat{v}_i = v_i / c_i^b$. The intuition behind this scoring function is that the most energy-constrained node should be considered when scoring ADUs for download. We evaluate all three scoring functions in the next section and show that they yield very different results in terms of spatial distribution and energy efficiency.

We evaluate each scoring function against an optimal solution calculated by using knowledge of all future ADUs and solving the multi-dimensional knapsack problem. Since an online solution is required, this provides an upper bound on the achievable performance of the online scoring functions.

4.7.4 Evaluation and Results

Prior work [21] presents a thorough and detailed evaluation of Lance on a variety of real and synthetic workloads, evaluated both in simulation and in experiments on a large sensor network testbed [19]. We present a subset of these results here to further the discussion.

4.7.4.1 Simulation and Testbed Experiments

We began by evaluating Lance using a realistic system simulator which allowed parameters – ADU size, distribution of ADU values, network topologies, download speeds, energy costs and target lifetimes to be easily varied. Our simulations focus first on evaluating the candidate scoring functions described above, and secondly on assessing the performance of Lance as the parameters above are varied.

Our first simulation experiment used a simple 10-node linear topology. Figure 4.15 shows both the performance of each scoring function against the offline optimal system and a breakdown of the estimated energy consumed on each node during the simulation. We divide each node's energy usage into three components: *downloading* energy consumed by the node when it is transferring one of its own ADUs to the base station, *forwarding* energy consumed while transferring data upstream for a downstream node, and *overhearing* energy which accounts for the small overhearing penalty imposed by the TinyOS LPL implementation on nodes that overhear (but are not participating in) nearby transfers.

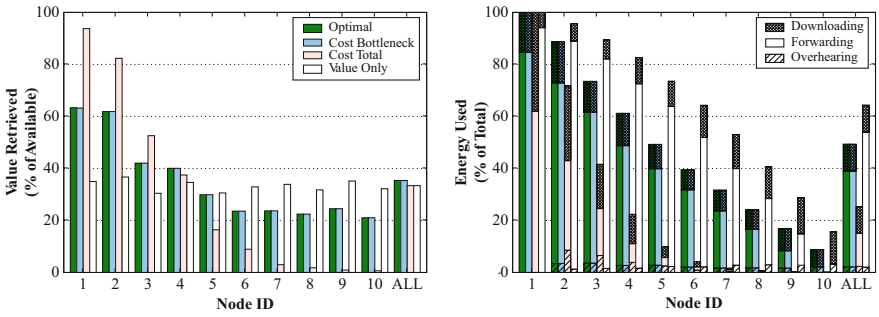


Fig. 4.15 *Per-node distribution of ADU value and energy usage for the linear simulation experiment.* The *left* graph shows the amount of data value downloaded from each node, while the *right* graph breaks down the amount of energy used by each node into the downloading, routing and overhearing components. Node 1 is closest to the base station

The energy costs for various operations are modeled as follows. The background current drain of each node is set to 2 mA, based on empirical measurements of a TMote Sky sensor node performing high-data-rate sampling and storing to flash. We also measured the current consumption to download an ADU from a sensor node, and derived the energy costs for downloading ($E_d = 17.6$ mA), routing ($E_r = 16.9$ mA), and overhearing ($E_o = 2$ mA). Our experiments assume that each node can only overhear its parent in the routing tree; developing more detailed overhearing models is the subject of future work. Computing the components of the cost vector for a particular ADU is done by multiplying the current consumption by the ADU download time for each node either downloading, routing, or overhearing the transmission.

Figure 4.15 confirms the intuition behind the scoring function behavior. *value-only* downloads roughly equal value from each node, but fails to match the optimal performance. *cost-total* downloads more data from nodes near the sink. *cost-bottleneck* comes close to matching the optimal solution, retrieving over 99% of the value retrieved by the optimal solution.

To confirm our intuition we ran a set of simulations using a more realistic 25-node tree topology and using a variety of different ADU value distributions. We draw ADU values from several distributions in an attempt to understand Lance's behavior as the properties of the sampled data change. Three value distributions are used: uniform random, exponentially distributed, and Zipf with exponent $\alpha = 1$. We also make use of an ADU value distribution based on a 6 h seismic signal collected at Reventador Volcano, Ecuador in 2005 [20] for a later experiment. Table 4.16 summarizes the results, showing that the *cost-bottleneck* scoring function outperforms the other two in most cases, with optimality values between 87.1% and 96.9%. The one exception is the 4-month Zipfian data set, where *cost-total* slightly outperforms *cost-bottleneck*.

		Scoring Functions		
Distribution	Lifetime	Value Only	Cost Total	Cost Bottleneck
Uniform	4 months	62.4%	90.5%	**93.2%**
	11 months	43.4%	68.0%	**96.9%**
	18 months	44.6%	49.0%	**90.0%**
Exponential	4 months	83.9%	85.1%	**88.0%**
	11 months	70.4%	82.0%	**93.0%**
	18 months	67.2%	72.8%	**91.2%**
Zipfian	4 months	84.7%	**91.4%**	87.1%
	11 months	63.8%	91.1%	**96.2%**
	18 months	53.1%	86.9%	**93.8%**

Fig. 4.16 *Optimality of different scoring functions.* This table summarizes simulation results evaluating the three different scoring functions. Results are shown for several different lifetime targets and value distributions. *cost-bottleneck* out-performs the others in almost all cases

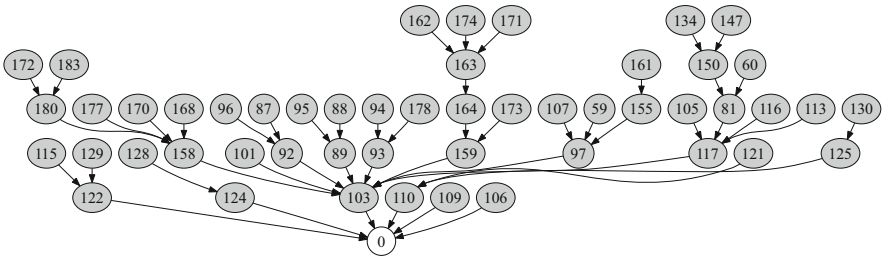


Fig. 4.17 *Topology for testbed experiments.* This graph shows the 50 node topology used for the testbed experiments shown in Fig. 4.18

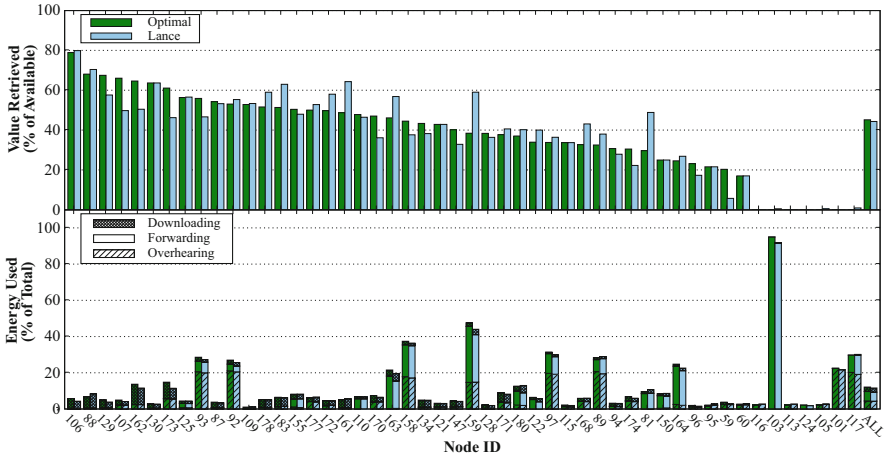


Fig. 4.18 *Optimality and energy use in the 50-node testbed experiment.* Lance achieved near-optimal performance during this 8-h testbed experiment, retrieving 98% of the value obtained by the offline optimal algorithm

We also ran Lance on our MoteLab [19] Wireless Sensor Network Testbed, in a 50-node configurations shown in Fig. 4.17. These experiments stress the system in a realistic setting subject to radio interference and congestion, and exercise the multihop routing protocol, Fetch reliable data-collection protocol, and ADU summary traffic generated by the nodes. For these experiments, we injected artificial ADU values directly into each node rather than relying on the nodes sampling real sensor data; this approach allows us to perform repeatable experiments that explore a wider range of ADU value distributions. We use the *cost-bottleneck* scoring function.

Figure 4.18 shows the results of a 50-node testbed experiment using a Zipfian data distribution and a target lifetime of 6 months. The upper portion of the figure shows the amount of data value obtained by Lance from each node, compared to the optimal solution (which was computed offline). Nodes are sorted by decreasing optimal value. As the figure shows, Lance achieves very close to the optimal solution, with an optimality of 98% overall. In some cases, Lance incorrectly downloads more

data from some nodes and less data from others; this is due to the inherent limitations of an online solution that cannot foresee future ADU values. The lower portion of the figure shows the energy breakdown for each node with downloading, forwarding, and overhearing costs shown. Some nodes consume more than others because of their location in the routing tree. For example, node 103 in uses a great deal of energy for routing packets as it is one hop from the base station, although no ADUs are ever downloaded from that node.

4.7.4.2 2007 Deployment Analysis

While the earlier, priority-based version of Lance was used to drive our 2007 deployment at Tungurahua, we were still able to analyze this system's performance using utility-based metrics. To evaluate Lance's behavior with respect to an "optimal" system, we took the 8,483 RSAM summaries received during a 16-h period when the debiasing filter (see Sect. 4.6.2) was enabled. Using this information, we compute the set of ADUs that the optimal system would have downloaded, with complete knowledge of all ADUs but limited to the same time duration the original network was operating. We assume the download throughput for a given node is always the mean throughput for that node observed during the deployment (see Fig. 4.12). This calculation ignores energy constraints because the deployed system did not consider energy costs.

An optimal system would have downloaded 392 out of the 8,483 ADUs, whereas the actual system downloaded 418 ADUs during this time.[3] The total value of ADUs downloaded by the optimal system is 10,678, whereas the value of the actual network was 10,629, for an optimality of 99.5%. Lance did an exceptional job of extracting the highest-value data from the network using our online heuristic algorithm.

We can perform a similar analysis during the period of time during which the network was using the EWMA-based summarization function. As with the RSAM-based summarization function, we estimate the optimal set of ADUs that an oracle would have downloaded. During a 25-h period, the network reported 11,012 unique ADU summaries. An optimal system would have downloaded 554 ADUs with total value 5,77,377. The actual network downloaded 518 ADUs with a value of 5,39,115, for an optimality of 93.3%.

As a final evaluation metric, we wish to consider how well Lance, configured in this manner, was able to download seismic signals representing earthquakes. Given the low level of volcanic activity, it turns out that most of the ADUs downloaded by Lance contain no discernible seismic signal. In fact, upon manual inspection of the 518 ADUs downloaded during this period, we identified only 20 ADUs showing a clear earthquake signal, corresponding to only 9 separate seismic events. Note that

[3] The optimal system would download fewer ADUs than the real system due to the variation in the throughput to each node: the optimal system would download more ADUs from nodes with lower throughput, thereby limiting the total number of ADUs it could download.

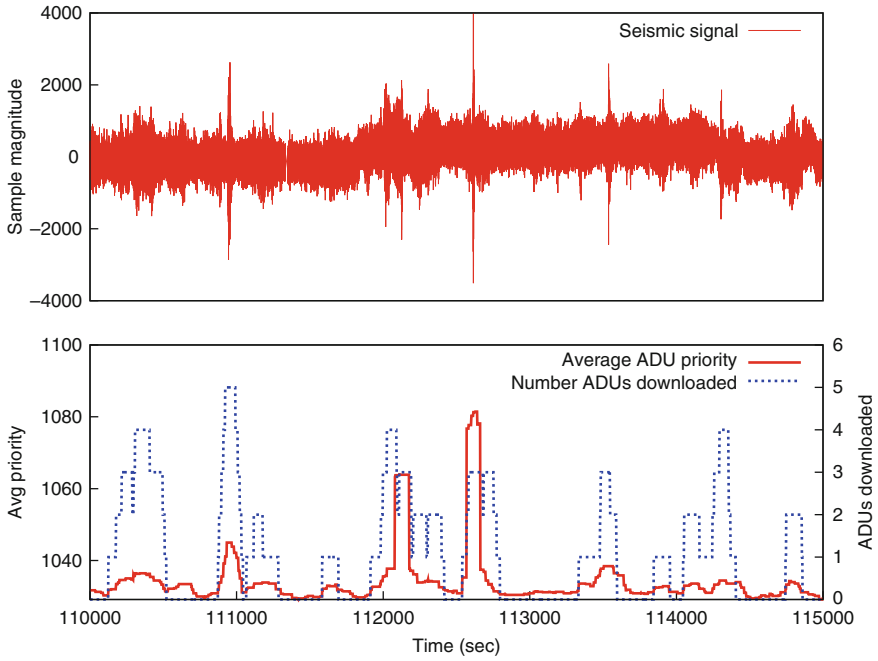


Fig. 4.19 *Lance download behavior overlaid with average ADU value.* The *top* plot shows the continuous seismic signal collected by a single node. The *lower* plot shows the average value of ADUs and the number of ADUs downloaded for each window

we did *not* configure Lance to explicitly download correlated earthquakes by using an appropriate policy module (described in Sect. 4.6), so we would not expect a high degree of coverage for the same event across multiple nodes.

Figure 4.19 shows the behavior of Lance during a representative 83-minute period. In the figure, we have broken time into windows of one-half an ADU duration (55 s in this case), and computed the mean ADU value as well as the number of downloaded ADUs that overlap each time window. As the data shows, elevated seismic activity is well-correlated with an increase in the ADU value from across the network, as well as the number of downloaded ADUs. Moreover, the few cases of clear seismic activity in the trace (at times 1,11,000, 1,12,700, and so forth) tend to have more ADUs downloaded. Of the 9 separate seismic events, a total of 27 ADUs were downloaded, representing a per-event "coverage" of 3 ADUs per event. This represents just under half of the 7 nodes participating in the network.

4.7.4.3 Results Summary

To summarize the results, we found that the *cost-bottleneck* scoring function allows Lance to approach optimal performance across a variety of network sizes,

bandwidth distributions and target lifetimes. By directing scarce energy resources towards the most valuable data, the overall efficiency of the network considered as a whole can be significantly increased.

4.8 Conclusions and Future Work

Based on our experience with data quality spanning three deployments and multiple system components we see many directions for future work. Every component of datum and dataset quality mentioned here is open for future research and continued development.

While we have made progress in pinning down the the datum quality requirements specific to this domain, work continues on hardware-software codesign, time synchronization and reliable data collection. We are currently designing a sensor interface board for the iMote2 intended to support the volcano application as well as other high data-rate sensing tasks, and are revisiting many of the original design decisions that produced the volcano monitoring board described here. Recent efforts within our group have continued to work on reliable time synchronization in other settings such as body sensor networks. And we have begun a redesign of the data collection component inspired by approaches such as Hop [7] that seek to reduce power consumption and improve performance over lossy, low-power links.

As the Lance work brings out, there are tradeoffs possible between a deployed networks target lifetime and the quantity and quality of the data provided to the application. Lance takes a position on one end of a sliding spectrum in that it holds the system lifetime constant, treating it as an input parameter that must be met, and then tries to turn other knobs to maximize the output data quality given other constraints. Another approach would be to hold the output data quality constant and try to allow the system to last as long as possible while providing data at a minimum fidelity set by the application. Between these two endpoints there are multiple approaches which would tend to trade off application data quality for increased system lifetime. Given the difficulties inherent in performing this tradeoff, we have not yet explored this interstitial area. However, this area seems quite fertile for future work.

In addition, maximizing the output of a deployed network under resource constraints requires continuing to build strong connections between the notion of data quality operative within the network and the actual needs of the application. In Lance, we do not dictate that applications use a simplified scalar score to indicate data quality, but our architecture tends to push applications in this direction. Many applications, however, may not be able to reduce a complex set of considerations into a single scalar value. Enriching this interface may help allow further dataset quality optimizations. In general, however, the interface between the end user's goals and the simplified metrics operating within the network must receive further attention to ensure the broader applicability of solutions that attempt in-network optimization.

In this chapter we have attempted to zero in on data quality issues inherent in the development of a sensor network supporting the monitoring of active volcanoes. Through our experience in this area spanning three deployments we have grappled with many aspects of data quality, from where the sensor meets the sensor node up to complicated optimization approaches running at the base station. We have attempted to narrate, in a helpful way, our experiences in this area, and hope that some of the struggles we have elucidated will smooth the waters for other deployments.

Acknowledgements The authors would like to acknowledge those who helped make this project possible. Professors Jeff Johnson (New Mexico Tech) and Jonathan Lees (University of North Carolina), our seismology collaborators, made the entire endeavor happen, schooling computer scientists in the fine art of volcanologic research and providing invaluable assistance with the design and deployment of each of our systems. Mario Ruiz (IGEPN) and Omar Marcillo (New Mexico Tech) provided additional domain science and deployment assistance. Harvard students Konrad Lorincz, Pat Swieskowski and Stephen Dawson-Haggerty helped design and implement the sensor network systems we deployed. Researchers, staff and students at the Instítuto Geofisico, Escuela Politécnica Nacional (IGEPN) in Ecuador provided essential ground support during each of our deployments. Finally, Jim MacArthur and William Walker at the Harvard School of Engineering and Applied Sciences Circuit Lab aided significantly in the design and construction of our interface board and enclosures.

References

1. Benz J et al (1996) Three-dimensional P and S wave velocity structure of Redoubt Volcano, Alaska. J Geophys Res 101:8111–8128
2. Chouet B et al (2003) Source mechanisms of explosions at Stromboli Volcano, Italy, determined from moment-tensor inversions of very-long-period data. J Geophys Res 108(B7):2331
3. Dietel C et al (1989) Data summary for dense GEOS array observations of seismic activity associated with magma transport at Kilauea Volcano, Hawaii. Tech. Rep. 89–113, U.S. Geological Survey
4. Hui JW, Culler D (2004) The dynamic behavior of a data dissemination protocol for network programming at scale. In: Proc. 2nd ACM Conference on Embedded Networked Sensor Systems (SenSys'04)
5. Kim S, Fonseca R, Dutta P, Tavakoli A, Culler D, Levis P, Shenker S, Stoica I (2007) Flush: A Reliable Bulk Transport Protocol for Multihop Wireless Networks. In: Proc. SenSys'07
6. Lees J, Crosson R (1989) Tomographic inversion for three-dimensional velocity structure at Mount St. Helens using earthquake data. J Geophys Res 94:5716–5728
7. Li M, Agrawal D, Ganesan D, Venkataramani A (2009) Block-switched networks: A new paradigm for wireless transport. In: NSDI'09: Proceedings of the 6th conference on Symposium on Networked Systems Design & Implementation, USENIX Association, Berkeley, CA, USA
8. Maroti M, Kusy B, Simon G, Ledeczi A (2004) The flooding time synchronization protocol. In: Second ACM Conference on Embedded Networked Sensor Systems
9. McNutt S (1996) Seismic monitoring and eruption forecasting of volcanoes: A review of the state of the art and case histories. In: Scarpa, Tilling (eds) Monitoring and Mitigation of Volcano Hazards, Springer-Verlag Berlin Heidelberg, pp 99–146
10. Moran S, Lees J, Malone S (1999) P wave crustal velocity structure in the greater Mount Rainier area from local earthquake tomography. J Geophys Res 104(B5):10,775–10,786
11. Moss D, Hui J, Levis P, Choi JI (2007) Cc2420 radio stack. TinyOS Extension Proposal TEP-126, http://www.tinyos.net/tinyos-2.x/doc/txt/tep126.txt

12. Murray T, Endo E (1992) A real-time seismic-amplitude measurement system (rsam). In: Ewart, Swanson (eds) Monitoring Volcanoes: Techniques and Strategies Used by the Staff of the Cascades Volcano Observatory, 1980-1990, vol 1966, USGS Bulletin, pp 5–10
13. Neuberg J, Luckett R, Ripepe M, Braun T (1994) Highlights from a seismic broadband array on Stromboli volcano. Geophys Res Lett 21(9):749–752
14. Paxson V (1998) On calibrating measurements of packet transit times. In: Measurement and Modeling of Computer Systems, pp 11–21, URL citeseer.ist.psu.edu/paxson98calibrating.html
15. Paxson V (2004) Strategies for sound internet measurement. In: IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement, ACM Press, New York, NY, USA, pp 263–271, DOI http://doi.acm.org/10.1145/1028788.1028824
16. Phillips W, Fehler M (1991) Traveltime tomography: A comparison of popular methods. Geophys 56:1649–1649
17. Scarpa R, Tilling R (1996) Monitoring and Mitigation of Volcano Hazards. Springer-Verlag, Berlin
18. Werner-Allen G, Johnson J, Ruiz M, Lees J, Welsh M (2005) Monitoring volcanic eruptions with a wireless sensor network. In: Proc. Second European Workshop on Wireless Sensor Networks (EWSN'05)
19. Werner-Allen G, Swieskowski P, Welsh M (2005) MoteLab: A Wireless Sensor Network Testbed. In: Proc. the Fourth International Conference on Information Processing in Sensor Networks (IPSN'05)
20. Werner-Allen G, Lorincz K, Johnson J, Lees J, Welsh M (2006) Fidelity and yield in a volcano monitoring sensor network. In: Proc. 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2006), Seattle, WA
21. Werner-Allen G, Dawson-Haggerty S, Welsh M (2008) Lance: Optimizing high-resolution signal collection in wireless sensor networks. In: Proc. ACM Conference on Embedded Networked Sensor Systems (Sensys), Raleigh, NC, USA

Chapter 5
VoxNet: Reducing Latency in High Data Rate Applications

Michael Allen

Abstract High data-rate sensing is a challenging aspect of WSN research due to the large amounts of data generated by each sensor node. When data is being generated faster than it can be transported over multiple network hops, there is a need to apply on-node, in-network event detection, filtering and other data processing techniques. Although contingent on the specific application, signals in the audible acoustic spectrum must typically be sampled at kHz rates. This makes acoustics a particularly challenging phenomena in the high date rate class.

In the context of high data-rate sensing, this chapter describes in detail the deployment of a specific application on a platform for distributed acoustic sensing applications called *VoxNet*. The application, on-line source localization of animals from their vocalizations is a compelling tool for evolutionary biologists; timely in-field position estimates can enable biologists to augment their observations.

A ten-day deployment of VoxNet highlighted several important problems which could not have been predicted in advance, largely related to the instrumentation of the system and end-to-end latency from event detection to position estimate. Using the extensive log data gathered during the deployment, two strategies to improve end-to-end system timeliness were developed. These are *Lazy Grouping*, a centralized algorithm which performs on-line grouping of event data and facilitates its collection from the network, and an *Adaptation policy* which allows nodes in the network to individually and dynamically evaluate whether to process data locally based on previous data transfers. Whilst the design and evaluation of these refinements is based on application-specific experiences, the techniques themselves are transferable to a variety of high data rate applications.

M. Allen (✉)
Singapore-MIT Alliance for Research and Technology, S16-06-10, 3 Science Drive 2, Singapore 117543
e-mail: michael@smart.mit.edu

E. Gaura et al. (eds.), *Wireless Sensor Networks: Deployments and Design Frameworks*,
DOI 10.1007/978-1-4419-5834-1_5,

5.1 Introduction

In wireless networked sensing, applications which require sensor nodes to sample data at high rates (on the order of 100 Hz or more) face a number of challenges which can be ignored when sampling data at lower, sub-hertz rates. Data generated at high rates often cannot be streamed across a multi-hop network at the same rate. The *sense and send* approach, where all raw data is collected at a central point is not applicable in these cases. Instead, local processing techniques must be used to reduce the amount of data which must be sent over the network. Furthermore, if a phenomena must be sampled at a high rate, it is hard to conserve energy by duty cycling.

Assuming the lower threshold of high data rate applications to be 100 Hz [32], it is clear that the higher the data rate, the more pressing these challenges become. Audible acoustics applications, where data must be sampled at tens of kHz, are indicative of the higher end of the high data-rate spectrum.

This chapter uses the motivating application of acoustic detection and localization of marmots in their natural habitat to inform the design and deployment of VoxNet: a distributed, rapidly deployable hardware and software platform for distributed acoustic sensing. The motivating application requires that marmot calls are detected by nodes within the network, and that these node-level events are correlated across the network and processed to result in a position estimate which the user can act upon.

While the design of VoxNet was motivated by bio-acoustics applications, it holds the potential to be applied in other domains thanks to a flexible, deployable hardware platform and a high-performance distributable programming interface.

As the first of its contributions, this chapter provides a detailed account of the ten-day deployment of the first VoxNet system iteration [3]. In particular, several issues were found to affect the end-to-end latency of the system in providing position estimates during its deployment.

The second contribution of this chapter is the design and proof-of-concept evaluation of two system refinements based on the observations and system logs made in-situ – Lazy Grouping and Adaptation. These refinements aim to reduce end-to-end latency of position estimation and improve timeliness by reducing the amount of data sent over the network.

Lazy Grouping is an event grouping and data collection approach that aims to reduce latency by gathering only data that is closely grouped in time and thus corresponds to a network-wide event. Adaptation aims to reduce latency by allowing a node to decide whether to process a specific part of the localization data flow locally. This decision is made dynamically, based on recent transfer history, and allows the node to adapt to changing network conditions.

Data transfer latency is an issue for many high data rate applications, and although the refinements presented here are defined in the context of the specific marmot localization application, both solutions can be applied to a variety of event-based acoustic sensing applications.

This chapter is organized as follows: in Sect. 5.2, the motivating marmot localization application is discussed. In Sect. 5.3, the architecture and design of

VoxNet is presented. The in-situ deployment of the marmot localization application is described in Sect. 5.4 and the subsequent latency-related problems discussed in Sects. 5.5–5.7 present the design and evaluation of Lazy Grouping and Adaptation respectively. Finally, Sect. 5.8 discusses future work for VoxNet.

As with other chapters in this book, the deployment experiences presented are valuable for the novice deployer. The deployment and latency problems are described in Sects. 5.4 and 5.5. From these sections, the reader should get a good understanding of the importance of in-situ deployment as a key step in identifying areas for system improvement. The reader should also find several useful techniques to address the unpredictable setbacks that occur during deployment as well as the the vital importance of correct instrumentation of a deployed system to support off-line performance analysis.

5.2 In-situ Acoustic Source Localization

The field of bioacoustics research is broadly concerned with the reception of acoustic signals made by animals. One particular aspect of bioacoustics research focuses on understanding the purpose behind certain acoustic signals produced by animals and how these relate to the environment in which they are produced.

In bioacoustics, counting the number of individuals present in a given recording, known as *acoustic census*, is important for the researcher to help estimate the population size in a given area. Acoustic census is possible because many mammals and birds produce loud alarm calls, territorial calls, and songs that are species-specific, population-specific, and often individually identifiable [23]. As such, these vocalizations can be used to identify the species present in an area, and potentially to count individuals. Acoustic monitoring for census has been shown to be useful for cane-toad monitoring [9], elephants [25], birds [17] and whales [11].

Recognizing animal and bird calls and distinguishing between different individuals can be a difficult problem. For example, performing an accurate census of birds in dense, tropical forests requires a skilled ornithologist. Furthermore, some animal vocalizations are out of the range of human hearing, such as the infrasonic signals produced by elephants [25].

In this chapter, the focus is on acoustic localization of animals. This is for several reasons: firstly, the localization of animals is a key component in census since determining the position of an animal based on its call can help the scientist disambiguate similar calls. Secondly, localization can also help the scientist understand territorial behavior of the animal of interest. Thirdly, source localization is a challenging problem in distributed acoustic sensing.

The particular sound source being considered in this chapter is the marmot, a medium-sized rodent native to the western United States. Marmots are notable for their high pitched *alarm calls* when they sense danger. Localization in this case is helpful for understanding where an animal was when it made a call, and why it made the call – to warn others, or protect itself, for example. Although marmots call relatively infrequently, they are prone to *bouts* of alarm calls, where 20 or more

calls are made in a single position, with an interval of around 1 s between each call. A marmot call is short, lasting around 40 milliseconds, with a center frequency of 5-6 kHz; it is also high energy, making it noticeable against background noise.

5.2.1 Enabling In-Situ Automation and Interaction

Traditionally, bioacoustics researchers gather data by wiring one or more acoustic sensors to a multi-channel recorder situated near the animal habitat of interest. The researcher attends the deployment and keeps a detailed log of interesting acoustic events to aid and complement data post-analysis in the lab.

This approach commonly results in several hours' worth of continuous acoustic recording, which must be manually examined off-line by the scientist. Researchers find sections of audio that warrant further analysis by listening to the full streams as well as examining the data in the frequency domain using spectrograms. They use a variety of domain-specific applications and frameworks for data acquisition, event detection and analysis, such as Raven [6] and Ishmael [24]. General purpose audio processing and recording tools such as Audacity [22] and Baudline [5] also allow the user to investigate the characteristics of audio signals. Often they can be used in combination with general purpose tools like Matlab [7] to perform ad-hoc processing that mimics the functionality of more specialized processing frameworks.

Relying exclusively on off-line analysis is limiting: by the time the analysis is completed, the experiment is over, and opportunity to collect additional data about important events have passed. Furthermore, in-field analysis opens up possibilities for active observations such as call and response.

Although a prototype WSN for recording time-synchronized acoustic data from multiple nodes (using the Acoustic ENSBox platform) has been presented [14], it does not provide any on-line processing capabilities. A WSN-based system would be a good fit for a tool to enable in-field, on-line analysis; data could be collected and processed by nodes, and presented to the user. Additionally, wireless operation would allow for convenient physical reconfiguration and larger spatial coverage during experimentation, when compared to a wired approach.

5.2.2 Usage Scenario

In the intended usage scenario, a scientist wants to detect marmot alarm calls and determine the location of marmots, relative to the known positions of their burrows. By obtaining the marmot position estimates as the system is running, the scientists can augment their written log-traces with pictures of animals under observation. Ideally, these pictures could be taken by automated imagers which are actuated based on the results of the position estimate provided by the localization system.

Such a localization system may also enable the scientist to record additional data about the current habitat conditions, such as what caused the animal alarm call, and which animal raised it. This requires that position estimates are delivered as close to real-time as possible, in order to enable actuation and image collection.

The user's expectation is that a position estimate will arrive in a *timely* manner, so that it can be used to make specialist application related observations and decisions. Therefore, the *timeliness* of the system is an important measure of its end-to-end performance. Unlike a soft or hard real-time system, specific timing deadline is not imposed on the system for converting detections to position estimates; however, the longer the user has to wait for the result, the less likely it is that the information can be of use. Therefore, the focus of the system is on presenting results *as quickly as needed*.

Since the target users of this system are not wireless sensing experts, it is crucial that the system be easy for them to deploy, initialize, and configure/reconfigure. To be useful as a field tool, there must also be facilities for reconfiguration and tuning during operation.

5.2.3 Localization Algorithm and Components

In this work we used a source localization algorithm taken from previous, related, proof-of-concept work to localize marmots off-line from synchronized audio streams gathered in-situ [1] and adapted it to cater for the on-line nature of the usage scenario.

Figure 5.1 shows the conceptual flow for on-line source localization. Possible alarm calls are detected using an on-line event detector that runs on each node in the network [2]. When a node detects an event, it computes a bearing estimate using

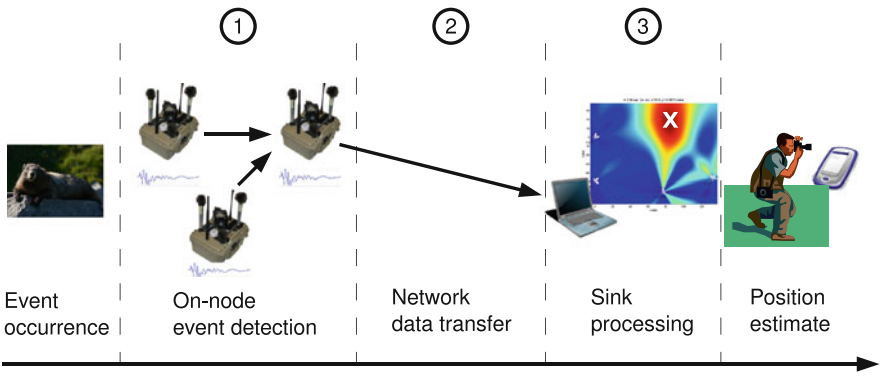


Fig. 5.1 The data flow for an on-line source localization system

the Approximated Maximum Likelihood (AML) algorithm. Bearing estimates taken from multiple points are fused together provide an estimate of the caller's location.

The on-line event detector is based on an adaptive threshold. The detector estimates the noise in the channel using two Exponentially Weighted Moving Average (EWMA) filters, for mean and variance, assuming the noise in the channel is Gaussian. Any deviations from the estimate of noise level computed beyond a pre-determined threshold are flagged as events of interest. To provide application specific filtering, only the energy from pre-determined frequency bands are used in estimating noise or events. The energy bands of interest are determined a priori according to the characteristics of the acoustic phenomena. The energy of the signal is determined in the frequency domain as the magnitude of the sum of the pre-determined frequency bands of interest.

The Approximated Maximum Likelihood algorithm is a maximum-likelihood based direction of arrival estimator. The estimator is equivalent to beamforming, but is calculated in the frequency domain based only on the strongest frequency components. The output of the AML is a 360 element vector, where each element represents the likelihood that the sound came from a given direction relative to the node's orientation.

The data fusion algorithm combines the AML vector from each node to determine the most likely position of the marmot call. This is performed by a brute force search of a pre-defined 2D space around the area covered by the nodes. Each point in the search space is assigned a pseudo-likelihood value based on the projection of the AML vectors from each node; this is essentially finding the region of maximum agreement of AML vectors. The largest aggregate value in the search space represents the most likely position of the marmot.

Under controlled experimentation, this AML and data fusion approach yielded an RMS positional error of 0.7 m when the source was inside the convex hull created by the sensors, and 2.07 m when the source was outside of the convex hull [2]. This result was based on a network of six Acoustic ENSBox [14] nodes placed in a rectangular configuration over 35 m by 60 m, with a 10 cm resolution search space.

Having introduced our motivating application, in the next section we describe VoxNet, the platform that supports the marmot localization application. The design of VoxNet is motivated by specific experience with bioacoustics-related sensing applications, but is intended to be general enough to support a variety of distributed, wireless, acoustics-based applications.

5.3 The VoxNet Platform

VoxNet is a hardware and software platform for distributed acoustic sensing. This section describes both hardware and software, covering the overall system architecture and individual software layers that make up the VoxNet platform.

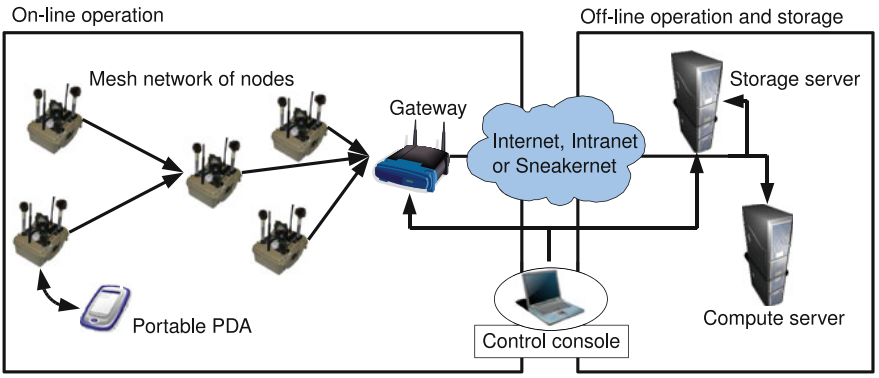


Fig. 5.2 The VoxNet system architecture, showing both on-line and off-line operation contexts. The control console is the user interface to both contexts

5.3.1 System Architecture

The full VoxNet architecture operates in two contexts: on-line and off-line. Figure 5.2 provides an overview of this architecture, representing a framework where programs can be written, compiled, and disseminated, and the results can be archived and visualized.

VoxNet is made up of several hardware components: a network of nodes, a gateway, a control console, the compute server and the storage server. *Nodes* form the main part of the on-line operation of the system, sensing and potentially processing audio data. The *gateway* provides a connection to relay traffic between the control console and the network of nodes. The *control console* provides a unified interface for both the on-line and off-line portions of the platform. It hosts the interaction tools, and acts as a sink to the programs running in the network. Results and diagnostic data are returned to the control console for display and visualization. A *PDA* may also be used to visualize data from network streams whilst mobile in the field. The *storage server* archives data acquired by nodes for later retrieval and processing. The *compute server* provides a centralized unit for off-line data processing and analysis.

5.3.2 Interaction Model

Distributed VoxNet applications are written in the WaveScript macroprogramming language [15], abstracting the programmer from the particular details of the network and specifics of the hardware platforms being used. WaveScript programs can operate over either real-time data streams or archived data residing in a distributed

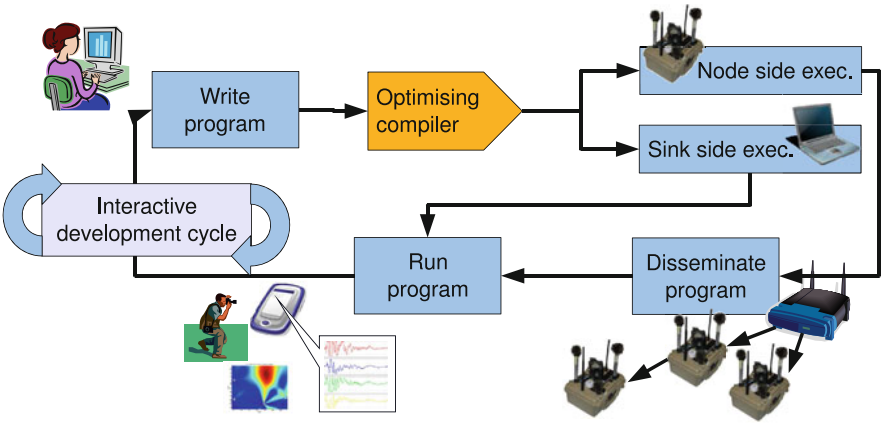


Fig. 5.3 The on-line, interactive development cycle for VoxNet

system of sensors and back-end servers. This model enables users to tune and further develop applications during pilot deployments, and enables the system to be used as an interactive measurement tool while it is deployed. Figure 5.3 shows the in-situ development cycle. This is important for many short-term scientific deployments, because it allows a scientist to immediately explore newly observed phenomena. By design, this model allows room for iterative improvement based on in-situ deployment experiences. As the system becomes more familiar to the user, their needs and expectations of the system's in-situ operation can be further refined.

When running on-line, WaveScript programs are created, compiled and disseminated to the network via the control console. Data arriving at the control console can be visualized as well as archived to a storage server. This data includes application specific streams, as well as logging data and operational statistics. When in off-line operation, streams of results and offloaded raw sensor data can be archived to a storage server and later processed off-line, using the same user interface. Applications that would previously have run on the control console and nodes, can be run on the compute server with data queried from the storage server.

5.3.3 Hardware

The basic node hardware used in VoxNet is a revision of the Acoustic ENSBox prototype [14]. Based on in-field experiences with the original Acoustic ENSBox, additional hardware components were added, and packaging was significantly improved to enable easier and more robust deployment.

The VoxNet node (shown in Fig. 5.4) shares the same main processor board as the original Acoustic ENSBox, based on the Sensoria Slauson board: a 400 MHz

Fig. 5.4 The VoxNet hardware platform

PXA 255 processor with 64 MB memory, 32 MB on-board flash and two PCMCIA slots. The two PCMCIA slots house a 4 channel sound card with 48 kHz sampling rate and a 200 mW 802.11 b wireless card with up to 200 m transmission range.

In addition to the main processor board, the VoxNet node utilizes an auxiliary processor, the Gumstix Connex 400, connected to the Slauson by wired Ethernet. The Gumstix hosts an 8 GB Compact Flash card that can archive data streams from the Slauson.

VoxNet nodes currently consume 7.5 W when running, and have an 8-h lifetime from an internal 5,400 mAh Li-ion battery. To support duty-cycling, the VoxNet node includes a Mica2 mote [4] that is always on, linked to the Slauson board via a serial line. The Mica2 manages several peripheral components, namely: two indicator LEDs, an external attention button, a GPS module, a 3-axis accelerometer and temperature sensor. It can also control power to the other components of the system, including the Gumstix, Slauson, the audio output power amplifier and audio input preamplifiers. Software on the Mica2 allows the Slauson to power the system down for a specified interval and awaken when the attention button is pressed. The Mica2 CC1000 radio is not currently used, but could be utilized to implement over-the-air wake-up of the main processing board.

The VoxNet node is a self-contained unit installed in an 1,150 Pelican case with the microphone array integrated into the lid of the case; it has a total weight of 2.3 kg. The microphone modules detach and pack inside the box for shipping, and all external connectors are weather-resistant and sealed. The microphones are protected by a waterproof latex cap and an artificial fur wind screen. During deployments nodes have survived repeated exposure to precipitation.

5.3.4 Software

The VoxNet software stack is made up of three integrated layers:

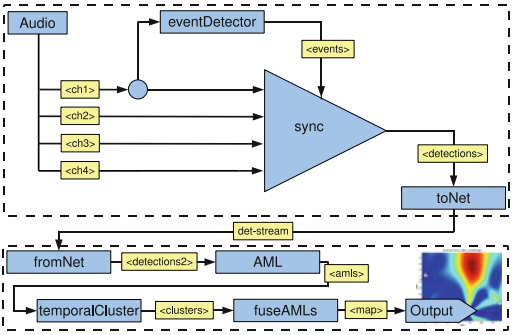
- The *high level application layer*, provided by the WaveScope stream processing engine, which handles compilation, optimization, and operation over the network.
- The *distribution and interaction layer*, which provides a collection of sub-application level control and visualization tools important for tracking resource availability, troubleshooting and usability in the field and in off-line computations.
- The *platform drivers and services layer*, which supports the operation of the interaction and application layers on the specific hardware used in VoxNet. This layer includes core components such as time synchronization and multi-hop network communication.

5.3.4.1 High Level Application Layer

In VoxNet, applications are run using the WaveScope stream-processing engine. The WaveScope project is an ongoing effort which started in 2006 at MIT with an aim to develop a stream processing system for high-rate sensor networks with data sampling at rates of hundreds to tens of thousands of Hertz per node [15, 16]. WaveScope programs are written using the custom stream-processing programming language, *WaveScript*. Similar to other stream-processing languages, *WaveScript* structures programs as a set of communicating stream operators, also known as kernels or filters. To write a program, the user writes a *script* that concatenates stream operators to form a directed graph. Stream operators consume one or more input streams, and produce one or more output streams. Users can write custom stream operators directly in WaveScript to augment the existing library of operators.

The WaveScript compiler converts the high-level application representation into efficient C code for either the sink (x86-based) or VoxNet node (ARM-based). The WaveScope engine used in VoxNet requires that users write both the sink and node side of the executables separately, and define where network communication takes place – that is, at which point the data streams must travel from node to sink. Two WaveScript operators are provided to allow this functionality, `toNet()` and `fromNet().toNet()` maps a local data stream onto a named network stream, and `fromNet()` subscribes to the named network stream, outputting it as a local data stream.

The high level data flow graph for the WaveScript implementation of the marmot localization application is shown in Fig. 5.5. Stream variables that flow between operators are shown in light gray, with text in ⟨angle brackets⟩, and stream operators are shown in dark gray. Audio data acquired by a node is directed into WaveScope by the `Audio` operator as four channels (`ch1` to `ch4`). Stream `ch1` is passed through the `eventDetector` operator, which produces a stream of tuples




```
// node application
(ch1, ch2, ch3, ch4) = Audio(44100)
events = eventDetector(ch1)
detections =
  sync(events, [ch1, ch2, ch3, ch4])
BASE <= toNet(detections,"det-stream")

// sink application
detections2 = fromNet("det-stream")
amls = AML(detections2)
clusters = temporalCluster(amls)
map = fuseAMLs(clusters)
BASE <= map
```

Fig. 5.5 The high-level graph of the WaveScope localization application. Operators and named streams are shown. The corresponding node and sink-side code is shown on the *right hand side of the figure*

called `events`, which correspond to the start and end times of marmot calls. The `events` stream is passed through the `sync` library operator, along with the four original audio channels. The `sync` operator outputs a four channel stream of raw data (`detections`) which corresponds to the data ranges specified in `events`. The `detections` stream flows through the `toNet` operator into a named network stream (`det-stream`).

At the sink, the `fromNet` operator receives data from any nodes which are publishing `det-stream`, outputting a `detections2` stream. `detections2` flows through the `AML` operator, producing `amls` – a stream of angle of arrival estimates. `amls` flows through the grouping operator `temporalCluster`, which time-correlates AML results, producing a stream of grouped data (`clusters`). This flows through a data fusion operator to produce a stream of position estimate maps, which are output to the user.

Implementing the marmot localization application using the WaveScript programming language and optimizing the compiler resulted in a 30% reduction in processor load and 12% in memory usage on-node when compared to an EmStar-based implementation on the same hardware [3].

5.3.4.2 Distribution and Interaction Layer

VoxNet implements network streams and a range of control and visualization tools to support the dissemination of applications from the control console to nodes, and the display of node health and application related-feedback. These tools are described below.

Network Stream Subscription. To support the flow of WaveScope streams across network boundaries in VoxNet, a TCP/IP based network stream abstraction using an advertise-subscribe model was developed. In this model, nodes or sink advertise streams that they serve, allowing clients to subscribe to them. The network stream

abstractions are used for all types of communication in the VoxNet system, including visualization, log messages, control messages and pushing new compiled programs. Networked streams are conceptually the same as WaveScope streams, flowing unidirectionally from one point to another. Additionally, a networked stream advertised by a node can flow to multiple clients.

To enable the network stream abstraction in VoxNet, the sink and each node in the network run subscription servers, which are responsible for creating new streams to be advertised, managing subscribers to each advertised stream, and subscribing to streams advertised by other nodes or the sink. Streams are identified by a descriptive name such as *detections* that must be unique to a given subscription server, but is not required to be globally unique.

Networked streams are intended to be 100% reliable to preserve the flow of data in WaveScope applications. To ensure reliable data transmission, the stream abstraction is built upon TCP/IP. The implementation of the network stream abstraction was built using EmStar's [13] event driven TCP server and client libraries as base functionality.

There are some well-known streams that are exposed by nodes and the sink during normal operation of VoxNet:

- *Control stream (Sink).* Allows the sink to send commands to all nodes in the network. All nodes subscribe to this stream.
- *Log stream (Node).* Allows nodes to send information to the sink. The sink subscribes to the control streams of all nodes known to the discovery server.
- *File stream (Sink).* Allows the sink to send data files such as new binaries to the nodes.

Discovery Service and Control Console. The control console is a centralized point of contact for the network that is responsible for node discovery, application dissemination, resource usage tracking, error logging, and profiling statistics. It serves as a mediator between users who want to install a program and the VoxNet distributed system, and hosts all appropriate compiler tools and scripts. The discovery service hosted on the control console maintains a list of active VoxNet nodes and back-end server machines, and tracks their capabilities and resource availability. When VoxNet nodes or servers start up, they connect to the control console at a well-known address and register with the network.

Control Tools. To control the VoxNet deployment, we developed the *WaveScope shell* (WSH): a text-based command-line interface to the control console, similar to the UNIX shell. WSH is implemented using GNU readline, and is based on the *remote broadcast shell* (RBSH) [12]. WSH communicates with nodes over the *control stream*, which all nodes in the network subscribe to by default. As the control console also subscribes to all nodes' control streams by default, log messages sent over this stream are passed to WSH for display to the user.

WSH can send arbitrary UNIX commands to nodes, as well as commands to pause and restart the current application, flush current stream buffers and send files from the control console to the node, such as new binaries. The shell also provides a

tabular display showing the current status of each node registered via the discovery protocol. Each stream a node currently exposes is displayed, along with the current send-buffer status for that stream, i.e. how much data is queued to send.

Finally, WSH provides a network test functionality, where all or a subset of nodes can be requested to send back arbitrarily sized amounts of data, with the resulting latency measured for each node. This can be used to diagnose network latency, and was extensively used during deployment.

Spill to Disk. Experience with field scientists showed that even in cases where detection and localization can be performed on-line, there is also value in recording a raw data set for future processing. While this desire may diminish as confidence is gained in data reduction algorithms, it will always be important in the testing phases of any new algorithm, as well as for interactive use, for example to replay and re-analyze recent data. To address these concerns, a *spill to disk* functionality was developed. The spill to disk component saves correctly time-stamped raw data streams to the flash card in the VoxNet node. In the main WaveScope program, an extra *toNet* operator is included so that the main processor advertises and serves the raw data being gathered as a network stream visible to the supervisory co-processor (the Gumstix). A subscription client on the Gumstix receives the data over the wired network from the main ENSBox processing board, and marshals it to files on its flash card, properly annotating with global timestamps.

5.3.4.3 Platform Drivers and Services Layer

In addition to the reusable tools and components described above, there are many platform-specific elements used to support VoxNet. Many of these are hardware drivers (such as audio acquisition), diagnostic software, and glue components. The two most important features are described below.

Time Synchronization and Self-localization. VoxNet inherits a time synchronization service and a self-localization subsystem previously developed for the Acoustic ENSBox [14], and adds additional glue to integrate these features into the WaveScope engine. Reference Broadcast Synchronization [10] is combined with a protocol to propagate global time from a node synchronized to GPS [20]. Timestamps in VoxNet systems are converted to global time before being transmitted over the network or saved to persistent storage.

IP Routing. VoxNet implements IP routing from each node back to the gateway node using an existing user-space implementation of the Dynamic Source Routing algorithm (DSR) [19]. The networking layer sitting above TCP in VoxNet was developed in a single-hop context, and using DSR was the quickest and easiest path to moving this single hop communication to a multi-hop context.

This particular implementation dynamically installed routing entries into the kernel routing table, allowing nodes to automatically forward traffic on behalf of one another [27]. Although DSR can establish routes between any two nodes on-demand, it was only used to find and maintain routes between the sink and nodes.

This is because from an application perspective, the sink was the only end point which nodes required communication with. To maintain paths, nodes send a special *ping* message to the sink every minute. The gateway forwards traffic between the *Pseudo-IBSS* network and a normal wired or wireless LAN.

5.4 In-Situ Deployment

From the 8th to 18th of August 2007, the VoxNet platform was deployed and tested in-situ at the Rocky Mountain Biological Laboratory (RMBL) in Gothic, Colorado, USA, where biologists have been studying marmot behavior for 30 years. Four people in total were present at RMBL to deploy VoxNet in-situ: the three-man *VoxNet development team* and a biologist studying marmot behavior, using the Acoustic ESNBox purely as a data acquisition tool. Throughout VoxNet's deployment, the biologist provided expert advice and guidance where required, and assisted with the physical deployment. The involvement of a domain scientist during deployment was a critical part of the process. Observations and intuitive decisions made by the scientist helped inform the deployment sensing goals, and identified ways the system could be made easier to use.

In the months leading up to the deployment, it was not possible to perform full system trials because the Acoustic ENSBox nodes utilized in VoxNet were being used by the domain scientist at RMBL to record time-synchronized acoustic data of marmots. The deployment therefore represented the first end-to-end outdoor trials of VoxNet. Out of the ten-day period set aside for short deployments, a week-long on-site integration and partial system testing period was required in Colorado from the 8th to the 14th of August.

In total over the ten day period, four attended deployments of the VoxNet network were undertaken. During each attended deployment, a network of eight nodes was deployed when marmots are typically active, from mid-morning until early afternoon. The eight nodes were deployed in the same location over an area of 140 m by 70 m (2.4 acres), as shown in Fig. 5.6. Each node was between 40–50 meters away from its nearest neighbors.

The node positions were chosen by the biologist based on expert knowledge: the deployment area encompassed three well-known marmot burrow locations. The node positions were kept consistent over different deployment days by removing only the nodes at the end of each deployment, keeping stands in place.

The gateway node and control console were placed in turn at two sites during the experiments. The first was 200 m west of the deployment, in a car-park. The second was 100 m southwest of node 100 (see Fig. 5.6). When at the second position, all nodes were within one hop of the gateway node, meaning that multi-hop communication was unnecessary. Both positions were far enough from the deployment area that they minimized interference with the marmots.

During deployment, the marmot localization application as described in Sect. 5.2.3 was run on the VoxNet platform. A laptop connected via Ethernet to the

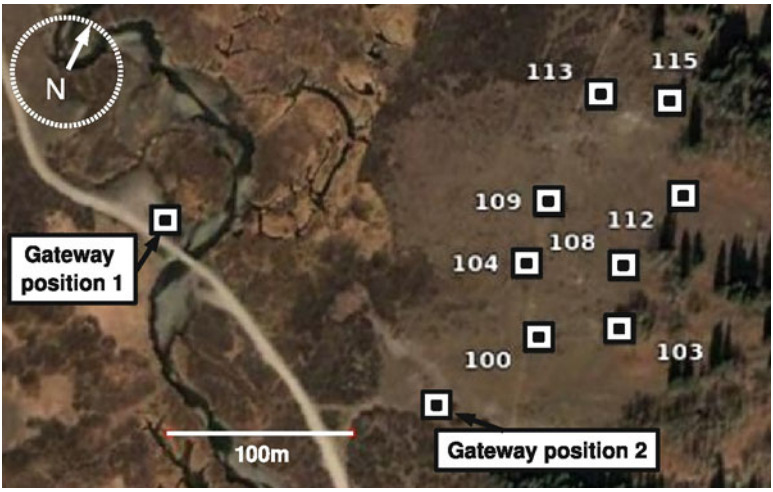


Fig. 5.6 The physical deployment of the eight nodes at RMBL. The nodes were deployed over a 140 m by 70 m area. The gateway and control console were deployed at two locations during the deployment, approximately 200 m west of node 104 in the car park, and approximately 100 m southwest of node 100 in an open, grassed area. Data SIO, NOAA, U.S. Navy, NGA, GEBCO, Image © 2010 DigitalGlobe © 2010 Google

gateway was used as the control console, as well as for the sink-side processing of the marmot localization application: the AML and position estimation. The control console was time-synchronized with the VoxNet nodes through the gateway, using a custom *time sync* service. The WaveScope shell (WSH) provided the interface to the control console, allowing control commands to be sent to the nodes from the laptop, as well as providing real-time updates of node status. Throughout the deployment, the WSH logged all detections and control stream messages that were generated by the nodes in the network as well as local debug and network data. Each node also recorded logging information for off-line analysis: link quality, network topology and application debugging output.

During the deployments, it was necessary to induce marmot calls to test the marmot localization application. This was achieved by a member of the deployment team walking slowly and quietly into the deployment area, and then making a sudden movement at the marmots. Usually, a single marmot would signal alarm and all would run back to their burrows temporarily. When many marmots were together, this behavior could result in multiple marmot calls.

5.4.1 Discussion of Problems Encountered During Deployments

Each of the deployments was intended to test the VoxNet platform and the marmot localization application in a realistic environment. It was important to understand the problems arising from running the localization application on-line, with real

marmots, and under the supervision of a domain expert. This allowed a comprehensive picture to be built of the necessary improvements for further design iterations.

In this section we summarize the general problems we encountered while deploying VoxNet. It was not always suitable to solve problems in-situ: laptops were running on batteries, and log data needed to be gathered, consolidated and analyzed. Additionally, development and controlled experimentation was required, both of which were not suitable to be performed at the deployment site.

Although the deployment area was within five miles of the accommodation, the time window in which to monitor marmots every day was limited. This meant a non-trivial software bug causing a return to the accommodation could effectively hamper the day's work. In addition, the biologist present throughout the deployment had to use the Acoustic ENSBox nodes to record continuous audio data streams on some of the days set aside for deployment. VoxNet could not be deployed on these days.

Weather conditions were hazardous for the gateway and control console: they were not waterproofed, so had to be protected from the rain. Additionally, too much sun made the laptop screen very difficult to read; a solution to this problem is shown in Fig. 5.7.

The screen glare problem was compounded by the fact that battery life at the control console was limited. Often, the screen brightness had to be reduced in order to conserve battery life. Several times, it was necessary to change laptops in order to keep experimentation running. This proved difficult for consolidation of data post-deployment. Ideally the battery life of the control console should match the lifetime of the nodes, or at least the intended duration of the deployment.

Fig. 5.7 Dealing with the problems of screen glare during in-situ experimentation

Each of the deployments identified a specific problem that had been overlooked or underestimated during the laboratory-based development of VoxNet. Therefore, with each deployment, the general operation of VoxNet was improved. The rest of this section describes each deployment in more detail.

5.4.1.1 Deployment One

The first deployment was made on August 14th, 2007 and involved running the full marmot localization application. There was light rain falling over the deployment area and as the gateway and control console were not weather-proofed, they were deployed at the car park (position one), inside the car.

VoxNet's multi-hop routing component was initially enabled for the deployment, but encountered a problem that had not been seen previously in lab or initial in-situ testing. The implementation of the DSR algorithm would find very long paths from node to sink of up to eight hops that would change often and appeared not to settle. It was not understood why this occurred, and time did not allow for a detailed analysis of the routing component. It is possible there was a bug in the implementation. To resolve the problem it was decided to limit the maximum length of a routing path between node and sink to be *three hops*. This change was implemented after the deployment, and worked for the remaining deployments.

The marmot localization application was run in single hop mode whilst still at the car park, as the weather had cleared by this point. When detections were triggered at each node, a data transport problem became apparent. It was found that nodes could transmit short log messages of around 30 bytes back to the sink to indicate that detections had been made, but detection data was not getting back quickly enough for detections to be fused together. This meant there was no suitable input for the localization algorithm which consequently was not invoked once. Instead, nodes were repeatedly getting disconnected from the control console as they tried to send data. The initial assumption was that this was due to poor quality of network links, so a decision was made to move the gateway and control console to position two, as shown in Fig. 5.6, to see if this would help with disconnection problems. Moving the gateway closer to the nodes did not seem to change the rate of disconnections, so it was decided to bring the nodes in to consolidate and analyze the data set that had been gathered. The total time from position one (car park) deployment to consolidation at position two was 25 min. Post-deployment, problems were found with the log data. The raw detection data messages sent from nodes had detection timestamps in the node's local clock rather than the network time. Additionally, the sink-side application recorded a log message when detection data from nodes arrived at the sink, but neglected to record the id of the node in this message, or the time it arrived at the sink. This meant the latency between detection at the node and arrival time at the sink could not be determined. This would have shown how long detections were taking to arrive and infer if moving the gateway did indeed help. The only data that could be gathered from the sink-side logs was that the detection

data being sent from nodes were varying in size, either 32 kB, 65 kB or 128 kB. The detection lengths should have all been 32 kB, so the extra data generated would have been filling up nodes' sending queues. This could have meant that data was not received at the sink because it was getting dropped, although there was no log information to support this. This configuration error was subsequently fixed to make sure any detection sent by the node was limited to be 32 kB in size. During the 25 min period, 86 detections were made by all nodes, but only 46 were received at the sink. This may be partly due to the fact that the application was stopped before nodes had cleared their data buffers, when feasibly they might have transmitted the remaining data.

Finally, the link information that was logged at each node showed that link quality improved between all nodes and the gateway after moving the gateway to position two. However, it was not clear from the gathered log information whether this had an effect on the latency of transmissions.

5.4.1.2 Deployment Two

The second deployment took place on August 16th, 2007. Starting at 11am, the VoxNet network was set-up in single-hop mode with the gateway in position two, running the marmot localization application for several hours. Subsequently, the application was stopped and VoxNet set into multi-hop mode. The intent was to gather marmot detection data in single-hop mode, and then troubleshoot multi-hop communication and data transfer latency as seen in deployment one, before finally starting the marmot application in multi-hop mode. Controlled experiments were carried out for 30 min, to be followed by the evaluation of the localization application. However, the multi-hop deployment had to be cut short due to poor weather when a storm interrupted deployment. As the rain started, the gateway was moved from position two to position one inside the car. The rain caused the marmots to retreat to their burrows, and the raindrops hitting the microphones and node cases caused the event detectors to constantly trigger. It was expected that the network would become heavily congested, and that nodes would drop data as their transmit queues overflowed. Over the course of just 436 s (7 1/4 min), 2,890 detections were triggered at a rate of around 6 per second network-wide. At the control server side, 342 detections arrived at the sink. This highlights a problem in the event detector that makes it susceptible to false detections (discussed further in Sect. 5.5.2). This episode gives an insight into the behavior of the marmot localization application running in VoxNet during heavy load. Each node dropped in the region of 90% of detection data queued to send. Despite these data losses, the system did not stop working, demonstrating that it could deal with network overloading in a graceful manner.

After the deployment, we discovered that the changes made to the VoxNet application to improve logging output had neglected to record the node id associated with the raw detection, meaning that the single-hop data gathered was unusable. Fortunately, the data would not have been useful under these conditions as during

rainfall marmots retreat back into their burrows, preventing further observation. In such a case, the scientist would either temporarily stop the application, or abort the experimentation for the day. In unattended deployments where the WSN is expected to be autonomous, it may be useful for the network to identify a change in weather conditions as a reason to temporarily stop processing data.

5.4.1.3 Deployment Three

The third deployment took place on August 17th, 2007. The weather was bright and sunny, so we decided to move the gateway node to position two. In the deployment, we attempted to manually write logs detailing marmot behavior: recording the rough locations where marmots called from and how many calls were detected. This was done in an attempt to relate events recorded by the system to real marmot events, an aspect that had been neglected in the two previous deployments.

Only seven of the eight nodes were used as the eighth node was experiencing intermittent problems causing it fail to respond to network communication. We believe this failure was caused by the rainfall in the previous deployment day. The nodes were deployed for 2 h 10 min. Within that period, the application was run in multi-hop mode for around 1 h 25 min, and in single-hop mode for around 45 min.

During the deployment, three periods of marmot activity were induced by members of the deployment team. The first period of activity lead to a set of eight detections over a two-minute period approximately where the sixth, seventh and eighth alarm calls were from different marmots but very close together. Figure 5.8 shows a graph of detection time from first detection on the x axis vs node id on the y axis. The individual marmot calls are evident on the graph as tightly grouped, vertically aligned detections across several different nodes. Notably, the sixth, seventh and eight calls are situated around the 80 s mark, where there are indeed multiple detections from each node within a small space of time. After this point, the control console was swapped over to a different laptop and the application restarted.

The second period of activity was a bout of twenty-eight calls made by one marmot as a member of the team slowly approached it. Calls were made at intervals of roughly one second. Photos were taken, but unfortunately the raw data corresponding to the set of detections was not recovered from the control console post-deployment, and was inadvertently overwritten during the next day's deployment. Attempts made to induce another bout were unsuccessful.

The third period of activity yielded a set of six marmot calls, but the data corresponding to these were lost in the same manner as the marmot bout. To further frustrate, the version of the sink-side application running at the control console had not been updated to reflect the logging changes from Sect. 5.4.1.2, where the node id was recorded along side the AML and detection data. Thus, the log data taken by the WSH and the sink-side application were not sufficient to draw a graph similar to Fig. 5.8.

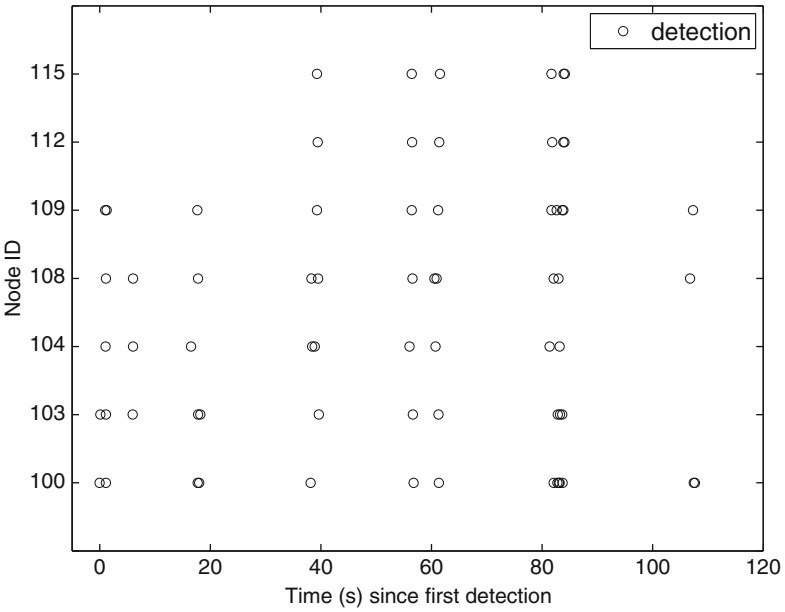


Fig. 5.8 A set of events taken from deployment three, dated 17th August, 2007. The x axis shows time since the first detection in seconds and the y axis shows node id. Each data point represents the time a detection was triggered at a node

5.4.2 Deployment Four

The fourth deployment took place on the 18th of August, 2007. The marmot localization application was run for just over 2 h whilst marmot alarm calls were induced by a team member. The gateway and control console were placed at the car park, and the network was started in single-hop mode. A larger antenna was available to use that provided one-hop communication to all nodes from the car-park gateway location.

Whilst the manual data logs about marmot behavior introduced in deployment three were reasonably helpful, it was decided that one of the nodes would record a time-synchronized audio stream in parallel with the marmot localization application. This recording could subsequently be used as a reference stream for the events detected by nodes during the deployment. The resulting data set of raw data files totaled 3.8 GB – four channels of 16-bit audio sampled at 48 kHz for around 2 h.

Within a 2 h period (7,194 s), a total of 683 detection events were sent to the sink by nodes in the network. Just 5 out of 683 detections were dropped for a 99.3% success rate. Although 100% data reliability was expected, the data drops were observed to be due to the overflow of the network buffers (512 kB per stream), which dropped new data during periods of network congestion and high detection rates (i.e. *tail dropping*). This indicated that the arbitrary buffer sizes chosen for streams were too small for the volume of detections that each node was experiencing.

5.4.3 Summary

This section has described a number of issues which affected the deployment:

- *Data logging issues* the logging data being gathered was not always sufficient to isolate problems or allow for off-line processing.
- *Multi-hop routing* the multi-hop routing implementation caused enough problems that it was necessary to run the system in single-hop mode to isolate other problems.
- *Time constraints and weather* there were only small time windows each day when marmots were active, and poor weather meant marmots did not surface from their burrows during these times.
- *Data transport issues* the volume of data had an effect on how quickly data could be transported to the sink.
- *Phenomena issues* marmots sometimes had to be encouraged to chirp, in order to stress-test the system.

Although we experienced many problems throughout the deployment process we considered it to be a success. We were able to identify and debug many problems, and managed to record some useful data about VoxNet's in-situ operation.

The first iteration of several real-life deployments is often marked by low data yield and quality, as well as debugging and operational difficulties. This trend is documented in many other systems including volcano monitoring [31], redwood tree micro climate monitoring [28], a precision agricultural monitoring [21] and soil contaminant monitoring [26]. At a superficial level, the source of these problems is lack of adequate preparation. However, trying to predict the behavior of even the simplest WSNs before deployment is very difficult because it is practically impossible to think of every potential failure point of a system before deployment. This is partially due to the fact that some obscure failure conditions are only revealed during in-situ operation, and partly because it is hard to know what failure modes are most likely to occur, i.e. some may not be predicted at all but may turn out to be the most important. In many cases, it is only through experiencing unpredictable, obscure problems that the deployment and operation of a system can be made more reliable and robust.

However, comprehensive preparation is difficult for first system iterations, due to the physical difficulty of deployment, as well as prediction of corner cases which will cause system to stop working, but only become apparent in unpredictable environments. In general, a rule of thumb favored by the author is that everything that can go wrong, will go wrong. To combat this, it is important that the WSN deployer be prepared for unpredictable eventualities by scheduling enough *time* for the deployment and making sure the system is suitably instrumented for gathering debugging and log data. Although only four days were used for deployment here, all of the ten days available were used to improve VoxNet's reliability and robustness.

This section has shown how important it is that a deployed system always gathers debugging and logging data so that (a) if it fails, the location of the problem in the code can be quickly identified and (b) performance information can be derived. Post

deployment analysis of system log data is a vital part of fixing or recreating problems in controlled environments. In some cases, these changes may be attempted during the time allotted for deployment, but in other cases the system modifications must be introduced post-deployment, back in the lab. For the latter case, it is not always clear exactly which logging data will become useful during post analysis. The goal should be to log as much informative data as possible without severely compromising the running of the system.

The next section discusses an issue found to be problematic during deployment: latency of data transfers. We focus on this because it has a general appeal to other high data-rate applications.

5.5 Factors Affecting Timeliness

When a WSN-based system is required to provide feedback to a user in the field, timeliness is an important consideration. The user or the network may need to act on the feedback to enable other observations or actuation. In the case of the application at hand, the scientist wants to know where the marmot is quickly so that a picture may be taken. If the animal moves before the position estimate is made then the feedback is no longer timely, and most likely not useful.

Many monitoring applications which run in an on-line manner are event-based in nature – the system must react to the occurrence of an event, be it a structural fault, gunshot or animal call. When the phenomena being sensed must be sampled at high rates, the amount of data generated means that some data processing must be performed locally. Therefore, maintaining timeliness is a difficult trade-off. It is affected by the requirements of the motivating application, the speed and reliability of the wireless communication channel between nodes, and the processing power available locally at each node. Since at least one of these factors - usually the wireless communication - is dynamic and can change radically over a short time period, it is important to address this trade-off dynamically.

In the marmot localization processing chain, there are three contributors to the end-to-end latency of position estimates and thus timeliness in the system: node latency, network latency and sink latency. Node and network latency are due to on-node event detection and the transfer of that detection data to the sink. Sink latency is due to the processing of detection data through the AML and data fusion algorithms, and the graphical presentation to the user.

During deployment, we observed that network transfer was by far the largest and most variable contributor to end-to-end latency in the system, ranging from tens of milliseconds to several seconds. The on-node event detection ran in real-time, and the processing load on the sink was nominal, consuming on the order of tens of milliseconds.

This section looks at the factors which induce position estimation latency from an application-oriented perspective. The rationale for this is as follows: the network latency an individual node experiences is affected by the amount of data the network

must carry at any given point. This in turn is affected by the volume of events that are being triggered at each node. These events may be real, or they may be false positives. Solutions can address this problem locally in two ways. First, from an individual node's point of view: how do I reduce the amount of data I send based on local information? Second, from a network-wide point of view: how can the overall amount of data sent be reduced based on network information?

5.5.1 Event Frequency

Recall from Sect. 5.2 that how frequently marmot calls are made is reasonably well understood: individual marmots call relatively infrequently, but that several marmots may call within a short period, or a single marmot may produce a *bout*. However, in the context of the system, event frequency refers not only to the rate at which the animals call, but also to the rate at which the event detector running on the nodes is triggered. In an ideal scenario, the trigger rate and the call rate would be identical. In practice, false positives trigger a node's event detector, as discussed later in this section. Nonetheless, as the frequency of events being detected by each node in the network increases, it follows that the amount of data that needs to be sent back to the sink will increase. At each node, outgoing data is queued, meaning it will take longer to transfer the most recent detections if there is already data waiting to be sent.

To show the effects of event frequency on data queues, and the subsequent transmission times, we consider an example from a data trace of actual detection events from the system running in-situ, at RMBL. Seven of the eight nodes were used (node 108 was not used), with each node one hop from the sink.

Figure 5.9 shows the effect of a large number of detections across the network in a short period of time. The data shown on the graph is for a period of 3 s, when 19 detections were triggered across various nodes. Each data point represents a detection, where the x axis is the detection time, and the y axis is the number of events that were queued network-wide when the detection was made. The graph shows how data can quickly accumulate in the network. Ultimately, this shows that the event frequency, and hence local and aggregate queue size will have an effect on the time taken to transfer data to the sink.

5.5.2 False Detections

As previously mentioned, false positives can increase the end-to-end latency of the localization system. There are two ways in which false detections affect the on-line operation of the system. First, the event detector at each node can be triggered by events which are not marmot calls; these are false positives. Since each node will send the data corresponding to any detection it makes to the sink, unnecessary data is

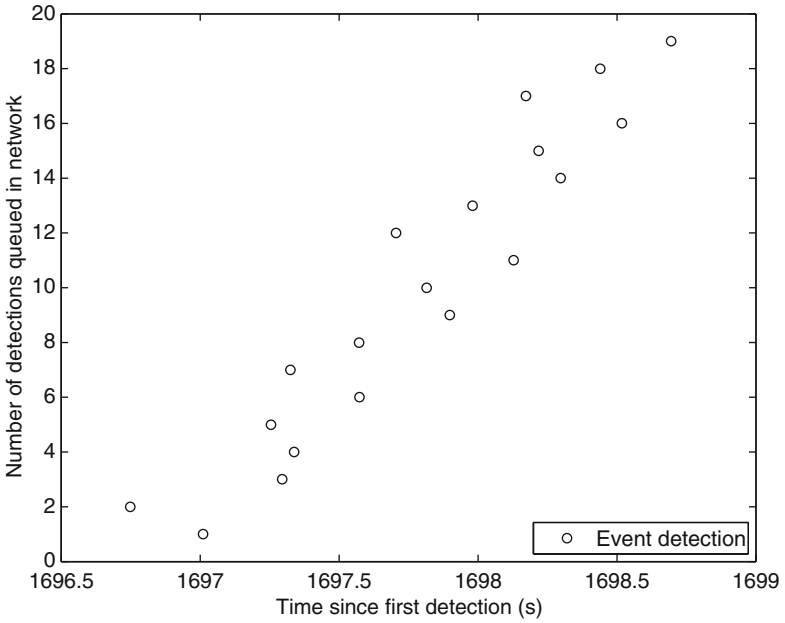


Fig. 5.9 A graph of detections being sent or queued vs. detection time from data gathered at RMBL. Each point on the graph is a detection made by a node, where the x axis is the time since the first detection in the deployment was made and the y axis is the aggregate instantaneous number of events either queued or being transmitted across the *whole* network

being sent. For example, it was noted that hand claps could trigger the event detector. Network-wide, a loud enough clap could be picked up by several nodes, representing a network-wide false positive. Furthermore, uncorrelated weather events such as rainfall and strong winds were observed to trigger the event detectors frequently, as shown in Sects. 5.4.1.2 and 5.4.2.

Second, if non-marmot events trigger several nodes in the network simultaneously, the data generated may still be processed as though it were a proper animal call. In this case, unnecessary processing is performed at the sink that results in a useless output.

One approach to addressing the problem of false positives is to attempt to reduce them locally, at the node. A suitable candidate for this approach is the application of automated classification techniques. Events detected using the energy-based detection technique could be submitted to a more rigorous examination to determine if they require further processing, for example through the AML. Work has been presented in the literature on the use of Hidden Markov Models (HMMs) to classify species of acorn woodpecker [29] and other types of bird [30]. HMMs are a popular approach for automated speech recognition (ASR) of human voices. In [30], the HMM is trained against reference bird calls to create *observation vectors*, which can be used to classify the signal as it evolves. The *observation vectors* can be created using techniques such as building Mel-Ceptstrel Coefficients (MFCCs), Linear

Predictive Coding (LPC) or Artificial Neural Networks (ANNs) [18]. Through experimentation, MFCCs are seen to be more suited to recognition of bird song than LPC, due to sharp transitions that are present in bird calls [30]. Whilst the evaluation of an observation vector in a trained HMM is not computationally complex, the creation of the observation vector may require significant processing.

Aside from its uses to aid event detection, classification is a generally important component of the higher-level sensing goal for smart bio-acoustics sensing: automated census for different species of animals and birds. It is clear that with suitable consideration for real-time operation on an embedded node, classification techniques could be a good candidate to help reduce false detections; however, these techniques have not been implemented or tested on the VoxNet nodes to date.

5.5.3 A Case for Dynamic Processing

It is already understood that high data rate applications preclude the streaming of raw data from nodes to a sink. However, it may also be the case that the filtered streams of data produced by nodes cannot be delivered in a timely manner either. Examples of this have already been shown in this section, where events are generated frequently, some of which may be false positives. Since the effect of event generation may have a dynamic effect on the network latency, it is important to have a dynamic means to adaptively adjust based on the current state of the network.

In the case when the network is congested, it makes most sense for a node to consider processing some of its data locally. In some cases it may take less time for a node to perform some local processing rather than gamble with high latency on a congested network.

It was established in [3] that for the localization application, the angle of arrival algorithm (AML) could be processed locally on a VoxNet node. This processing takes longer than it normally would at the sink – 2.4 s vs 0.08 s, but results in significantly less data that must be sent to the sink – 32 kB for audio, 800 bytes for the AML vector. In [3], a simulation was carried out based on empirical measurements from the field to determine whether there were real conditions under which it was beneficial to process the AML locally. Figure 5.10 is a graph based on the results in [3]. The graph is organized by number of hops between node and sink, and shows the aggregate time for both AML processing and data transmission. The data was gathered by measuring the time taken for nodes to send either 32 kB or 800 B in a three-hop network. For the given data set, there is a trade-off point that lies between two and three hops from the sink; beyond this point, it makes sense for a node to process data locally if possible [3].

This shows that if a trade-off can be drawn between locally processing and network transmission, the general solution to the trade-off can be found by determining *when* to enforce this dynamic local processing.

This section discussed the application-related issues found during the deployment of VoxNet which affected the end-to-end latency. Specifically, these issues were focused around the generation and transmission of unneeded data by nodes.

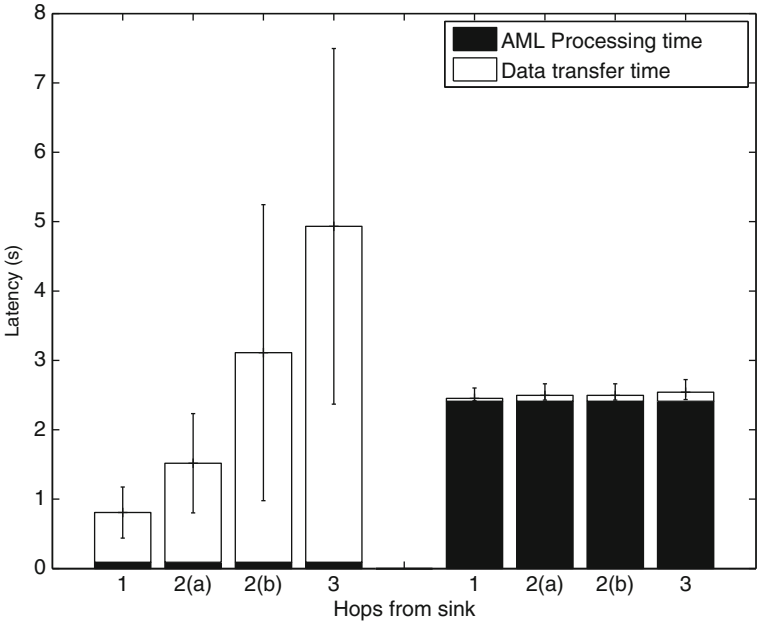


Fig. 5.10 Simulation results showing the difference in end-to-end latency between processing at the sink and sending raw data (*left hand side*), and processing at the node and sending the AML result to sink (*right hand side*). Each bar contains two parts: the AML processing time (*lower part*) and the data transfer time (*upper part*). The x axis shows the number of hops from the sink, where 2(a) and 2(b) are different branches of the routing tree

This unneeded data was a result of false detections, or a rapid build up of events over a small period of time. Furthermore, the sending of uncorrelated detections to the sink presents unnecessary data transmission.

The following two sections present two approaches to reduce end-to-end latency and thus improve timeliness based on application related, deployment-observed problems. These are Lazy Grouping, which provides network wide data filtering and collection, and an Adaptation policy which allows a node to decide whether to process data locally or send to the sink based on the dynamic state of the network. Even though they are evaluated with respect to the marmot localization application, in the context of high data-rate, event-based applications, these approaches are generally applicable.

5.6 Lazy Grouping

An important aspect of meeting the goal of only sending data that supports the application's overall aim is to provide some way of deciding if data is *useful* before sending it. In the context of the motivating marmot localization application, data

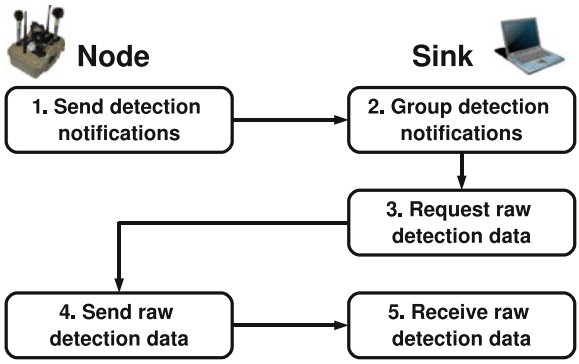


Fig. 5.11 The flow of Lazy Grouping's on-line grouping (1, 2) and data collection stages (3, 4, 5)

sent by the nodes is only *useful* if it is being used as part of a position estimate calculation. The *Lazy Grouping* algorithm is a potential solution to this problem. This approach is inspired by Bentley's *lazy evaluation* strategy of never evaluating an item until it is needed, thus avoiding evaluation of unnecessary items [8]. The rest of this section describes Lazy Grouping in detail and justifies the approach through simulation.

The overall flow for Lazy Grouping is shown in Fig. 5.11. Lazy Grouping operates in two phases: detection grouping and data collection. When the event detector of a node n is triggered by an acoustic event of interest, a detection event d with locally unique sequence number s is created. Each detection $d_{n,s}$ consists of a segment of audio a corresponding to the detection, and a timestamp t indicating the start of the detection. The node saves this detection event to a local buffer B and also sends a *detection notification message* d_{notify} to the sink, containing the node's ID n and the timestamp of the detection t.

The sink receives detection notifications from nodes in real-time, and continuously organizes them into groups g according to two rules: *identity* and *temporal* consistency.

Identity consistency asserts that node IDs within a group are unique. If $I(x)$ gives the node ID associated with detection x then,

$$I(x) = I(y) \Rightarrow x = y \tag{5.1}$$

for all $x, y \in g_j$.

Temporal consistency asserts that the timestamps in all detections in a group are within a certain time window of each other; this is called an *uncertainty factor*, and denoted as u. The center of the window is determined to be the average timestamp value of the current detections in the group $m(g_j)$. The logic followed when trying to add a new detection notification to a group is:

1. If the absolute difference between the detection timestamp and the current mean of the group is less than the uncertainty value ($|T(d_i) - m(g_j)| \leq u$), then the detection should be added to the current group.

2. If the absolute difference between the detection and the current mean of the group is greater than the uncertainty value ($|(T(d_i) - m(g_j)| > u$) and the difference is positive ($((T(d_i) - m(g_j)) > 0$), then a new group g_{j+1} should be started (because the event happened more recently than the current group and is not part of it).
3. If the absolute difference between the detection and the current mean of the group is greater than the uncertainty value ($|T(d_i) - m(g_j)| > u$) and the difference is negative ($((T(d_i) - m(g_j)) < 0$), then the event happened before the current group g_j.

Each group g_j has a watchdog timer w which is used to trigger its data collection phase, conditional on there being at least x detections in group (enough to perform a position estimate). In the data collection phase for a group, the sink sends out requests to nodes for the transmission of full detections $d_{n,s}$. Upon receiving a request, a node locates and extract the relevant detection data from its local buffer and sends it back to the sink. When the sink has received all of the detections for a given group it triggers the next stages of the processing in the position estimation data flow: AML, followed by data fusion.

5.6.1 Lazy Grouping Experimentation

Two simulations were carried out, using data gathered during the deployment period at RBML, in order to validate Lazy Grouping. The first simulation used the RMBL in-situ data trace with an off-line implementation of Lazy Grouping, in order to determine the algorithm's potential for reducing data transmissions. The second simulation used the same in-situ data trace in conjunction with a set of data transfers gathered from a controlled experiment from the same network configuration at RMBL to observe the relative reduction in latency in gathering data corresponding to groups. All of the data gathered and used in these simulations was from a one-hop rather than a multi-hop network, thus latency from multi-hop transfer is not considered.

5.6.1.1 Experiment One: Data Reduction

The goal of the first experiment was to quantify the reduction in data transfer that could be possible using Lazy Grouping. It was expected that Lazy Grouping would provide significant benefit when applied to the RMBL data set. The experiment was performed using simulation. To achieve this, an off-line version of the Lazy Grouping algorithm was run over a real event stream gathered at RMBL. The off-line version of Lazy Grouping only simulated the creation of groups, not the requesting and transferring of data – this was simulated in the second experiment. In the RMBL data set seven nodes all one hop from the sink sent back any events that were

triggered by their event detectors regardless of whether they were actually marmot calls or not. For each detection that arrived at the control console, the sending node ID, global time of detection and time taken to transfer the data were recorded. In this data stream, the amount of data sent per detection was 128 kB, rather than 32 kB, due to a misconfiguration of the system which was not possible to rectify during the deployment. This disparity in data size does not affect the analysis presented here. The amount of detection data transferred during the actual in-situ experimentation was 44.125 MB, corresponding to 353 events. When the off-line Lazy Grouping algorithm was run over the data set using an *uncertainty factor* of 440 ms, and a minimum group size of 3 detections, the amount of data that would have been requested by the sink was 15.75 MB, corresponding to 126 detections. This represents a 64% reduction in data transferred – a positive indication of the data reduction potential of the Lazy Grouping approach. In total, 24 groups were made: 14 groups with 3 detections each, 16 groups with 4 detections each and 4 groups with 5 detections each. No groups larger than 5 were found.

Figure 5.12 shows a window of the first 18 min of the RMBL data trace, with time since the first detection on the x axis, and node id on the y axis. Detection events are marked as dots and events which have been grouped together are marked with squares. To aid understanding, the mean timestamp of each group is shown as a vertical line. This shows the filtering effect of the Lazy Grouping approach and

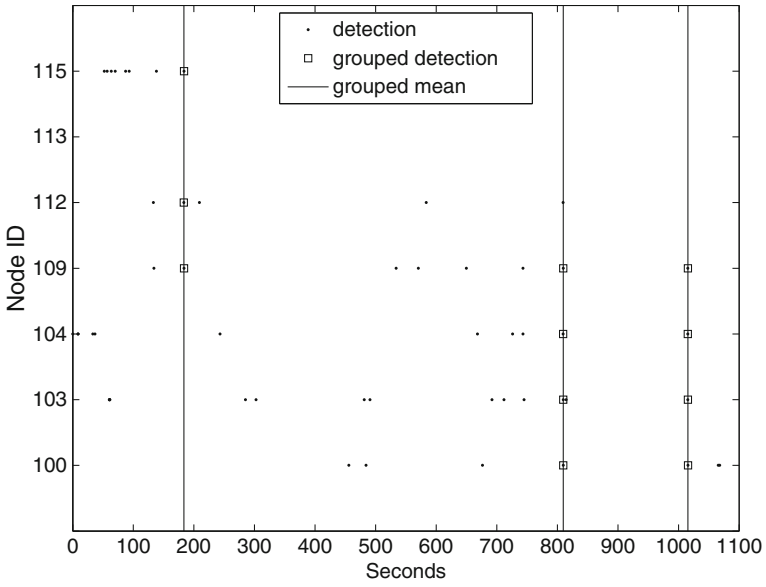


Fig. 5.12 A plot showing the result of grouping detections to decide which data should be sent to the sink by nodes. Each point on the plot is a detection, made by a node (shown on the y axis) at a certain time (shown on the x axis). Vertical lines show the mean of the group to which detections marked as squares belong to. All other detections are discarded

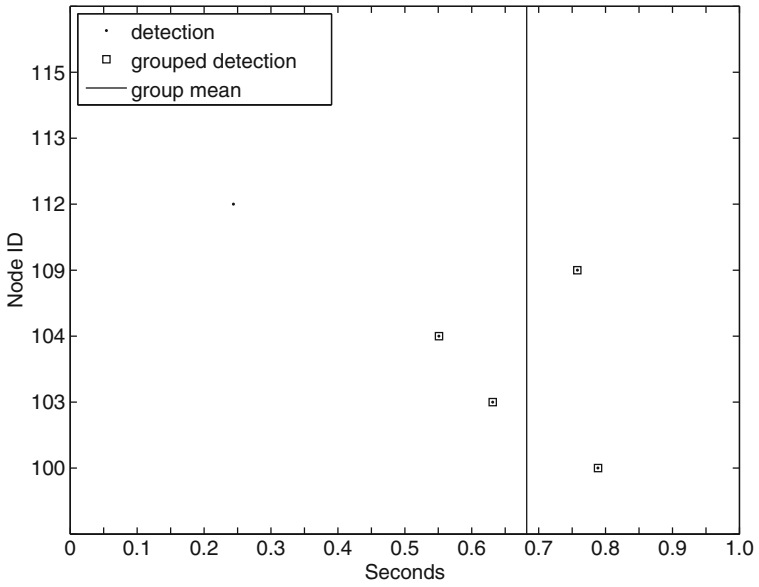


Fig. 5.13 Rejection of detections that are not close enough to the mean of the event group. In this case, the detection from node 112 was more than 440 ms away from the mean established by the detection timestamps from nodes 109, 104, 103 and 100

gives an indication of the number of detections that can be disregarded based on the temporal and identity consistency rules. Figure 5.13 shows a particular group of detections from Fig. 5.12, highlighting how the temporal consistency rule rejects detections which are not within the *uncertainty factor* of the group's mean.

5.6.1.2 Experiment Two: Latency Reduction

The goal of the second experiment was to simulate the data collection phase of Lazy Grouping to demonstrate the potential for end-to-end latency reduction. Recall that the Lazy Grouping data collection phase consists of the following steps: (1) receive detection notifications from nodes, (2) send out requests for data and (3) receive raw data responses from nodes. More formally, assume that a group g_i containing $n(g_i)$ detections is created by the Lazy Grouping algorithm.

Assume $\ell_g = \{g_i\}_{i=1}^{n(g_i)}$ is the set of latencies for collecting data from all nodes in a group g, where each element of ℓ_g is determined by

$$\ell_g = q_i \cdot (d_{raw,i} + d_{det,i} + d_{req,i}) \tag{5.2}$$

where q is the number of detections the ith node had waiting to be sent in its local queue before the current request, d_{det} is the time taken for the ith node to send a

detection notification and d_{req} is the time taken for the sink to send a data request to the ith node. The overall latency $l(g_i)$ to gather the data from the group g_i is therefore

$$l(g_i) = \max_g \ell_g \tag{5.3}$$

The simulation of Lazy Grouping's data collection process as discussed above was implemented in Matlab. The simulation iterated through the list of detections belonging to the groups created by the Lazy Grouping algorithm in experiment one. The list was ordered by detection time. For each detection, q_j was inferred by counting the number of detections node j had made as of the current position in the list and subtracting the number of detections the sink had received; this could be inferred from the reception time logged for detection.

The other parameters needed to calculate l_i (d_{raw}, d_{det} and d_{req}) were determined from empirically gathered data. Eight 128 kB and twenty-two 800 B simultaneous seven-node data transfers were requested during controlled experimentation in a one-hop network at RMBL. For each 128 kB or 800 B data request, the start of the first transmission and the arrival of the last transmission were recorded, giving the total time to gather 128 kB or 800 B from all seven nodes. The mean of the twenty-two 800 B transfer times were used to model both d_{det} and d_{req}.

For comparison, the time taken to gather the groups in the in-situ data trace were calculated, simulating what would happen if the on-line grouping algorithm was simply grouping data as it arrived at the sink, rather than requesting it. To do this, the earliest detection time and latest arrival time of the detections in each group were found. The results of the Lazy Grouping simulation and the latency from the normal event stream are shown in Fig. 5.14. The bars represent mean time taken to collect the data for the group, with error bars representing the min and max times.

There is a clear improvement in the time taken to gather a group of data using Lazy Grouping. The best improvement in data collection time is seen when collecting data for groups of five detections: the average time goes from twenty seconds to five seconds, a reduction in latency of 75%. In general, the trend indicated by both sides of Fig. 5.14 is that larger clusters take longer to gather. This is particularly notable in the non-Lazy Grouping simulation. However, the latency increase is likely caused by an increased amount of data being sent, due to nodes' event detectors being triggered more often.

This is exactly the kind of situation that Lazy Grouping can address, and it does so by ensuring that only useful data is sent.

5.6.2 Discussion

It is clear that applying filtering without considerable in-field experimentation or consultation with the domain expert in choosing the relevant parameters would not be sensible. It is intended that features such as Lazy Grouping are made available to allow the domain expert to apply some knowledge to tune application

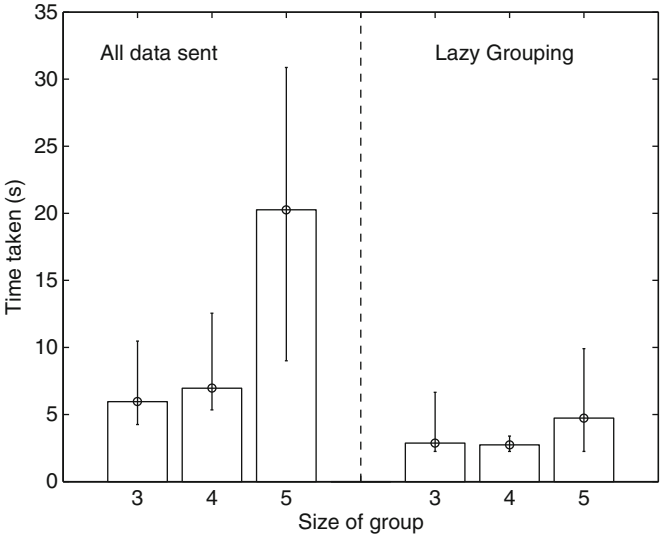


Fig. 5.14 A graph showing the potential benefit of Lazy Grouping's data collection process in simulation. For comparison, the time taken for the data to arrive at the sink for the original data trace is shown. The error bars represent the min and max times for each transfer, and are ordered by group size

related-parameters to improve application (and network) performance in-situ. During this tuning, it is important not to lose data that may be useful to the domain expert – this is the reason that the *spill to disk* functionality exists, where a full data set can be gathered for offline analysis, but not preclude any on-line, in-field analysis that may be of great interest to the domain expert (see Sect. 5.3.4.2).

The Lazy Grouping approach is suitable in the marmot localization application for reducing the amount of data that is sent over the network when it is likely that event detectors will yield large numbers of uncorrelated detections, which can be assumed to be false positives. If a nodes event detector is triggering due to many false detections, this will cause the transmission of significant amounts of useless raw data back to the sink (useless in terms of performing marmot localization).

This evaluation of Lazy Grouping did not consider multi-hop data transmission. The most apparent effect in using Lazy Grouping over multiple network hops would be the increase in latency for transmission of both detection notifications and raw detection data.

In general, Lazy Grouping is likely to be suitable for event-based applications where simply sending data about every event occurrence still creates too much data to send over the network. This will likely happen when large numbers of events occur, or the event detection on nodes is noisy, possibly due to a lack of processing resources.

The advantage of this approach is that the processing required is relatively lightweight. Lazy Grouping uses closeness in time to determine whether a set of

detection events are correlated, i.e. refer to the same event. As presented here, Lazy Grouping does not prioritize collection of data, and does not distinguish between actual marmot calls and false positives.

A more fully-featured system might utilize on-node event classification to discriminate marmot calls from false positives before detection notification. It is clear that this would help reduce the amount of false positive, but this should be traded off against the potential increase in processing requirements.

5.7 Adaptation Policy

Section 5.5.3 made a case for local processing of the AML algorithm when the network latency is higher than the cost of processing the related data locally. This section presents a node-level approach to determining *when* this decision should be made, known as an Adaptation policy.

The Adaptation policy uses data that is available from the running system to make the choice to process locally or not, by means of a dynamic estimator. The policy is as follows:

If $t_{aml} \cdot n(q_{aml}) < \hat{\ell}(d)$ then the node should process locally

Here, t_{aml} is the time to taken process an AML locally on a node, $n(q_{aml})$ is the number of raw detections waiting to be processed locally and $\hat{\ell}(d)$ is the estimated time that the current detection will take to transfer over the network $\ell(d)$.

Each node must record the latency of all raw data transfers it makes: the latency of these previous transfers is used to estimate how long the next transfer will take. The node determines how long a transfer took by recording its start time locally and subtracting this from the timestamp recorded by the sink when the transfer was completed; the completion timestamp is provided to the node as part of the sink's acknowledgement that the given message was delivered successfully.

Two different implementations of the dynamic estimator function to estimate $\ell(d)$ are presented in this section: $\hat{\ell}_1(d)$ and $\hat{\ell}_2(d)$. Both of these implementations use the time taken for previous transfers to predict the next transfer. $\hat{\ell}_1(d)$ uses a goodput measurement, which is the measure of useful data received divided by the time taken to transmit. $\hat{\ell}_2(d)$ uses a *pseudo-goodput* measurement, which measures the time elapsed from the detection being placed on the outgoing queue until it was received at the sink. Both estimators for $\ell(d)$ determine their predictions as the amount of data that must be sent on the path from node to sink, divided by the goodput (or pseudo-goodput) estimate, which is calculated from previous transfers. The estimate of $\ell(d)$ using $\hat{\ell}_1(d)$ is

$$\hat{\ell}_1(d) = n(q_{send}) \cdot D/g_k \tag{5.4}$$

where $n(q_{send})$ is the number of detections queued to send including the current detection, D is the amount of data to be sent, and g_k is the mean goodput of the last

k transfers the node has made. The amount of data to be sent D is given by

$$D = h \cdot s(d_{raw}) \tag{5.5}$$

where h is the number of hops the node is from the sink and $s(d_{raw})$ is the size (in kB) of a raw detection, so that the forwarding of data over multiple hops is considered. The mean goodput g_k is calculated from the last k transfers on a *per-node* basis as

$$g_k = \frac{1}{k} \sum_{i=x-k}^{n} \frac{D_i}{t_i} \tag{5.6}$$

where x is the number of transfers sent so far, t_i is the time taken for the ith transfer and D_i is the total amount of data that was sent over the path between node and sink for a given transfer, as in Equation 5.5. This allows the node to use an arbitrary number of previous transfers to help in predicting the time that the next transfer will take.

The pseudo-goodput estimate $\hat{\ell}_2(d)$ for $\ell(d)$ is

$$\hat{\ell}_2(d) = D/p_k \tag{5.7}$$

where p_k is the mean pseudo-goodput derived from the past k raw data transfers. The pseudo-goodput differs from the mean goodput g in that it includes the time spent on the message queue q_i. The mean pseudo-goodput p_k is

$$p_k = \frac{1}{k} \sum_{i=x-k}^{n} \frac{D_i}{q_i + t_i} \tag{5.8}$$

where q_i is the time each detection spends on the message queue. The pseudo-goodput estimate $\hat{\ell}_2(d)$ may be easier for the higher level system to determine, because $q_i + t_i$ is determined by timestamping when the data was queued to send and when the transfer was completed, rather than when the data transmission actually began.

5.7.1 Evaluation

The dynamic estimator was evaluated as a proof of concept in simulation, using realistic data traces gathered in-situ. Simulation was carried out using Matlab based on data traces gathered from in-situ experimentation. This section describes the approach used to gather the data traces, how they were used in simulation, and the analysis of the results obtained.

The evaluation of the Adaptation policies through simulation had two goals. The first evaluation goal was to quantify the correctness of choices made by the policy.

The second evaluation goal was to determine the accuracy of the dynamic estimators $\hat{\ell}_1(d)$ and $\hat{\ell}_2(d)$. Accuracy refers to how closely the predicted and actual data transfer latencies were. Accurate prediction of transfer latency allows the Adaptation policy to be generally suitable in making local processing decisions where the local processing operation takes an arbitrary amount of time. In addition to the latency prediction, the factors which could affect the accuracy of the transfer latency predictions were also evaluated: the amount of previous transfers to use in latency estimation, and the amount of time a local processing operation took.

5.7.2 Gathering Empirical Data

In total, we gathered three data traces: one from live system operation with real event detectors running (taken from RMBL in 2007), and another two traces from controlled outdoor and indoor experiments at CENS, UCLA, using the `test` functionality of the WSH discussed in Sect. 5.3.4.2.

All experiments used seven nodes, and a gateway with laptop attached to act as the sink and host the control console. For each experiment, the network topology was consistent for the experimental period, shown in Fig. 5.15. Table 5.1 summarizes the data sets. It should be noted that the multi-hop topology used in the CENS

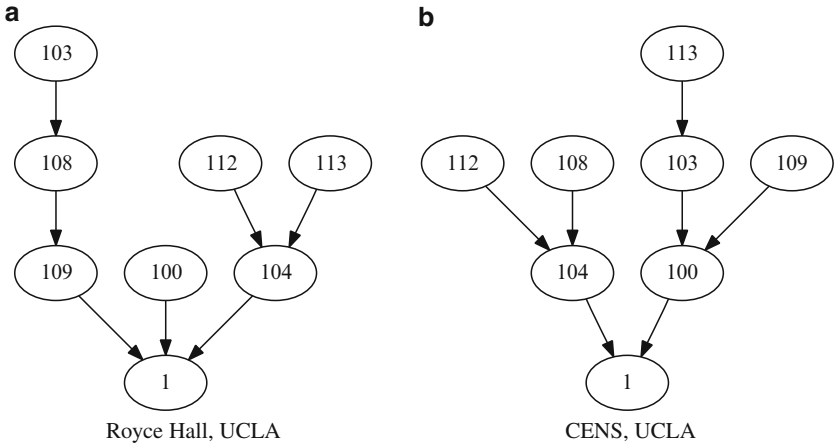


Fig. 5.15 The multi-hop routing trees used in experimentation. Note that the RMBL, CO experimentation was single-hop, thus is not shown here

Table 5.1 A description of the data sets gathered, including location, network topology and data source

Experiment location	Topology	Data source	Per-detection size
RMBL, CO, 2007	single-hop	in-situ events	128 kB
Royce Hall, UCLA, 2008	*dynamic* multi-hop	request plan	32 kB
CENS Lab, UCLA, 2008	*fixed* multi-hop	request plan	32 kB

Laboratory was created by using MAC address filtering at each node to enforce a particular multi-hop topology, that is the DSR component could *only* find the particular multi-hop topology enforced.

Both the Royce Hall and CENS experiments were divided into phases, where in each phase an increasing number of nodes were requested to simultaneously send data, starting with one node and ending with all seven. At each phase, there were 21 data requests sent, and each node in the network was requested to send data an equal number of times, so that the latency experienced by an individual node could not dominate the data set. The time in between each data request was either 1 or 5 s, to simulate event patterns of varying frequency; each experiment was run twice.

For each data transfer in each experiment the node ID, detection time, time the detection was placed on the queue and time the sink received the full detection message were recorded. The size of the message and the number of hops the node was from the sink were also recorded. In addition to these raw measures, the goodput, number of detection events in outgoing message queue and aggregate number of detection events in the network were derived. These traces provided enough data to enable simulations of the Adaptation algorithms to be run off-line.

5.7.3 Simulation

To simulate the operation of the Adaptation policy, the data transfer records were duplicated into two lists: one sorted by event detection timestamp to simulate the order in which they were detected, and the other sorted by detection arrival timestamp to simulate the order in which the transfers were received at the sink. Therefore, for any event detection record, it was possible to determine all of the transfers that had previously occurred for that particular node, allowing computation of the k most recent transfer latencies. For the simulation $k = 3$ was used, as it was found to be the best trade-off between historical transfer latency and latency prediction accuracy. The detection list was iterated through, and for each data transfer record, both latency estimators $\hat{\ell}_1(d)$ and $\hat{\ell}_2(d)$ were used to predict the transfer latency.

5.7.4 Performance

For each detection event that is triggered at a node, the goal of Adaptation is to correctly decide whether it will take longer to process raw detection data locally or to send it to the sink. Therefore, the best way to evaluate the dynamic estimator is to compare the processing choices made with the choices that ensure minimum system latency, referred to as the *correct choices*. The Adaptation policy chose to process locally if the estimated time taken to transfer $\hat{\ell}(d)$ was greater than the time taken to process the raw detection through the AML algorithm locally t_{aml}, assuming no

Table 5.2 The relative performance comparison between the $\hat{\ell}_1(d)$ and $\hat{\ell}_2(d)$ latency estimators. The Estimates were made using the mean of the last three received transfers for each node

Experiment	$\hat{\ell}_1(d)$ (%)	$\hat{\ell}_2(d)$ (%)
RMBL	74.79	70.54
Royce	94.07	96.3
Lab	97.87	97.02

AML processing queues. Every choice made by a policy was classed as *correct* if it met one of the following criteria, otherwise it was classed as incorrect;

- The decision is to adapt and the actual transfer time was greater than t_{aml} seconds – correct because sending would have taken longer.
- The decision is *not* to adapt, and the actual transfer time was less than t_{aml} seconds – correct because sending would have taken less time.

The choices for all of the transfers were recorded according to these criteria, and compared to what would have been the correct choices determined from the actual transfer latencies. Table 5.2 shows the percentage of correct choices made by both dynamic estimators.

There is clearly a difference in the latency estimator performance that is dependent on the data set. For the $\hat{\ell}_1(d)$ and $\hat{\ell}_2(d)$ latency estimators, the correct choice percentage is very high for both Royce and Lab data sets – between 94% and 97%, but around 20–25% lower for the RMBL data. One explanation for this may be the difference in detection data size for the RMBL experiments: 128 kB for RMBL as opposed to 32 kB for both Royce and Lab data sets.

5.7.5 Accuracy

This section examines the accuracy of both $\hat{\ell}_1(d)$ and $\hat{\ell}_2(d)$ in predicting the latency of raw data transfers. The accuracy of latency estimates has a bearing on the generic suitability of the Adaptation policy. If the estimates provided by $\hat{\ell}_1(d)$ and $\hat{\ell}_2(d)$ are sufficiently accurate, the dynamic estimators can be used to make choices for arbitrary local processing times (rather than the specific t_{aml}). The metric M_{err} used to evaluate the accuracy of the latency estimates made by $\hat{\ell}_1(d)$ and $\hat{\ell}_2(d)$ compared to the actual transfer latencies $\ell(d)$ is given by

$$M_{i,err} = ((\hat{\ell}_i(d) - \ell_i(d))/\ell_i(d)) \cdot 100 \tag{5.9}$$

where $\hat{\ell}_i(d)$ is the estimated latency (estimated by $\hat{\ell}_1(d)$ or $\hat{\ell}_2(d)$) and $\ell(d)$ is the observed latency. This metric describes the estimate $\hat{\ell}_i(d)$ as a percentage of

Table 5.3 The accuracy of transfer latency prediction. The columns show the percentage of relative error between estimated and observed latency, and each cell shows the percentage of predictions which fell within that interval. For example, 67.71% of transfers were within 50% of the actual value for the RMBL experiment

	% of estimates within error bound		
Experiment	±50% error	±25% error	±10% error
RMBL ($\hat{\ell}_1(d)$)	66.86	46.46	27.20
RMBL ($\hat{\ell}_2(d)$)	67.71	49.01	28.33
Royce ($\hat{\ell}_1(d)$)	70.99	43.99	16.91
Royce ($\hat{\ell}_2(d)$)	75.88	47.76	18.75
Lab ($\hat{\ell}_1(d)$)	89.54	62.67	28.40
Lab ($\hat{\ell}_2(d)$)	90.14	63.52	28.66

the actual latency. If $M_{i,err}$ is positive, the error is an over-estimate, and an under-estimate if negative. This allows for comparison of error across different latencies in a way that the absolute error would not be suited to. For example, a prediction of 0.01 s seconds when the actual latency was 0.10 s is only 0.09 s error, but a prediction of 1 s when the latency was actually 10 s shows a 9 s error. Using $M_{i,err}$ would show they have both been under-estimated by 90% (-0.9).

The evaluation was performed as follows: the results of applying $\hat{\ell}_1(d)$ and $\hat{\ell}_2(d)$ to the all of the detection transfer records in the RMBL, Royce and Lab data traces were used with the actual latencies for each transfer to calculate $M_{i,err}$. Table 5.3 shows the results across each of the data traces (RMBL, Royce, Lab) with respect to three arbitrarily chosen accuracy bounds: ±50%, ±25% and ±10% of the actual latency. For each bound, the percentage of all latency estimates for a given experiment that were within these bounds are shown. The specific bounds were chosen to represent increasing degrees of accuracy: ±10% error in latency prediction could be considered *very accurate*, ±25% considered *moderately accurate* and ±50% considered *not particularly accurate*.

Bearing these classifications in mind, the results in Table 5.3 do not show particularly accurate performance. An ideal result for the latency estimators would be to have a large percentage of the data transfers be classed as *very accurate*, i.e. within ±10% all of the time. However, for each of the three data traces, the latency estimates provided by $\hat{\ell}_1(d)$ and $\hat{\ell}_2(d)$ were classed as *very accurate* only 17%-29% of the time, depending on experiment. Furthermore, the latency estimates for RMBL and Royce are classed as *not very accurate* around 66%–75% of the time.

Overall, the accuracy of latency estimates was best for the Lab data set, where around 90% of all transfers were at least within ±50% of the actual latency, although this is still classed as *not very accurate*. The $\hat{\ell}_2(d)$ latency estimator shows better performance on the whole than the $\hat{\ell}_1(d)$ latency estimator, although they are never further than 5% apart in the different error bounds categories.

5.7.6 *Effects of Varying Local Processing Time*

An important part of understanding the general suitability of the dynamic estimators is their performance with respect to different on-node processing latencies rather than just the previously observed time to process the AML. To evaluate this, the $\hat{\ell}_2(d)$ latency estimator was run over the RMBL, Royce and Lab data sets. Only one estimator was used because both latency estimators showed similar performance in previous results. For all transfers in all data sets, twenty different local processing times were evaluated, ranging from 0.5 s to 10.0 s in 0.5 s intervals.

Figure 5.16 shows a graph of the total correct decisions made by the dynamic estimator for each local processing time, and over each data trace. The x axis is local processing time and the y axis is the total percentage of correct choices the policy made. For the Royce and Lab data sets in Fig. 5.16, the $\hat{\ell}_2(d)$ latency estimator performs increasingly well in making Adaptation decisions, never falling below 80% correct in total. Both Royce and Lab data sets see an increase in correct decision percentages as the local processing time increases. However, the RMBL data set shows a different pattern: the correct choice percentage starts very high and reduces dramatically around the 2–2.5 s mark. In order to better understand the reason for the dip in correct Adaptation decision percentages, in Fig. 5.17 we plot two bar graphs for the Royce and RMBL data sets, showing the breakdown of total correct choices by choice type: either to process locally (adapt) or not (do not adapt).

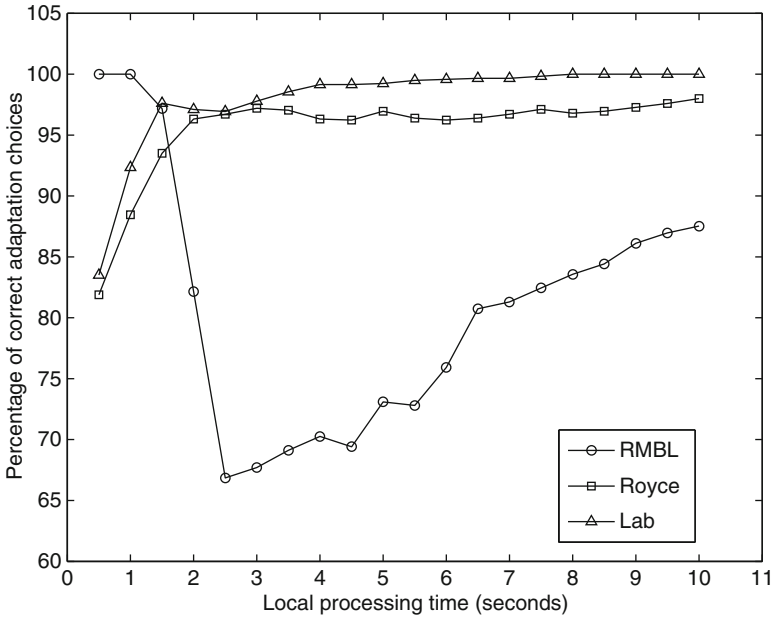
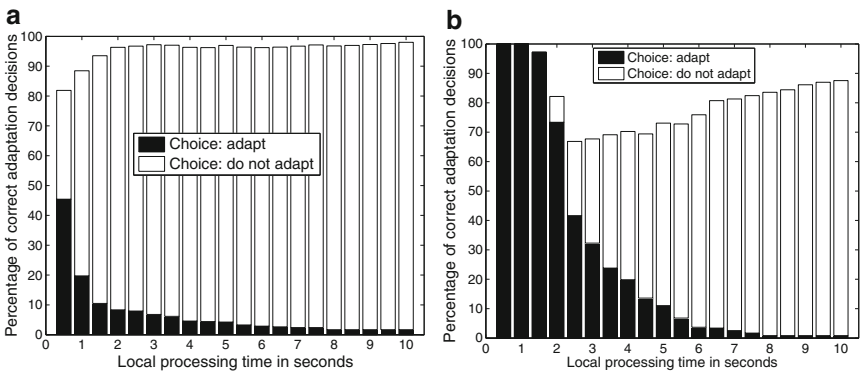


Fig. 5.16 A graph showing different processing times vs correct percentage of Adaptation choices for the three different experimental data traces used



Breakdown of correct choices for Royce data set

Breakdown of correct choices for RMBL data set

Fig. 5.17 The relative breakdown of choices for each data set. Black shows correct decisions to adapt, and white shows correct decisions not to adapt

The x axis represents the processing time, in 0.5 s intervals, and the y axis represents the percentage of *correct* choices made with the latency estimator.

For the Royce data set in Fig. 5.17(a), apart from the 0.5 s local processing time the relative breakdown of correct choices is more in favor of not processing locally and the correct choice percentage increases as local processing time increases. This is because as the processing time increases, the network's latency is much less than the local processing latency, hence more *do not adapt* choices are made.

For the RMBL data set in Fig. 5.17(b), the correct Adaptation decision percentages are entirely dominated by choices to process locally between 0.5 and 1.5 s. As the local processing time increases, the total percentage of correct Adaptation decisions decreases and more choices to not process locally are seen. The total percentage of correct decisions is worst at 2.5 s, and increases gradually from then on, where a similar trend to the Royce data set is seen.

Taking the median transfer times across all transfers in each data set reveals a potential cause for the disparity in performance. The Royce data set and the Lab data set had median transfer latencies of 0.58 s and 0.55 s respectively, whereas the Royce data set had a median transfer latency of 2.98 s. This difference in median transfer latency between RMBL and the other data sets is most likely related to the difference in the size of the raw detections that were transferred; in the Royce and Lab experiments each detection was 32 kB and in the RMBL experiment each detection was 128 kB, so hence would have taken longer to transfer. When comparing the median transfer latencies with the local processing times in Fig. 5.16, the point at which the lowest correct decision percentage occurs is around the point at which the median transfer time is located.

This means that the dynamic latency estimator gets the least correct decisions around the point at which the median transfer time is roughly the same as the local processing time. Based on the accuracy experiments, this makes sense: unless the latency estimators' predictions are very accurate, they will perform worst at this

point. It is be expected that a more accurate latency estimator would improve the percentage of correct choices when the median transfer time is roughly the same as the local processing time.

5.7.7 Summary

The aim of the Adaptation policy is to help improve the end-to-end *timeliness* of the acoustic localization system by allowing nodes to decide whether to process locally or not, based on observed network conditions. The decision to process locally or not was made using the predicted latency of the transfer, based on the recorded latency of previous transfers. Two separate latency estimators were proposed and evaluated: $\hat{\ell}_1(d)$ and $\hat{\ell}_2(d)$.

The approach used is generally applicable, as it could be used for arbitrary data payload sizes, arbitrary local processing times and arbitrary network sizes. This is because latency estimates are based on real data transfers that occur in the network, and are size-adjusted depending on the current number of hops the node is from the sink.

The latency estimators did not require direct consideration of lower-level layer specifics, such as TCP performance under high traffic, loss and congestion, and variable bit-rate transmissions. Instead, application level data was used: when a packet was sent or queued to be sent, and when it arrived, as well as the number of hops a node was from the sink. This data was easier to gather, and simple to use.

5.8 Conclusion

This chapter highlighted many of the challenges present in developing and deploying WSN systems to support high data rate applications.

A specific example of a high data rate application was used – the on-line source localization of marmots in their natural habitat. This was a suitable motivating application because it required the monitoring of a high data-rate phenomena – audible acoustics in real-time, in order to give feedback to the user of the system. The source localization algorithm was implemented on VoxNet, a platform for distributed acoustic sensing applications. VoxNet's design was influenced by experiences with bio-acoustics based applications.

The in-situ deployment of VoxNet provided two important experiences which could not have been determined from lab test or simulations. Firstly, the deployment showed up several bugs and instrumentation issues – namely to do with system data logging and multi-hop networking. Secondly, the deployment experience and post analysis of logging data showed the directions in which the system could be refined in order to improve in-field performance. These experiences could only have been gained through the physical deployment and usage of the system alongside a domain expert. Experience with the domain expert allowed informed decisions to be made

with respect to both the original design of VoxNet and its use during deployment. Experiences with the scientist, and real phenomena in the field provided motivation for improving the timeliness of the system.

Data transfer latency was identified as a key factor in the end-to-end timeliness of the marmot localization application that ran on the VoxNet platform. Latency was a side-effect of too much data being generated by nodes in the network, due to false positives and frequent triggering of the event detector.

Two approaches were proposed to help reduce this latency, Lazy Grouping and an Adaptation policy. Both were evaluated as a proof-of-concept, using data traces gathered through controlled in-situ experimentation. Information logged during deployment of the system allowed the refinements to be tested as a proof-of-concept.

Whilst the simulation results from performing off-line evaluation of Lazy Grouping and Adaptation indicated their potential to be useful in deployments, this does not constitute a full evaluation. This proof-of-concept evaluation represents only the first step in a full evaluation; a full deployment is vital and will reveal the cases where these mechanisms do not work or could be improved. However, this is beyond the scope of this chapter.

In general, acoustics represent a good example of a high data-rate phenomena which cannot easily be streamed over a multi-hop network. Therefore, it is the contention of the authors that the themes and approaches used in this chapter are suitable for many different high-date rate applications, not just audible acoustics-based localization.

Acknowledgements VoxNet is an ongoing, NSF-funded project, and the deployment work and system design in this chapter was developed from a previous paper entitled *VoxNet: an interactive, rapidly deployable acoustic monitoring platform* [3].

The author would like to acknowledge the contributions of the VoxNet development and deployment team to the work presented: Lewis Girod, Ryan Newton (both from CSAIL, MIT) and Travis Collier (UCLA). Lewis and Ryan were developers of the WaveScope engine, and main VoxNet collaborators in this work. Ryan also developed and currently maintains the WaveScript language and compiler. Travis Collier was present through all deployment and developed the VoxNet node with Lewis Girod.

In addition to the VoxNet development team, the author acknowledges the contributions of Prof. Dan Blumstein and Prof. Charles Taylor, both from the Department of Ecology and Evolutionary Biology, UCLA and Prof. Samuel Madden of CSAIL, MIT.

Support for this work was provided by NSF Cooperative Agreement CCR-0120778, NSF Grants CNS-0720079, CNS-0453809, and CNS-0520032.

References

1. Ali AM, Collier T, Girod L, Yao K, Taylor C, Blumstein DT (2007) An empirical study of acoustic source localization. In: Proceedings of the Sixth International Conference on Information Processing in Sensor Networks (IPSN '07), pp 41–50
2. Ali AM, Asgari S, Collier T, Allen M, Girod L, Hudson R, Yao K, Taylor C, Blumstein DT (2008) An empirical study of collaborative acoustic source localization. Journal of Signal Processing Systems

3. Allen M, Girod L, Newton R, Madden S, Blumstein DT, Estrin D (2008) Voxnet: An interactive, rapidly-deployable acoustic monitoring platform. In: IPSN '08: Proceedings of the seventh international conference on Information processing in sensor networks, ACM Press, New York, NY, USA
4. Anonymous (2002) Mica2 wireless measurement system datasheet. http://www.xbow.com/Products/Product_pdf_files/Datasheets/Wireless/6020-0042-03_A_MICA2.pdf
5. Anonymous (2007) Baudline. http://www.baudline.com/
6. Anonymous (2008) Cornell lab of ornithology. http://www.birds.cornell.edu/Raven
7. Anonymous (2009) Matlab. www.mathworks.com/products/matlab/
8. Bentley J (1986) Programming Pearls. ACM, New York, NY, USA
9. Driscoll DA (1998) Counts of calling males as estimates of population size in the endangered frogs geocrinia alba and g. vitellina. Journal of Herpetology 32(4):475–481
10. Elson J, Girod L, Estrin D (2002) Fine-grained network time synchronization using reference broadcasts. In: Proceedings of the 5th symposium on Operating systems design and implementation (OSDI '02), Boston, MA, pp 147–163
11. George JC, Zeh J, Suydam R, Clark C (2004) Abundance and population trend (1978–2001) of the western arctic bowhead whales surveyed near barrow, alaska. Marine Mammal Science 20:755–773
12. Girod L (2005) A self-calibrating system of distributed acoustic arrays. PhD thesis, Univerity of Caliornia at Los Angeles
13. Girod L, Elson J, Cerpa A, Stathopoulos T, Ramanathan N, Estrin D (2004) Emstar: a software environment for developing and deploying wireless sensor networks. In: Proceedings of the 2004 USENIX Technical Conference, pp 283–296
14. Girod L, Lukac M, Trifa V, Estrin D (2006) The design and implementation of a self-calibrating distributed acoustic sensing platform. In: Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (SenSys '06), pp 71–84
15. Girod L, Jamieson K, Mei Y, Newton R, Rost S, Thiagarajan A, Balakrishnan H, Madden S (2007) The case for WaveScope: A signal-oriented data stream management system (position paper). In: Proceedings of Third Biennial Conference on Innovative Data Systems Research (CIDR07)
16. Girod L, Mei Y, Newton R, Rost S, Thiagarajan A, Balakrishnan H, Madden S (2008) Xstream: A signal-oriented data stream management system. In: Proceedings of the International Conference on Data Engineering (ICDE08), pp 1180–1189
17. Hobson KA, Rempel RS, Greenwood H, Turnbull B, Wilgenburg SLV (2002) Acoustic surveys of birds using electronic recordings: New potential from an omnidirectional microphone system. Wildlife Society Bulletin 30(3):709–720
18. Hosom J, Cosi P, Cole R (1998) Evaluation and integration of neural-network training techniques for continuous digit recognition. In: In Proceedings of the International Conference on Spoken Language Processing (ICSLP '98), vol 3, pp 731–734
19. Johnson DB, Maltz DA, Broch J (2001) DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks, Addison-Wesley, chap. pp 139–172
20. Karp R, Elson J, Estrin D, Schenker S (2003) Optimal and global time synchronization in sensornets. Tech. Rep. CENS-0012, Center for Embedded Networked Sensing
21. Langendoen K, Baggio A, Visser O (2006) Murphy loves potatoes: experiences from a pilot sensor network deployment in precision agriculture. In: IEEE International Parallel and Distributed Processing Symposium (IPDPS '06)
22. Mazzoni D, Dannenberg R (2007) Audacity. http://audacity.sourceforge.net/
23. McGregor PK, Peake TM, Gilbert G (2000) Communication behavior and conservation. In: Gosling LM, Sutherland WJ (eds) Behaviour and conservation, Cambridge: Cambridge University Press, pp 261–280
24. Mellinger DK (2001) Ishmael 1.0 user's guide. Tech. rep., NOAA Tech. Report OAR-PMEL-120
25. Payne K, Thompson M, Kramer L (2003) Elephant calling patterns as indicators of group size and composition: The basis for an acoustic monitoring system. African Journal of Ecology 41(1):99–107

26. Ramanathan N, Balzano L, Burt M, Estrin D, Kohler E, Harmon T, Harvey C, Jay J, Rothenberg S, Srivastava M (2006) Rapid deployment with confidence: Calibration and fault detection in environmental sensor networks. Tech. Rep. 62, Center for Embedded Networked Sensing, the University of California, Los Angeles
27. Stathopoulos T (2006) Exploiting heterogeneity for routing in wireless sensor networks. PhD thesis, University of California at Los Angeles
28. Tolle G, Polastre J, Szewczyk R, Culler D, Turner N, Tu K, Burgess S, Dawson T, Buonadonna P, Gay D, Hong W (2005) A macroscope in the redwoods. In: Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems (SenSys '05), pp 51–63
29. Trifa V (2006) A framework for bird songs detection, recognition and localization using acoustic sensor networks. Master's thesis, École Polytechnique Fédérale de Lausanne
30. Trifa V, Kirschel ANG, Taylor CE, Vallejo EE (2008) Automated species recognition of antbirds in a mexican rainforest using hidden markov models. Journal of the Acoustical Society of America 123(4):2424–2431
31. Werner-Allen G, Lorincz K, Johnson J, Lees J, Welsh M (2006) Fidelity and yield in a volcano monitoring sensor network. In: 7th USENIX Symposium on Operating Systems Design and Implementation 2006, pp 381–396
32. Werner-Allen G, Dawson-Haggerty S, Welsh M (2008) Lance: optimizing high-resolution signal collection in wireless sensor networks. In: Proceedings of the 6th ACM conference on Embedded network sensor systems (SenSys '08), pp 169–182, DOI http://doi.acm.org/10.1145/1460412.1460430

Chapter 6
Failure Is Inevitable: The Trade-off Between Missing Data and Maintenance

Thomas Schoellhammer

Abstract Fixing sensor faults is intuitively important, but in a class of applications where the environment is easily disturbed, there is a trade-off: maintenance requires disruption of the phenomena, so that for a period after maintenance the measurements made are not meaningful. An instance of such an application is soil monitoring. Because of its spatial heterogeneity, in-situ sensing via wireless sensor networks is a natural fit to monitoring soils. However, deploying a sensor below ground requires disturbing the soil. After a settling time, which can be weeks or more, measurements can then be trusted to reflect the state of the soil prior to excavation. Maintenance requires a similar operation, and affects all nearby sensors, not simply the one being fixed. In the San Jacinto Mountains, soil monitoring sensors were deployed to measure the flux of CO_2 from the ground to the atmosphere. The deployment consisted of several sites, where each site monitored soil temperature, moisture, and CO_2 concentration at multiple depths. The sensors within a site were within a few centimeters of each other. This chapter addresses the trade-off between fixing a broken sensor, and leaving it broken, by designing and implementing a system to both fill in missing data and keep the scientist informed about the need for maintenance. The result is potentially half as much maintenance, which translates into far more reliable data being collected.

Keywords Sensor failure · Sensor deployment · Mobility · Environmental monitoring · Fault tolerance · In-field interaction · Data estimation · Data imputation · Data uncertainty · Application-cognizant

6.1 Introduction

The carbon cycle describes the movement of carbon in its various forms between the air, the ocean, and the ground [10]. One carbon compound that is also a greenhouse gas is carbon-dioxide. Biologists and ecologists recognize that

T. Schoellhammer (✉)
The Center for Embedded Networked Sensing, UCLA, Los Angeles, CA, USA
e-mail: tschoell@cs.ucla.edu

E. Gaura et al. (eds.), *Wireless Sensor Networks: Deployments and Design Frameworks*,
DOI 10.1007/978-1-4419-5834-1_6,

understanding the carbon cycle is crucial to understanding and monitoring global warming. A significant component of the carbon cycle is the CO_2 that evolves from the ground. The constant metabolism of microbes living in the soil produces CO_2 as a byproduct. For this reason, the concentration of CO_2 is generally higher in the soil than in the atmosphere, causing CO_2 to diffuse upwards through the soil. This flux of CO_2 through the soil is, therefore, of great interest to biologists and ecologists. Although it is possible to measure how much CO_2 is coming out of the soil by taking measurements at the soil surface, determining where within the soil CO_2 is being produced needs to be done in-situ.

Unfortunately, soils are notoriously difficult to monitor in-situ. This difficulty arises for many reasons, not least of which are that soils are heterogeneous. Soil composition varies spatially in unexpected ways, which then affects how phenomena are played out within the soil. It is the norm for soil structure to be unpredictable, (for example, patches of clay may exist among otherwise homogeneous sand) and therefore, for phenomena to display markedly different behavior despite seemingly similar environmental conditions or close physical proximity. Monitoring only a few sites may lead to incorrect scientific conclusions due to unfortunate deployment locations. Because of the spatial heterogeneity of soil, and the need for in-situ sensing, embedded networked sensing is a natural fit to monitoring soil ecology.

However, the application of embedded sensing technology is only the beginnings of a solution because of the impact that embedding sensors has on the soil itself. In-situ instrumentation of the soil is a destructive operation. A hole must be dug, sensors then placed, and the hole filled, all of which alters the soil's structure and composition. Because of this disturbance, after installation, scientists are uncomfortable using the data collected from sensors for several weeks, even months, as the soil resettles to something akin to its original condition. The time spent waiting for a soil deployment to be ready to collect credible data can be enormous, and this time cost means that any sensor failures can be particularly heart wrenching. When failures do occur, which is inevitable and potentially frequent [15] due to the harshness of soil environments and the delicacy of many chemical soil sensors, domain scientists are confronted with a dilemma.

Domain scientists can allow the fault to persist, but then they have to deal with the gaps in the data set from the broken sensor. On the other hand, they can fix the sensor, which may address the problem, but then the settling time must elapse again before using the data from the fixed sensor and its neighbors. In some environments, it could take months for soil to settle. This settling time leads to a long gap in the data stream from these sensors. In addition, fixing a sensor may also lead to further data loss: it has been found that nearby sensors can also break during the maintenance procedure [24], and in the case of soils, nearby sensors may be disturbed during the maintenance procedure, resulting in the loss of credibility for those sensors as well. Thus, fixing one sensor can lead to paying the settling cost, weeks of unreliable measurements, for several sensors.

The deployment dilemma arises whenever the cost to deploy a sensor and the cost to fix a sensor are comparably expensive. Three concrete deployment scenarios are remote deployments, deployments where the cost of sensing hardware is extremely

high, and monitoring animals that are sensitive to disturbance. A remote deployment scenario, where the primary cost to deploy or fix a sensor is the time and resources spent traveling to and from the deployment site, encourages technicians to consider alternatives to maintenance when sensors break. Similarly, deployment scenarios where the hardware cost dominates also fall into this class of deployment, as there may be a limited amount of replacement hardware, or the monetary cost of fixing every faulty sensor is prohibitive. Finally, the impact of disturbance is not unique to soil deployments, but is prevalent in several deployment scenarios, such as bird behavior monitoring [1].

Most deployments have reported issues with data quality [15, 22–24, 27], and dealing with the resulting data loss or corruption is inevitable during the analysis phase of almost all deployments. Although the intuitive reaction to tolerating faults is to suggest the deployment of more sensors to increase the sensing density or to add redundant sensors, physical constraints may make this approach impossible. The physical size of each sensor (a practical constraint of the technology), and the desired spacing between sensors (a constraint imposed by the scientific inquiry) may leave little space for extra sensors nearby. Furthermore, even in cases where there is enough physical space, the sensing technology itself may impose spacing requirements beyond the size of the sensor itself: soil moisture sensors often use the dielectric effect to estimate the water content of the soil, and nearby metal objects (such as other soil moisture sensors) can adversely affect the measurement process [4].

In this chapter, we first begin by describing in detail a soil monitoring deployment called the AMARSS transect [25, 26], that uses traditional wired data loggers. We then describe the hardware, software, and deployment process we used to augment this transect with wireless sensing nodes, done in order to increase the spatial density of sensing and to provide access to data in real-time. We use our experiences to demonstrate the dilemma faced by scientists when dealing with missing data and broken sensors. We conclude with a solution that we have developed to aid scientists in addressing this dilemma. The system we have built, called Vigilance, is designed to monitor deployments that periodically collected data from a set of sensors, and to expose the impact of missing data on applications that expect it.

6.2 Difficulties in Traditional Soil Monitoring

Nestled in the San Jacinto Mountains of California, at an elevation of about 1600 m, the James San Jacinto Mountain Reserve (or simply James Reserve) is a biological field station and reserve owned by the University of California. The habitat is primarily conifer and hardwood forest, and is home to several habitat monitoring systems. A handful of conventional weather stations, large wireless micro-climate mote arrays, cameras, and robotic elements collect vast amounts of environmental data [5]. One deployment in particular, called the AMARSS (Automated Minirhizotron and Arrayed Rhizosphere Sensor System) transect is used to research soil

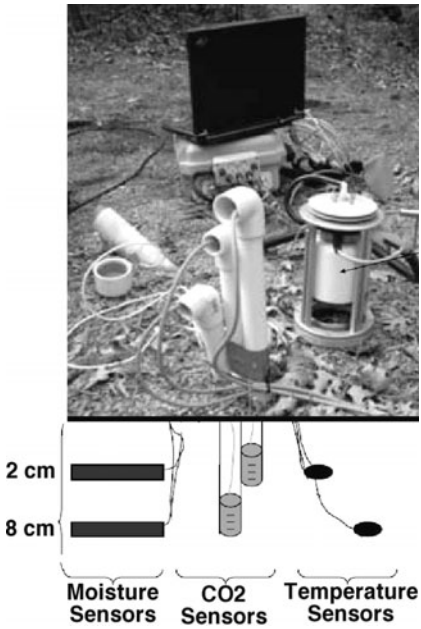


Fig. 6.1 Each site consists of a data logger and transmitter above ground, with bundles of sensors buried at depths of 2 cm and 8 cm

ecology at dense spatial and temporal scales [25, 26]. Ten sites within a 72 m transect measure micrometeorologic conditions (air temperature, relative humidity, rain, and photosynthetically active radiation) above ground, and temperature, moisture, and CO_2 concentration at 2 cm, and 8 cm below ground. Figure 6.1 illustrates the layout of the deployed sensors at each site, and Fig. 6.2 shows a schematic of the AMARSS deployment, showing the locations of the ten original sites within the transect.

Since October 2005 the deployment has collected over 26 million measurements using traditional data loggers to store data from sensors. A technician who visits each site periodically collects the data via a wired connection. Without real-time data collection, problems with the data loggers can persist for long periods of time before they are recognized by the visiting technician and fixed.

Once the data is collected, it is added to a repository to be analyzed by domain scientists, who are studying the carbon cycle. Their primary objective is to estimate the flux of CO_2 from the soil. By looking at the CO_2 concentration difference between a pair of CO_2 sensors, and correcting for soil temperature and moisture, scientists can estimate how much CO_2 is evolving from the soil, and approximately where it is coming from. The model used to calculate CO_2 flux between two depths is:

$$F = \alpha \, (T_2 + T_8)^{1.75} \left(\frac{C_8}{T_8} - \frac{C_2}{T_2} \right) \tag{6.1}$$

where F is the flux, C is the CO_2 concentration (in ppm), T is the temperature (in Kelvin), and α is a proportionality constant related to various soil properties. The subscripts refer to the depth (in centimeters) of each sensor. The change in

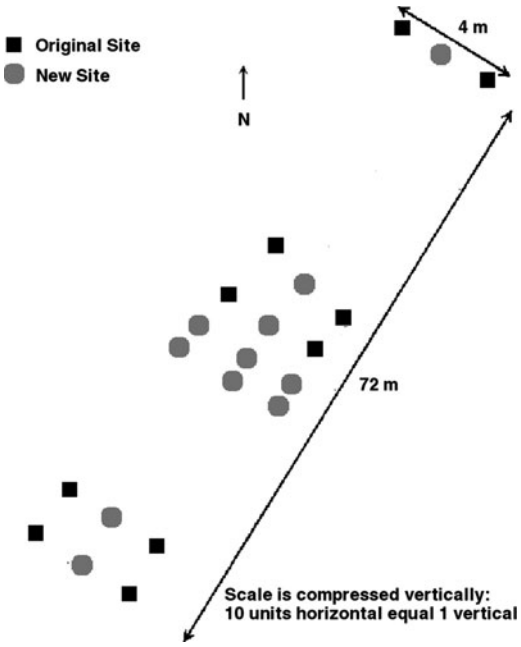


Fig. 6.2 Layout of the AMARSS transect

temperature is typically about 20 K, which is only about 5% of the absolute temperature, as compared with CO_2 concentration which can change by about 50% each day. For this reason, this model is particularly sensitive to the values of the CO_2 sensors, and largely insensitive to below ground temperature. Therefore, the CO_2 sensors are more valuable to the application of estimating CO_2 flux than the temperature sensors are. In addition to estimating the CO_2 flux, there is also interest in determining how much heat is stored in the soil. The model for heat storage is described by:

$$G = \alpha_1(T_2 - T_8) - \alpha_2\left(\frac{T_2 - T_8}{\Delta t}\right) \tag{6.2}$$

where T is the temperature (as above), α_1 and α_2 are proportionality constants, and Δt is the sampling period. The heat storage model does not use CO_2 concentration as an input, and is therefore independent of it.

It is difficult to apply the mathematical model used to estimate CO_2 flux and heat storage because of faults throughout the data set. There are two predominant types of fault; glitches and fail-stop faults. Glitches cause erroneous measurements at random times, causing one or two fallacious measurements to be recorded. These are infrequent and can often go unnoticed: before computing CO_2 flux, the data is averaged over non-overlapping time windows of one hour to make a clean data set, making their presence unnoticeable. The second type of fault is a fail-stop, which comes in two forms: either no data is reported (usually because the sensor's battery is exhausted), or bogus data is returned. Bogus data can be easy to identify, as the values often appear to be random, or stuck at a single value. Bogus data can occur

due to failed sensing hardware, or something as simple as forgetting to connect the sensor. For both types of fail-stop faults, the result is a gap in the data stream that will persist until a technician intervenes. The result can be weeks, often months of data that is erroneous, and therefore, missing, from the perspective of the analysis.

For the data gathered, scientists have dealt with sensor failure by dividing the AMARSS transect (shown in Fig. 6.2) into three regions (upper, middle, and lower) and aggregating the data within each region to produce a data stream that is not missing any data (for the most part). Within each region there are multiple sensing sites (two in the upper region, and four at each of the middle and lower regions). The raw data from sensors of the same type, deployed at the same depth, and within the same region are averaged for each hour to produce a single value. The result is a cleaned data stream that will be used for analysis that consists of one value per sensing modality, per depth, per region, per hour. In the history of the AMARSS transect, there is only one time where all the nodes within a region failed, resulting in missing data for the analysis.

This averaging procedure effectively masks the presence of failed sensors and allows analysis (i.e. the application of the CO_2 flux and heat storage models) to proceed as if there were no failed sensors during the deployment. If two identical data collections, one with no failures and the other with several, by chance resulted in the same cleaned data stream the results of the analysis would be identical. Swaths of missing data in the raw data set are in no way reflected as uncertainty in the final results. However, the interpretation of scientific results may vary drastically depending upon the uncertainty.

6.3 Wireless Sensor Network Deployment

It was the spatial variability of CO_2 flux estimates that inspired the deployment of more sensors at the AMARSS transect. These sensors were used to increase the spatial density of sensors, in order to capture variability in smaller areas, rather than to increase the amount of area covered by the deployed sensors. The deployment design itself was augmented to add a new sensing site between each of the existing five pairs of data loggers, and to densely instrument a 2 m by 1 m area that was not previously being monitored near the middle region of the transect.

In the Summer of 2006, this incremental upgrade to the spatial resolution was performed using Mica2 [3] motes to provide real-time data collection for these new sites. The wireless connectivity of the Mica2 motes provided data to scientists in real-time, avoiding the need for a technician to visit the sensors unless a fault was detected. The locations of the added sites can also be seen in Fig. 6.2.

The augmentation process benefited from the marriage of two cultures, that of the soil scientists and of computer scientists. The soil scientists are primarily concerned with where and how the sensors are placed, while the computer scientists were concerned with network connectivity, and reliable data collection. Here, we describe the deployment procedures used, and then briefly talk about the design of the software running in-network that made it possible.

6.3.1 Deployment

During the augmentation process, we were constantly assisted by one of the soil scientists who is interested in the data generated at the transect. His help was invaluable for a number of reasons. First, during the deployment process it would be easy to damage the existing sensing sites. It would seem that a spacing of 2 m between adjacent sites would provide plenty of space in which to operate. However, a healthy margin around each existing site was enforced to ensure lots of space between where we walked and worked, and where the existing sensors were buried, so as to avoid any disturbance. This only gave us about a 1 m of usable space to work within. Second, he was experienced in deploying below ground sensors, something the rest of us had never done. When digging a hole, it is important to leave one vertical wall of the hole as intact as possible. It is into this vertical wall that sensors will be pushed. It is also against that wall that the depth of each sensor will be determined.

Finding a particular depth at which a sensor should be placed is surprisingly difficult because it is not clear where the surface begins. As seen in Fig. 6.3, the surface of the soil is not smooth and continues to change. Further, sitting on the soil is a layer of decaying plant matter, referred to as mulch. The mulch slowly blends in with the soil, making it difficult to identify the top of the soil's surface. As a result, it is nearly impossible to precisely deploy a sensor at 2 cm below the surface, for example. According to our domain scientists, the surface was defined as the point where granite is first visible within the soil composition. Sensor depths were measured from this point. The sensors were placed by hand, using a translucent ruler that was placed in the hole in order to identify the soil surface and sensor depth. After placing all the sensors, the soil that was removed is placed back in the hole, and then it is critical to be careful with the wires that emerge from the soil. One accidental tug will dislodge the sensors. There is not much earth holding a sensor at a depth of 2 cm in place.

Fig. 6.3 Temperature sensors just after installation

The sensor wires then need to be routed into an enclosure, mounted about 1 m above the ground on a stake, where they will be connected to the mote. Not surprisingly, another source of difficulty was dealing with the enclosures that would house the motes during their stay at James Reserve. Each enclosure would house two Mica2 motes, along with four D-cell batteries, many wires, and ample opportunity for any number of wires to become frayed, broken, or crossed. The enclosure design itself was poor because the sensor connection wires and the Mica2 motes could not enter through the same aperture. The sensing wires themselves entered the box through a small hole in the bottom of the box, and the motes entered through a large door in the side of the box. For this reason, the motes and their sensors could not be assembled outside of the field. They had to be assembled in the enclosures themselves: the wires would have to be run through the hole in the bottom of the box, and then out the main door. Once there, they were attached to the mote, and then the whole mass of Mica2 mote, battery pack, and sensing wires were jammed into the enclosure. Two complete Mica2 motes per box was a tight fit. This cramped space caused all manner of difficulty when either changing batteries or looking for something that had gone wrong. The sensing devices themselves (buried below the ground) were difficult to deploy properly but did not fail during the duration of the deployment. However, batteries were exhausted, and various wires were frayed and broken, which lead to either no measurements reported, or erroneous measurements, both of which produce a gap in the data set.

6.3.2 In-Situ Interaction

At the time of this deployment, systems research to date had focused primarily upon mechanisms to create energy efficient wireless sensor networks. The usage model for these systems has generally been that a remote operator both tasks the network and analyzes the data. From our experience with early wireless sensor networks for environmental monitoring, we had recognized the need for these systems to also support and serve interactive users in the field. The need for interaction arises throughout the system life cycle, from designing, to debugging, to deployment testing, ongoing system health maintenance, as well as for visualization and analysis by users while in the field in order to support the collection of, and correlation with, manual observations. It is critical that interactive use not impact any ongoing data collections that the system is tasked with. A requirement that emerges from this interactive, in-the-field, usage scenario is the need for a data tasking and routing architecture that supports mobile and transient users who can submit queries.

The Hyper [21] system was thus developed, which has the following features, key to supporting mobility of the user and the addition of new sensing nodes to a network:

- Fast neighborhood assessment.
- Efficient tree convergence to offer a mobile user low latency access to a network.

- Multiple collection tree support for concurrent use by several mobile users and a fixed collection server.
- A link transmission policy that infers disconnection and reduces needless radio use.
- A storage system for delay tolerant transmission (when routes temporarily fail) and support of lonely mote clouds (when routes intentionally do not exist for long periods until mobile users connect to them).

All of these features are inspired by our own experience using and maintaining distributed sensing systems.

6.3.2.1 Mobility Usage Scenarios

From our experience deploying real systems at James Reserve and that of biologists using them, we have observed the following dominant scenarios for mobile users:

- *Users remain in each location for several minutes.* Technicians go to a location to add a new node, add new sensors, or change batteries. Users go to a location to verify data, annotate data, or manipulate the environment and observe the response. These tasks are all short-lived relative to the duty-cycled, latency insensitive data gathering style of many existing systems. The chief concern in supporting this sort of behavior is in providing a connection to the mote network as quickly as possible.
- *Even when moving, users rarely switch transmission domains.* Radio links are on the order of 50 m. After working with a node a user typically moves to a neighboring node.
- *Connectivity is not necessary during periods of mobility.* In-field users interact with nodes while seated near those nodes. The only reason for movement is to relocate near other nodes. Disconnections caused by large movements are not problematic or undesirable because sensor data services are not typically needed during transition (it is unwise to hike and look at your laptop simultaneously). Data that is stuck in-network due to the user moving somewhere else can either be dropped in-network, or be stored until a connection is re-established, depending on the user's tolerance for delay. This aspect of the usage scenario is very different from cellular connectivity or mobile IP, where the technical challenge is in maintaining a connection that provides timely delivery through a series of hand-offs and where disruptions in the connection are always undesirable.
- *A network often has a dedicated and stationary server that collects data from it.*
- *Disconnected operation of sensor nodes is a common case.* Some deployment sites are so remote that it is not cost effective to construct a multiple-hop routing tree to a permanent server-class collection sink. For these *lonely cloud* deployments, sensor nodes must store the information they collect until a user acting as a *data mule* can get close enough to them to establish network connectivity and offload their data.

6.3.2.2 Supporting In-Field Interaction

An in-field user is particularly latency sensitive; thus establishing a connection quickly is important. To connect by mote radio, Hyper provides three services designed to set up a connection with low latency: fast neighborhood evaluation, fast tree formation, and routing support for multiple sinks.

Fast neighborhood evaluation allows a newly activated node to quickly learn about what other nodes are nearby, and get an estimate of the link quality to each node. This is done by sending out a series of beacon packets that demand a response from all those that receive them. Beacon response packets indicate the number of beacons that have been received thus far. Link quality for a given neighbor is estimated from the number of beacon responses received from that neighbor, as well number of beacons received by that neighbor, as indicated in the most recent beacon response packet. Although it appears that an *acknowledgment implosion* is possible, in practice, node densities in our deployments are not high enough to warrant addressing this directly. The medium access control layer used by the radios performs well enough.

After evaluating the neighborhood, a node chooses its neighbor with the best estimated link quality as its parent in all aggregation trees that the network is currently participating in, and requests the ID of the root nodes that are currently active. In this way, a new node can quickly graft onto existing routing trees. Although the graft may not create the optimal routing tree, in practice this works well. Newly grafted nodes send a message to each sink to inform them that they have connected successfully. Nodes can participate in several data collections simultaneously, and do so by treating each routing tree as distinct. No effort is made to aggregate packets, or to multicast a single packet if two sinks request the same data. In addition to quickly learning about its one hop neighbors and grafting itself onto existing routing trees, new nodes can also request the current time as well as the current queries that are running in the network, so as to begin data collection as soon as possible. All of this has been tuned to take about 3 s.

For reliability, Hyper uses both link layer retransmissions and link layer acknowledgments, as well as persistent storage to queue packets. This not only ensures that data is saved despite transient network failures, but allows for networks of nodes that have been intentionally disconnected from a sink to collect data in remote regions were not even ad hoc infrastructure is available. Such lonely motes experience long-term disconnections, during which data is stored to persistent storage, followed by brief periods of connection when data is communicated in bulk to a data mule that visits the deployment site.

Hyper was tested extensively both in lab, and at remote sites outdoors [21]. Although lab testing was useful in the early stages of development, it was the deployment at remote sites that forced us to think about what tools and features were still lacking in Hyper's design. While deployed in-situ at the AMARSS transect, Hyper achieved a delivery rate of close to 99%.

6.4 Problems in Data Analysis

The AMARSS transect was diligently cared for to ensure that the network remained connected, batteries were replaced just before or soon after exhaustion, and broken wires were fixed quickly. However, after collecting a lot of data from the AMARSS transect and analyzing it, it was clear that a large amount of data was faulty or missing. After filtering out the faulty data, we were left with a data set with many holes from the perspective of estimating CO_2 flux and heat storage. Some amount of missing data would need to be tolerated for the sake of avoiding the resettling time for all of the sensors at a collection site in addition to the cost of maintenance, which is high: hours of travel time to visit the site, and at least a few hours of field work, and expensive sensing hardware. This implies that the missing data need be filled in somehow to allow the estimation of CO_2 flux and heat storage to proceed as usual.

Several WSN-related solutions to filling in missing data exist, notably BBQ [8], Presto [13] and work by Rossi et al [18]. Both BBQ [8] and Presto [13] propose using models that assume very little about the phenomenon being sensed to predict senor data within some tolerance. BBQ captures temporal correlations using a Kalman filter, and spatial correlations using a multivariate Gaussian distribution. After a learning phase, BBQ pulls data from the network whenever it cannot be modeled with sufficient accuracy. Presto uses a seasonal ARIMA model to model time series from each node. In addition to pulling data that cannot be predicted accurately, sensor nodes push data that does not conform to a local model. This allows Presto to save energy and capture unexpected events.

In contrast to using general modeling techniques, intimate knowledge of the environmental structure and the phenomenon can be used to accurately model sensor data. In [18], a one dimensional diffusion process is simulated, and a distributed set of sensors collects data, and fits parameters to a set of partial differential equations that describe the physics of diffusion. Once the parameters have been fit, sensor nodes can save energy by only sending model parameter updates to the base station. In these three cases, the data of interest is the raw sensor data (BBQ also supports queries for the average over a set of samples). Modeling is used to save energy for sensing devices by reducing the number packets that need to be communicated.

Because soil structure is notoriously heterogeneous and constantly changing (due to tree litter and root growth) relying on detailed structural knowledge is impractical. However, fully nonparametric techniques, like neural networks and Gaussian processes are governed by specific optimal rates of convergence on their learning capabilities. With little or no assumed structure, these techniques place a heavy burden on the data collected. As the number of parameters to estimate increases, the need for data explodes. In our soil deployment – which is small – ten sites comprise 130 sensors, not including nearby weather stations, imagery, and other auxiliary sources of information that ecologists want to correlate. An approach that disregards known structure may suffer for lack of data to train on. An ideal solution would take into account both: when structure is present, a more efficient estimator is possible, and less data is needed.

Aside from the techniques employed to estimate missing data, it should also be noted that BBQ, Presto, and Rossi et al.'s work are aimed at applications where access to the raw data is desired, but the node lifetime should be maximized.

In our motivating application, there are two challenges that are at odds with maximizing data yield: first, not all sensors are equally amenable to estimation from available data sources; second, not all sensors have an equal impact on the estimation of CO_2 flux (or heat storage). Because not all sensors (or combinations of sensors) are equally predictable or valuable, critical faults ought to be addressed quickly, while other faults can perhaps be allowed to persist until it is convenient to address them (or until they themselves become more critical). Clearly, a failed CO_2 sensor has no bearing on the estimation of heat storage. Although obvious in the case of heat storage, we have found that the importance of each of the various inputs to a scientific model may not be apparent to the technicians tasked with maintaining a deployment. Furthermore, the domain scientists interested in the data being collected may have little desire to be involved in the administrative tight loop of managing the sensor deployment, or may simply be unreachable.

Without the help of an expert, and without expending the energy of acquiring domain expertise, technicians are presented with a dilemma: when a sensors is observed to be faulty, should it be fixed? The wrong (and costly) answer to this question is "yes" or "no" depending on what is broken. A fix applied when a fault is not critical results in all the sensors at a site being disturbed, and therefore the loss of all data at that site until the soil resettles. A fault allowed to persist when the fault is critical can mean that the rest of the data collected from the site may be difficult to use, or useless, to the domain scientist. A single maintenance philosophy (e.g. fix all faults as soon as they arise) will yield poor results at some point.

Thus, in our application domain, scientists are interested in derived quantities, and are confronted with the reality of failures, some of which can be tolerated. Prior work in fault detection for sensor networks focuses on faults that effect transport in unexpected ways, so that the severity of a fault is proportional to the number of packets lost, corrupted, or delayed [14, 16, 17, 19].

The lack of consideration in the literature for the severity of faults and maintenance with respect to the missing data problem lead to the development of a new solution, called Vigilance. Vigilance is meant to run on-line and process data as it is collected, to provide a technician with expert decisions as to whether maintenance should be performed or not. Vigilance was developed, tested, and vetted using data collected from the AMARSS transect, and is currently awaiting its first opportunity to operate on-line during deployment. In the remainder of this chapter, we illustrate the design of Vigilance, and then test it by simulating failures within the collected AMARSS data, and show that it accurately predicts missing data, exposes the impact that missing data has on the certainty of derived results, and suggests proper maintenance.

6.5 Design of Vigilance

Our design goal is to assess the impact of a fault on an application in real time. Rather than wait for a user to notice missing data or to proactively replace damaged sensors as they break (and perhaps cause disturbance or accidental damage), we want to provide the administrator with just enough information to understand the impact of a fault on the application and to make timely and informed maintenance decisions.

Figure 6.4 shows the high-level structure and data flow of Vigilance. Vigilance runs at or near the sink of the network; it is supplied with time-series of raw sensor data, an *application model* that implements an equation, such as Equation 6.1 or 6.2, and a parameter that specifies how much uncertainty a user is willing to tolerate in the application model's output. The application models we consider require a complete set of inputs (referred to as a data vector) to produce output. When a fault causes sensor data to go missing, Vigilance estimates the missing data so that the application model can operate on a complete data vector. Data is labeled to differentiate real data from estimated data. Estimated sensor data, by its nature, has an associated statistical uncertainty that may impact the time series data the application produces by introducing uncertainty in the output of the application model.

Vigilance's primary outputs are warnings, when the application uncertainty goes above a user-defined threshold, and a prioritized list of maintenance suggestions. In addition, Vigilance provides a complete raw sensor data stream with missing elements filled in and time series data produced by applying an application model to the raw data from each deployment site. These latter two outputs may simplify higher-level application logic and may reduce redundant estimation processing across applications that would otherwise have to estimate missing data themselves.

The remainder of this section discusses three of Vigilance's major components, which estimate missing sensor values, propagate uncertainty in these estimates from the input of an application model to its output, and then produce maintenance suggestions.

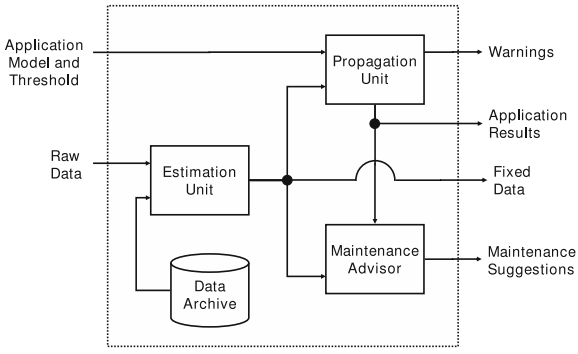


Fig. 6.4 Overview of Vigilance

6.5.1 Estimating Missing Data

Vigilance estimates missing data elements from a raw data stream as the first step in quantifying the impact of data loss. Raw sensor data arrives as tuples of the form $\langle time, sensorID, label, value \rangle$. The time and sensor ID uniquely determine when and where the sample was taken. The label describes the contents of the value field and take a value of `measured`, `missing`, or `estimated`. The label `measured` indicates that the sample was taken successfully and that the value field contains the result. The label `missing` indicates that some problem has occurred: either the sample did not arrive before some time out, or the value was deemed to be faulty. Tuples are added to the data stream labeled `missing` when expected data does not arrive within a timeout period.

Vigilance expects data to be organized on regular time boundaries, which, for example, occur every five minutes for our soil monitoring deployment. For each unique time stamp, Vigilance's estimation unit looks for missing data across all sites. Those vectors labeled `measured` are emitted unchanged. For each vector labeled `missing` the value of the missing data is replaced with a probability density function (pdf) that encodes the "best" single estimate of the missing data (the mean of the pdf) as well as the uncertainty (how the distribution spreads out from the mean). The more narrow the pdf, the more certain we are that the true value of the missing data is close to the mean of the pdf, indicating that the mean is an accurate estimator. A new vector, having the same time stamp and sensor ID is emitted, but the value field contains the pdf, and the label reads `estimated`. Note, that when estimating missing data, it is important that the data that the model is trained on resemble the inputs from which missing data is being predicted, as a mismatch may result in estimates that claim to have low variance, but are inaccurate. We leave for future work the step of deciding when the training data is suitable to predict missing data.

To create a pdf for the missing data, the estimation unit first trains a model to predict the values of missing data. While there are several procedures for learning a regression function (local polynomials, neural networks, support vector machines), and we have experimented with many, we use Polymars [12] because it provides a balance between model complexity and interpretability that is particularly well suited to our goals. Polymars is a flexible tool that can capture various smooth patterns easily (e.g., temperature time series), but may not be appropriate for modeling abrupt changes, as may be seen in target tracking scenarios, or for modeling alarms. A model trained by Polymars predicts the values of the broken sensor (which we refer to as the response, y) from data sources derived from the available sensor data (which we refer to as the predictors, x). The response and the set of predictors can be represented as a table, where the values in the first column correspond to the response, and each subsequent column corresponds to a particular predictor. In this table, each row is referred to as an *observation*. Polymars produces a function that, given the predictors for any particular observation, produces a *fitted value* that approximates the response for that observation. The difference between an observation's response and its fitted value is referred to as a *residual*.

In order for Polymars to train a model that will predict accurately, some of the predictors should correlate well with the response. We first explain the set of predictors we use and then motivate our use of Polymars over other adaptive fitting techniques.

Predictor Construction

Predictor construction is the process by which each available raw data stream (e.g. CO_2 at 2 cm) is transformed into a new stream that may correlate better with the values of a broken sensor. Predictors are used as input to Polymars, which then estimates lost data. Analogous sensors (those sensors of the same type, at the same depth, but at different sites) in our deployment often display highly correlated behavior. For example, in the soil deployment, all sites are affected by the sun in a similar way, and so analogous sensors typically behave similarly. While there may be site specific parameters (e.g. shading, organic litter) that lead to minor differences in behavior, the differences are small compared to the amplitude of the signal. Thus we use analogous sensors to construct predictors that leverage spatial correlations between sites.

Companion sensors (those sensors of the same type, at the same site, but at different depths) can also display highly correlated albeit time-lagged behavior. For example, within a single site, solar radiation received at the soil surface propagates downward through the soil. Shallow sensors experience a change in temperature, and at a later time deeper sensors experience a dampened version of the same change. Similarly, increases in CO_2 concentration at lower depths eventually result in increased CO_2 concentration closer to the surface as CO_2 rises. Temporal correlations can be seen within a site by looking at the cross-correlation function of companion sensors. If the lag between two companion sensors is known, then a sensor can be used as a powerful predictor for its companion. Unfortunately, the lag seen between companion sensors is neither constant within a site over the course of a year, nor is it the same for all sites at a particular time of year.

Through analyzing more than a year of sensor data traces, we have found that the lag between companion sensors falls within a small range. Lag between two sensors observing the same phenomenon occurs in several different sensing modalities, typically when there is some sort of diffusion process at work, and there are standard statistical tools for determining it; for example, in the soil deployment, regardless of site, the absolute lag between companion temperature sensors or companion CO_2 sensors was always less than three hours. Rather than try to determine the lag for each pair of companion sensors, for each site, for each time of year, we take a simpler approach that works well regardless of season, site, and sensing modality. Each operational sensor that is local to a broken sensor is used to generate several shifted versions of itself, with lags ranging from -3 hours to $+3$ hours, in 10 min increments. The set of shifted time series makes up our set of predictors that capture temporal correlations. A handful of these predictors will correlate well with historical data from the broken sensor, irrespective of what type of sensor is broken.

Model Generation

A model is trained from the set of predictors so that missing data can be estimated. Here, we provide a brief overview of the operation of Polymars; the reader is referred to [12] for details. Polymars is not necessarily the right modeling procedure for every observational deployment. However, its ability to model smooth data well makes it applicable to a wide range of microclimate monitoring deployments. Moreover, in Vigilance, it is straightforward to replace Polymars with another modeling technique, which may be desirable when, for example, the data collected is either not smooth, or when the user has knowledge of a more application-specific modeling technique. Polymars is an adaptive function estimation routine, meaning that it will identify both the important components in a model (i.e. which predictors to use) as well as their functional form (i.e. how they are used). In essence, given a set of predictors $\{x_1, x_2, x_3\}$, Polymars expresses a function f in terms of a constant, the *main effects* (each predictor by itself), and the *two factor interactions* (the product of two predictors), to approximate the response, y:

$$f(x_1, x_2, x_3) = \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 + \alpha_{12} x_1 x_2 + \alpha_{13} x_1 x_3 + \alpha_{23} x_2 x_3$$

Polymars finds the values of the α's that produce a good fit, preferring models with fewer terms (i.e. with more α values set to zero) when additional terms offer only a marginal improvement to the approximation of y by f. Several standard tools exist to analyze a model with this functional form in order to help an investigator understand why the model fits the data well. Higher order interactions, as a rule, are more difficult to interpret, and for this reason Polymars forgoes them (e.g., note the absence of $\alpha_{123} x_1 x_2 x_3$).

After training a model f, the set of residuals r, computed from all observations, is saved and associated with f. If f fits the data well then the distribution of r will be narrow and centered at zero. We use the distribution of fitting residuals as an indicator of how well f will be able to predict the value of missing data: the more narrow the distribution of residuals, the more accurate the predictions. When a prediction e is made using f, e is added to the distribution of fitting residuals r, and the result is taken as an empirical distribution for the missing data.

6.5.2 Application Uncertainty

Vigilance's propagation unit determines whether the pdfs for missing data made by the estimation unit induce too much uncertainty in the application. When data is missing, the uncertainty in the output of the application model is also represented as a pdf. The wider a sensor data pdf is, the wider the application output pdf may be. The width of the application output pdf is compared to the application threshold, which is user specified. A narrow width (below a threshold) indicates a tolerable

application uncertainty and thus that the estimates for missing sensor data are sufficient for the application's purposes. Conversely, a wide pdf suggests that the sensor data estimates are too uncertain, and thus that maintenance is required.[1] A warning that indicates the site and its broken sensors is displayed to the administrator.

The propagation unit uses the application model, the application threshold, and the fixed sensor data as input. Vigilance keeps a function pointer that references the application model, which takes a vector of values as input and produces a single value as a result. Vigilance assumes that there is a set of data per time stamp that is to be processed by the application, and part of Vigilance's configuration specifies the application model as a function, as well as what set of data should be passed to it. For our deployment, the data values for a single site and time are passed to the application model, although nothing prevents passing data from multiple sites together.

The set of vectors for a particular site i and time stamp t are gathered together to make X_{it}, referred to as the *sensor input*. If the sensor input vector contains no filled in data, then it is passed to the application model, F_A, and the result Y_{it} is emitted.

$$Y_{it} = F_A(X_{it})$$

When one or more elements of the sensor input vector are pdfs then Vigilance uses the application model to produce a pdf describing Y_{it}.

To estimate how input uncertainty propagates through the application model, a missing data method called Multiple Imputation (MI) [20] is applied. Because each missing input is approximated by a pdf, samples can be drawn from each pdf, and placed in their corresponding positions within the sensor input vector to produce a complete input vector, suitable for the application model to operate on. Several complete versions of the sensor input vector are created this way, and each is processed by the application model to create a set of application results. The operation of MI is illustrated with a small example.

Suppose a transform T takes an input vector of the form $\langle a, b\rangle$ and produces a single output c.

$$T(\langle a, b\rangle) = c \tag{6.3}$$

If b is missing, T cannot operate. However, using a pdf b_{pdf} that describes the distribution of b given the value of a, this limitation of T can be circumvented. Several samples are drawn from b_{pdf} to create several candidate values for b, referred to as *imputations*, and denoted by

$$b^{(1)}, \ldots, b^{(n)}$$

Each imputation is used to create several candidate input vectors

$$\langle a, b^{(1)}\rangle, \ldots, \langle a, b^{(n)}\rangle$$

[1] It can also point to the need to reexamine the model being used.

Each instance is processed by T as normal, creating a set of possible results.

$$T(\langle a, b^{(1)} \rangle) = c_1$$
$$\vdots$$
$$T(\langle a, b^{(n)} \rangle) = c_n$$

The mean of the results $n^{-1} \sum_i c_i$ is used as the best estimate of c, and the distance between the 2.5% and 97.5% quantiles of the results (referred to as the 95% confidence interval) is used as an estimate of the uncertainty in c due to b's absence. Although we focus on this quality metric (the 95% confidence interval) others are possible (such as the standard deviation) and Vigilance could use them as well.

MI is the only missing data method we know of that allows complete data analysis techniques, such as the soil CO_2 flux and heat storage models, to be used without modification. In addition to its simplicity, MI has several advantages over other missing data methods. First, the collector of the (incomplete) data set may have a good idea as to why particular elements went missing. This allows the collector to publish a data set where those missing elements are explicit, and simultaneously provide candidate values (or a generative model) for each missing element that captures the collector's insight. Second, the collector may not know who plans to use the data or for what purpose. The collector can provide multiple candidate values for each element of missing data, which allows for a clean separation between the data producer and the data consumer. Third, candidate values can be generated from multiple prediction models, allowing model uncertainty to be reflected in the analysis as well. Fourth, MI can be expressed in a Map-Reduce framework [7], allowing it to be efficiently parallelized. As observational sensing systems grow in size, the amount of missing data due to faults will grow as well. MI offers a scalable means of estimating the impact of missing data on application inference.

After sampling the pdfs to generate several candidate instances of the the sensor input vector, each is transformed by the application model, and the set of application results is collected into a pdf. Vigilance uses the mean of the application pdf as the single best estimate of Y_{it}, and takes the width of the 95% confidence interval as an estimate of spread in the application output. The more narrow the 95% confidence interval, the more confident we are that the true value of Y_{it} is close to the mean. The size of the 95% confidence interval is compared to the application threshold, and intervals that exceed the threshold are flagged. The output of the propagation unit is a vector containing the time, the site ID, the mean value, and the 2.5% and 97.5% quantiles $\langle time, site, value, lowQ, highQ \rangle$.

6.5.3 Maintenance Suggestions

When the application uncertainty exceeds user-defined limits, the maintenance advisor suggests fixes that could be performed and provides estimates of how much

the uncertainty will decrease (as measured by the narrowing of the 95% confidence interval) by doing so. The maintenance advisor takes the output tuples of the propagation unit, as well as the fixed data stream from the estimation unit, and stores a time series of application results and fault combinations for each site. The maintenance advisor considers one site at a time. For each site, the maintenance advisor determines the unique fault combinations that have occurred at that site over time. For each unique fault combination, the maintenance advisor takes the average of the associated 95% confidence intervals. Thus, the maintenance advisor can see how the application 95% confidence interval has widened on average as faults have accumulated at a site. Therefore, the maintenance advisor can estimate how the 95% confidence interval will shrink at a site if faults are fixed in the opposite order in which they occurred. The maintenance advisor produces messages that indicate the site ID, the fault combination, and the associated 95% confidence interval. In general, this approach is desirable, however, the specific constraints of the phenomenon being monitored may preclude its use. For example, in our soil deployment, fixing any sensor at a site disturbs the whole site. For this reason, when the maintenance advisor suggests necessary maintenance, all sensors at a site should be fixed, not just those that have the most impact.

6.6 Implementation of Vigilance

Vigilance has been implemented and extensively tested using real soil monitoring data, and two real soil monitoring applications. However, to date, Vigilance has not yet been integrated with the soil monitoring deployment's collection server, thus our presentation of its real time behavior is derived from offline analysis. Various fault combinations are simulated by labeling some data as `missing`. Because of Vigilance's heavy reliance on statistical analysis, it is implemented entirely in R [2], a free statistical computing package. Vigilance can therefore be easily integrated into existing workflow platforms that support R processing boxes, such as Kepler [6]. This section describes the implementation details of our data estimation and model output uncertainty process.

6.6.1 Missing Data Estimation

For each faulty data point, Vigilance takes an inventory of the available sensors that can be used to generate the set of predictors described in Sect. 6.5. To generate the shifted version of the available local sensors that Vigilance uses, it must wait three hours from the time a piece of data goes missing to the time when it can address the fault. For this reason, Vigilance currently operates three hours behind the current time. This amount of time could be reduced, as in practice we only see shifts of about one hour being used by Polymars. After collecting the available predictors,

Polymars produces a model capable of predicting the missing data, as well as the set of fitting residuals. In our implementation, all neighboring nodes that are available at the time of a fault are used as predictors, and local sensors are used if they are available for a six hour time window that surrounds the fault time.

6.6.2 Uncertainty Propagation

When elements are missing from a sensor input vector, X_{it}, Vigilance independently draws several random elements from each of the constituent pdfs to produce several instance of X_{it}. The number of sensor data input vectors Vigilance generates grows exponentially with the number of missing elements in the sensor input vector. Specifically, when k elements are missing, $I = max(20, 10^k)$ estimated vectors are constructed. Empirically, we found that 20 offers good results when the vector is missing only one element. After creating and processing the I instances, the mean and 2.5% and 97.5% quantiles are calculated from the set of application results. When none of the values are missing from the sensor input vector, Vigilance passes the complete vector directly to the application model.

6.6.3 Maintenance Suggestions

The maintenance advisor keeps a time series of the application output's 95% confidence interval for each site. Per site, the historical data is scanned for times where one or more faults were present. Because faults are often persistent, the same fault scenario may be seen for several different time stamps. The average 95% confidence interval size is calculated over the set of time stamps that all exhibit a particular fault. This creates a mapping between each fault scenario that the site has experienced, and the average application uncertainty resulting from that fault scenario. Whenever a new tuple arrives that exhibits a failure at a site, the maintenance advisor scans its history of data to update the mappings between fault scenario and average 95% confidence interval size for that site, and produces a warning that indicates the site, the 95% confidence interval size, and the sensors that have failed.

6.7 Evaluation

We show that Vigilance can reduce required maintenance by accurately filling in missing data, quantifying the resulting uncertainty in the application, and then by calling attention to only those sites that require maintenance because they induce uncertainty that exceeds user-defined constraints. We expect Vigilance's modeling technique to predict missing values accurately and to differentiate faults by their

respective severity. We begin by evaluating the prediction accuracy and coverage of our modeling technique. We show that Vigilance predicts data up to twenty times more accurately using its specially crafted set of predictors compared to simply using the set of analogous sensors for prediction, which is commonly done in the literature.

We quantify how Vigilance's suggestions affect maintenance efforts and network uptime. Using a simulated data set and a typical sensor failure probability, we show that Vigilance reduces the required maintenance by a factor of three, and increases the amount of usable data collected by a factor of three.

Next, we discuss the implications that Vigilance has for deployment design. We discuss how, using Vigilance as a deployment design tool, we can now formulate a deployment upgrade or new deployment plan as a decision problem, amenable to optimization. Lastly, we illustrate Vigilance memory, and processing requirements and its scalability.

6.7.1 Methodology

Simulations are performed on a data set that spans almost 19 days. Faults are simulated by deliberately withholding data, which is then used as ground truth for the simulation results. The time at which a fault starts is chosen at random, and the fault duration, referred to as the gap size, is varied from five minutes (one sample) to one day. All combinations of sensor failures have been simulated, and selected results are presented. Only temperature and CO_2 sensors faults were simulated because both application models that we examine are insensitive to the range of soil moisture values seen within our simulation data set. In addition, only results from where either one or two sensors are broken (referred to as single fault scenarios, and double fault scenarios, respectively) are presented because larger failure scenarios exhibit intolerable application performance and always require maintenance. Each failure scenario is simulated twenty times, and the 2.5%, 25%, 50%, 75%, and 97.5% quantiles are calculated. Simulations were run on a quad-core Intel Xeon, running at 2 GHz, with 5 GB of memory. Vigilance does not take advantage of multiple cores, and simulations were run four at a time.

6.7.2 Prediction Accuracy

We look at how well a model made by Polymars using our full set of predictors performs on novel data compared to other methods. We compare a model trained from our choice of predictors (referred to as *pm*) with predictions made by models trained on different subsets of predictors. The first subset only uses the set of temporal predictors derived from sensors local to a failure (referred to as *temporal*). This includes predictors derived from all other local sensors that are operational, not

just the companion sensor to the failed sensor. The second model only uses the set of neighbor sensors that are analogous to the failure (referred to as *spatial*). As a baseline, we also compare the prediction accuracy of the above models predictions made by a linear least-squares line. The ten closest points to a missing point are used to fit the line because, experimentally, we determined that they offer good resilience to noise while avoiding the incorporation of non-local data for small gaps. We find that *pm* produces predictions that are on the same order as the manufacturer specified accuracy of the sensors, outperforms the other prediction methods (with minor exceptions), and has prediction accuracy that is nearly constant over a range of gap sizes.

For each fault scenario and gap size, the prediction error is computed for twenty trials, and the quartiles of the prediction error over all trials is computed. Because the distribution of prediction error is roughly symmetric and centered around zero, the interquartile range is computed, and is used to compare prediction accuracy. Throughout this evaluation, we refer to the prediction error interquartile range as simply the *prediction range*.

Figure 6.5 shows the size of the interquartile range for the four single fault scenarios, for each prediction model. Over the four single fault scenarios, the

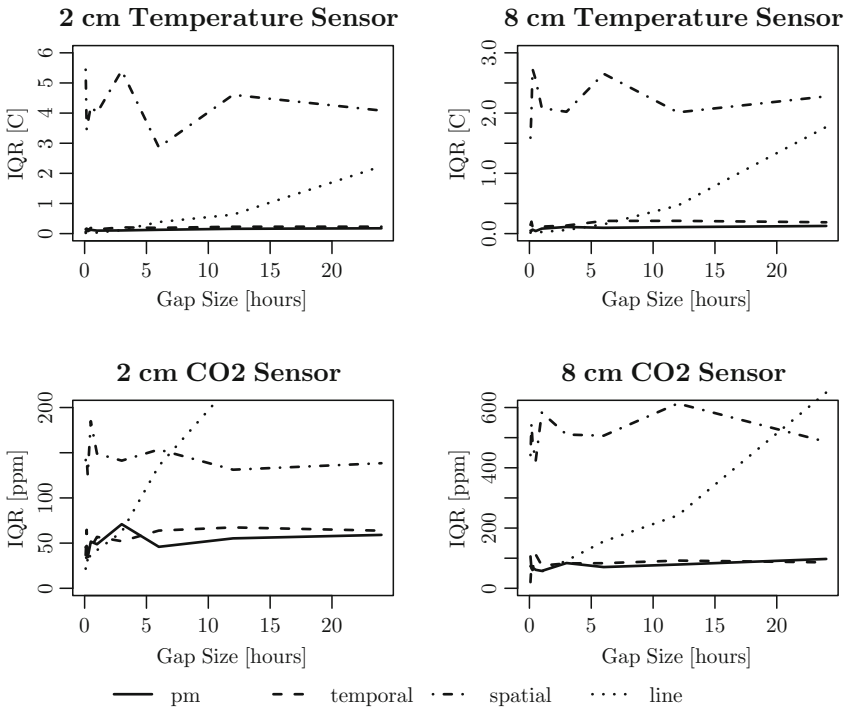


Fig. 6.5 The prediction error interquartile range for the temperature and CO_2 sensors. The best-fit line performs marginally better than other models for small gaps. The performance of other models is steady across the range of gap sizes

least-squares linear fit has a prediction range comparable or smaller than any of the other prediction models when the gap size is less than 30 minutes, and in some cases (the 8 cm temperature and the 2 cm CO_2), it is smaller for up to a gap size of 3 hours. What this suggests is that for short gaps in measurements of a smoothly varying phenomenon, a complex prediction model need not be trained, and only a small amount of temporally local data from the faulty sensor is needed to accurately predict missing data. However, as gap sizes become larger than about 4 hours, the least-squares line performs poorly.

Interestingly, *pm*, *spatial* and *temporal* exhibit nearly constant prediction range over the set of gap sizes, with *pm* having the smallest range among the three (with some minor exceptions), which is expected as *pm* is trained on a superset of predictors. For the 2 cm and 8 cm temperature sensors, the prediction range for *pm* is approximately 0.2°C, and *temporal*'s range is comparable. The data sheet for the temperature sensors quotes an accuracy of ± 0.7°C, implying that further refinement in prediction accuracy may be unnecessary. For the 2 cm CO_2 sensor, the prediction range for *pm* and *temporal* is approximately 50 ppm, and for the 8 cm CO_2 sensor the range is approximately 80 ppm for both. The higher variability in CO_2 concentration at 8 cm makes it more difficult to predict, however the data sheet for the CO_2 sensors quotes an accuracy of ± 150 ppm, again implying that further refinement in prediction accuracy may be unnecessary. Throughout all the single fault scenarios, *spatial* exhibits poor performance, with a range of approximately 4°C and 150 ppm at 2 cm, and approximately 2°C and 500 ppm at 8 cm, for temperature and CO_2, respectively. This brings to light the huge boost in prediction accuracy that a modeling technique that leverages the structure of the problem has over more general techniques: the set of temporal predictors, which simulate a diffusion process, enable a prediction accuracy that is about 20 times better than simply using spatial data.

When predicting a single broken temperature sensor, the inaccuracy due to *spatial* does not affect the estimation of CO_2 flux significantly, and in fact, our collaborators currently do something similar to estimate temperature data when a temperature sensor fails, so as to avoid maintenance. However, a single broken CO_2 sensor requires maintenance if *spatial* is being used to estimate its missing data. As will be shown in Sect. 6.7.4, using *pm*, even a single broken CO_2 sensor can be tolerated resulting in far less maintenance, and more usable sensor data.

Figure 6.6 shows the prediction range of the four most difficult double fault scenarios, where both companion sensors are broken (e.g. both temperature sensors, or both CO_2 sensors). The other eight double fault scenarios exhibit performance similar to the four single fault scenarios listed above (e.g. when predicting the 2 cm temperature sensor, and the second fault is one of the CO_2 sensors then the performance is similar to the single fault scenario where only the 2 cm temperature sensor is broken) and are omitted for brevity. The accuracy of *line* and and *spatial* are unaffected in the double fault scenarios since they do not depend upon other sensors that are local to the failure.

For both temperature sensors, the prediction range for *pm* is approximately 0.4°C, while for *temporal* it has jumped to between 1°C and 2°C in the absence of local information. This shows the utility of companion sensors when predicting

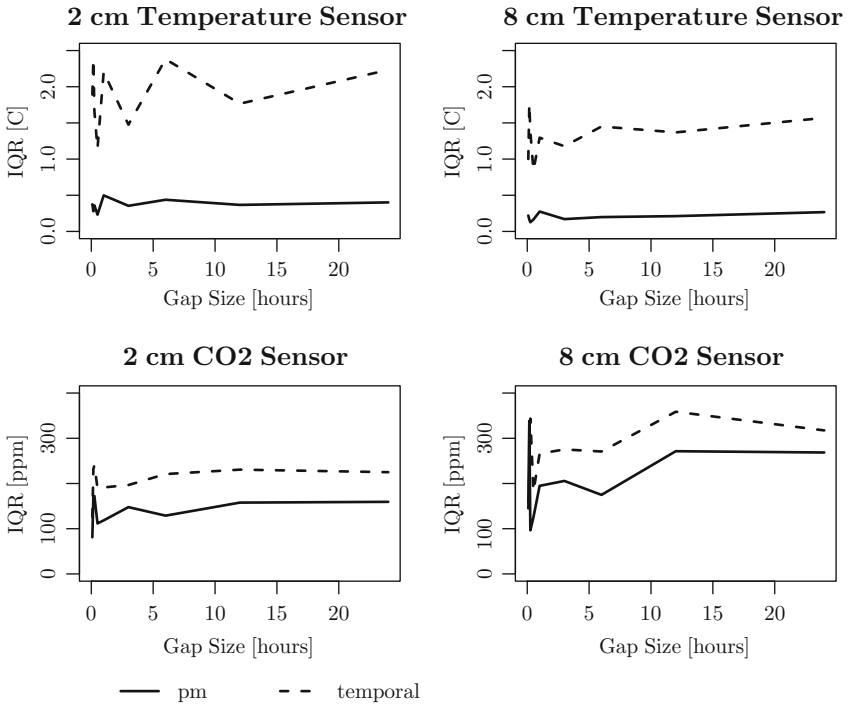


Fig. 6.6 The prediction error interquartile range for the double fault scenarios. Only double fault scenarios where both companion sensors are broken are shown

temperature. Whereas *pm* can fall back on neighboring analogous sensors, *temporal* has to rely loosely upon the available CO_2 and moisture sensors. The model for *temporal* produced by Polymars for either temperature sensor when only a single failure is present shows a heavy reliance upon the companion temperature sensor. Therefore, when the companion temperature sensor has also failed, and the neighboring analogous sensors are not available, the accuracy degrades significantly.

For the 2 cm CO_2 sensor, the prediction range for *pm* is about 150 ppm, and increases to about 250 ppm for the 8 cm CO_2 sensor (recall that there is more variation in CO_2 concentration at 8 cm compared with 2 cm). For *temporal*, the range is about 200 ppm for 2 cm and jumps to about 300 ppm for the 8 cm sensor.

6.7.3 Coverage

We look at how uncertainty in sensor data estimates made by *pm* propagates through the CO_2 flux application model. Vigilance calculates the 95% confidence interval for CO_2 flux for each fault. The frequency with which the 95% confidence interval

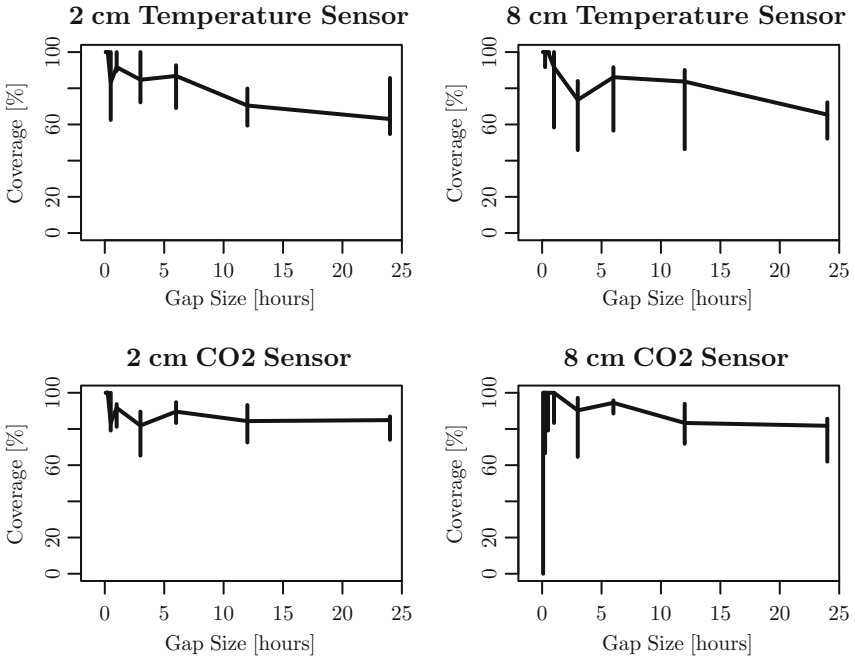


Fig. 6.7 The median coverage and interquartile range for the temperature and CO_2 sensors

actually contains the true flux value is called the coverage. We show that coverage for the four single fault scenarios is around 80% for gap sizes up to about half a day, and degrades for larger gap sizes. For each fault scenario and gap size, the coverage was computed for twenty trials, and the median coverage per gap size and interquartile range was determined.

Figure 6.7 shows coverage in the four single fault scenarios. For small gaps, the coverage is near 100%, but for a day-long gap it can be as low as 60% for temperature and 80% for CO_2 in the single fault scenarios. Although not shown, the coverage for the multi-fault scenarios hovers near 100% regardless of gap size, because the distribution is unusably wide. The coverage indicates that Vigilance's operating parameters need further tuning. Because Polymars simultaneously determines the model form and fitting parameters, the residuals are not independent, and as a result, the distribution of residuals is optimistically tight. Estimating confidence intervals for adaptive estimation routines is a subject of current research in statistics [11]. There are procedures proposed to widen the confidence intervals to correct for this [9], and this is left for future work. This result does not affect the major contribution of this work, which is to identify the relative severity of faults, and thus does not affect implications for deployment design, upgrade, and maintenance, discussed in Sects. 6.7.4 and 6.7.5.

6.7.4 Maintenance

Here we show that using Vigilance to monitor a simulated deployment similar to our own, maintenance is reduced by a factor of three for reasonable sensor failure rates, and the amount of data collected is three times more compared with doing maintenance whenever a fault occurs. First, we present the 95% confidence interval calculated by Vigilance for both the CO_2 flux and heat storage applications under various fault scenarios, and identify the set of scenarios that require maintenance to sustain the performance of both. We then use this information to simulate a deployment similar to our own, where both CO_2 flux and heat storage are being estimated for a year.

Table 6.1 shows the 95% confidence interval size for both applications under the fault scenarios considered. In the first column, a fault in one of the temperature sensors is indicated by a T, a fault in one of the CO_2 sensors is indicated by a C, and the subscripts indicate the depth of the sensors, in centimeters. For the CO_2 flux application, a width of more than $10\,\text{ppm}\,\text{m}^{-2}\,\text{s}^{-1}$ cannot be tolerated, and for heat storage, a width of more than $50\,\text{W}\,\text{m}^{-2}$ cannot be tolerated. There are several things to note about this table. First, there are clearly differences in maintenance requirements. Flux estimation tolerates the failure of both temperature sensors, while heat storage estimation cannot. Heat storage estimation can tolerate the failure of both CO_2 sensors, but flux estimation cannot. Second, looking at a single application, the relative size of the 95% confidence interval indicates the relative importance of various fault scenarios to that application. Knowing the relative importance, it is possible to prioritize maintenance for an application. Although in our deployment, the need to disturb a site implies that all faults within the site should be addressed, one can imagine a deployment where if the settling time is small enough an administrator could begin to ask which k out of the N faults over a deployment should be addressed to maximize the application data yield, or minimize the maintenance while maintaining some yield. Third, it is easy to see how the application

Table 6.1 The width of the application 95% confidence interval. Looking at the fault scenarios for a single application indicates the relative importance of the fault scenarios to the application. Looking across applications, maintenance decisions vary

Fault Combination	CO_2 Flux $\text{ppm}\,\text{m}^{-2}\,\text{s}^{-1}$	Heat Storage $\text{W}\,\text{m}^{-2}$
T_2	0.05	22.59
T_8	0.03	16.95
T_2 T_8	0.31	129.9 (fix)
C_2	4.37	0
C_2 T_2	5.06	22.59
C_2 T_8	5.04	16.95
C_8	6.60	0
C_8 T_2	7.60	22.59
C_8 T_8	7.51	16.95
C_2 C_8	21.6 (fix)	0

will suffer as faults accumulate. For example, assuming the CO_2 flux application threshold was actually seven, if sensor C_2 is already broken, then another failure in either temperature sensor would be critical, and an administrator may be inclined to take preventative measures. Fourth, when looking at multiple applications, the set of intolerable fault scenarios can be easily identified. Fifth, maintenance can be prioritized across multiple applications. For example, given M applications, each with their own preferences about which k out of N broken sensors to fix, which should be fixed Or, given C amount of money and the costs for each sensor type which sensors should be fixed?

From Table 6.1, to satisfy both applications simultaneously, Vigilance can tolerate at most the failure of one CO_2 sensor and one temperature sensor. Currently, in our soil deployment, CO_2 sensors are "important" so that when one breaks maintenance is performed. Temperature sensors are "important" to the heat storage application. Therefore, if both heat storage and CO_2 flux are being monitored then any temperature sensor failure or CO_2 sensor failure will engender maintenance if Vigilance is not being used, reducing the total amount of usable data collected for both applications.

To quantify this, we simulate a deployment similar to our own, where a each of ten sites has two CO_2 sensors and two temperature sensors, and we compare the data yield when Vigilance is not used to the data yield when Vigilance is used. We simulate one year data collection, where each sensor has a probability P of failing each day, and maintenance incurs a settling time of twenty days before usable data is again produced. When not using Vigilance, any failure causes maintenance to occur. When using Vigilance, as long as at least one temperature sensor and one CO_2 sensor are operational at a site then maintenance is not performed and data usable for both CO_2 flux and heat storage estimation is collected. The proportion of usable data (at the granularity of a day) is computed. The failure probability is fixed for each trial, and twenty trials were run per failure probability.

Figure 6.8 shows the percent of total amount usable data that could have been collected over the course of a year for each of the two deployment scenarios. The percent of useful data for each failure probability is represented by its interquartile range (the thick segments), and the symmetric 95% quantile range[2] (the thin segments). Note that the results are staggered so that the segments do not overlap, and does not indicate a difference in failure probability.

As the probability of sensor failure increases, the percent of usable data drops rapidly when not using Vigilance. With a failure rate of 1.5%, the median amount of usable data is 30%. The amount of usable data decays far more gracefully when using Vigilance, where the amount of usable data is 76% for a failure rate of 1.5%, increasing the amount of usable more than a factor of two. Although a failure probability of 1.5% seems high, a failure could be any event that requires a sensors to be removed from the ground. For example, a sensor may need recalibration periodically, and recalibration also requires disturbing the soil. An example of a soil

[2] This is the distance between the 2.5% and 97.5% quantiles

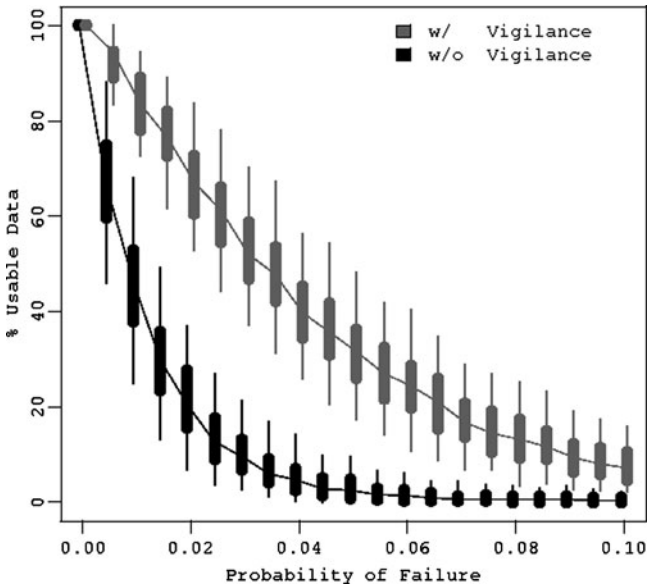


Fig. 6.8 A simulated deployment where the amount of usable data retrieved using Vigilance is compared to a deployment without Vigilance

sensor that needs frequent recalibration is an Ion Selective Electrodes (ISEs), used in ground water contaminant monitoring, and may require recalibration every two to eight weeks, leading to a "failure" rate of $\sim$1.5%.

Figure 6.9 shows the number of sites that need maintenance per day on average, with and without Vigilance. With a failure rate of just of 2%, a deployment without Vigilance incurs one maintenance operation per day, Vigilance, by comparison, allows sensors to fail with a probability of almost 8% to incur one maintenance operation per day, Over the range of failure probabilities shown, Vigilance incurs about three to four times fewer maintenance operations per day.

6.7.5 Deployment Design Implications

Using the data collected, we make specific design suggestions for our soil deployment, and offer it as anecdotal evidence to Vigilance usefulness in deployment design in general. In addition, we discuss how using Vigilance, a deployment upgrade or a new deployment plan can be formulated as a decision problem, amenable to optimization.

An important take-away from the prediction accuracy results from Sect. 6.7.2 are the implications of how to incorporate sensing resilience into deployment design, both for improving the design of the existing deployment, as well as the making more robust designs for future deployments. First, given a limited budget of sensors,

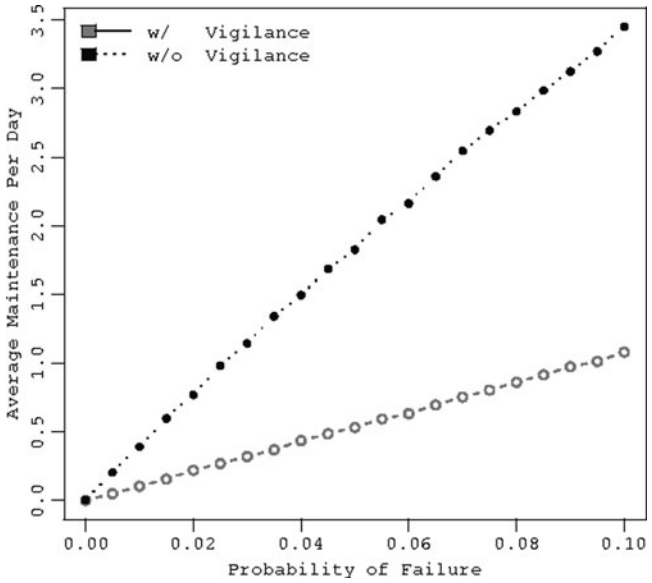


Fig. 6.9 A simulated deployment where the number of sites that require maintenance per day when using Vigilance is compared to a deployment without Vigilance

redundant sensors should be placed where there is most variability. In the case of temperature, if three sensors are to be placed at two different depths (e.g. 2 cm and 8 cm) then the extra sensor should be placed at 2 cm. Similarly for CO_2, an extra sensor should be placed at 8 cm since there is more variation there. Looking at the predicting accuracy for the two temperature and two CO_2 sensors, we see that the 2 cm temperature is the most difficult of the temperature sensors to predict, and the 8 cm CO_2 sensor is the most difficult of the CO_2 sensors to predict. Second, the comparison of *pm* to *spatial* and *temporal* reveal whether spatial or temporal correlations are most important for aiding prediction accuracy. Better fault resilience for CO_2 readings can be achieved by putting more sensors within the same stack of soil, as opposed to instrumenting more sites to increase the number of analogous sensors. Because *spatial*'s performance in the single fault scenarios is poor, this implies that CO_2 may leverage local data more than spatial data. Looking at the double fault scenarios, both *pm* and *temporal* perform comparably, implying that the lack of the companion CO_2 sensor is most detrimental to prediction CO_2. Temperature, on the other hand, achieves a high degree of resilience from a diversity of sites as well as local data. Again, *spatial*'s performance in the single fault scenarios is poor, but *pm* performs well in the double fault scenarios while *temporal* does not, implying that temperature prediction can fall back on spatial data if need be, in addition to other local sensors. In addition, Table 6.1 also indicates weaknesses in a design for a particular application. Not only is it clear that the CO_2 flux application is sensitive to the failure of CO_2 sensors, but the relative importance of the CO_2 sensors is also evident.

There results of Sect. 6.7.4 show that deployment maintenance can be posed as an optimization problem. The settling time is a cost associated with doing maintenance, and there are certainly other costs that we have only briefly touched on, such as the number of maintenance operations and the monetary cost of sensing hardware. In addition to the costs, there is a set of constraints on the applications that are being supported: the confidence interval size must be less than some user specified threshold. Given a set of applications constraints, and deployment costs, it becomes possible to specify a deployment plan as an optimization problem e.g. maximize the amount of usable data collected while minimizing the sensing hardware cost.

6.7.6 System Performance

Here, the processing and memory requirements of Vigilance are shown to be reasonable. For the estimation unit, the time to estimate missing data increases linearly with the number of missing elements, requires only about 30 s for almost 300 missing data points, and is dominated by the time to train a model. For the propagation unit, the time to process a set of candidate vectors increases linearly with the size of the set, and requires about 500 s to process about 5.5 M candidate vectors. Although the size of the set of candidate vectors increases exponentially as the number of faults at a site increases, the operations of the propagation unit can be easily expressed in a Map-Reduce [7] framework, and is therefore highly scalable.

Table 6.2 shows the quartiles of processing time for both the estimation unit and the propagation unit as a function of the number of faults. The number of faults dictates the number of models trained (one model is needed per sensor in these simulations), and the time to train a model is roughly constant when the amount of training data is fixed. The median estimation time increases linearly as the number of models trained increases. The propagation time is a function of the number of imputations that need to be generated, and candidate vectors processed. Because the number of imputations increases exponentially with the number of faults the processing time of the propagation unit increases exponentially. Although an exponential run-time is anathema in systems research, there are two things to keep in mind. First, the maximum number of sensor faults that a node can sustain is small

Table 6.2 The quartiles for estimation time and propagation time as a function of the number of faults, in seconds. The gap size in all cases is one day. The estimation time scales linearly with the number of faults at a site, and the propagation time scales exponentially

	Estimation Time			Propagation Time		
Faults	25%	50%	75%	25%	50%	75%
1	24.70	24.93	30.89	40.99	41.17	41.25
2	42.16	44.08	46.76	44.61	45.24	46.41
3	54.91	56.77	60.36	81.57	81.65	81.66
4	60.83	66.16	74.98	462.6	463.18	467.6

Table 6.3 The quartiles for estimation time and propagation time as a function of the gap size, in seconds. The number of faults in all cases is one. The estimation time and propagation time both scale linearly with the gap size

	Estimation Time			Propagation Time		
Gap Size	25%	50%	75%	25%	50%	75%
5 min	17.44	20.41	24.07	3.44	3.64	3.96
10 min	16.95	19.28	21.41	3.57	3.87	4.02
15 min	16.83	18.96	21.57	3.69	4.02	4.17
30 min	16.49	18.94	21.31	3.61	3.97	4.27
1 h	17.43	19.89	21.66	3.81	4.16	4.43
3 h	17.09	19.17	21.60	4.32	4.66	5.16
6 h	20.70	22.60	23.66	5.28	5.59	6.01
12 h	24.41	25.90	27.90	6.91	7.63	8.04
24 h	27.03	29.05	31.58	10.7	11.6	12.2

(there are only six below ground sensors per site). Second, each imputations could be processed in parallel, and then all the results are collected, using Map-Reduce.

Table 6.3 shows the quartiles of processing time for both the estimation unit and the propagation unit as a function of the gap size, when there is only one fault and the amount of training data is held constant. The time to build a model dominates the estimation time for small gaps, requiring about 17 seconds. When the gap size is larger than six hours, the time to estimate missing data begins to dominate the time to train. Given a fixed number of faults, the number of candidate vectors that the propagation unit needs to process is constant for each time where data is missing. From Table 6.3, the run-time increases linearly for the propagation unit as the gap size increases.

The memory requirements of Vigilance are currently dominated by the amount of sensor data that is stored, and by the amount of memory required to store the fitting residuals. The run-time environment, R, uses about 8 B for each variable. Our data set, made up of about 5,500 observations with 61 values per observation, fits into just over 2.5 MB of memory, and a Polymars model takes almost 3 KB. The residuals associated with the model depends upon the amount of training data, which was bout 5,000 observations. Thus, the fitting residuals required almost as much storage as the data set itself. Fortunately, the distributions of residuals suggest that they can be approximated by a Gaussian distribution, which would reduce the amount of storage needed to about 16 B.

6.8 Conclusion

For many practical embedded sensing deployments, failures are inevitable. In addition, as deployments increase in scale, the odds that no component has failed becomes slim. Similarly, the frequency of maintenance procedures may become

impractical, or demand a batching strategy. Furthermore, the cost of maintenance (which includes the monetary cost, as well as the cost of environmental disturbance) may be too high to pay. Finally, it is not always clear what "redundancy" means for sensor deployments since a sensor's operation is determined by its pose, and no two sensors can have identical pose.

Rather than mask the presence of failed sensors, and the effect of missing data, tools are needed to both help compensate for the loss of data, as well as to estimate the amount of uncertainty introduced by missing data. In our own soil deployment, reluctance to disturb the soil inspired the study of just how important each sensor input was to the scientific applications at hand, and ultimately the development of one solution to the problem of dealing with missing data. Vigilance both exposes the effect of missing data on the actual applications that ecologist are studying, as well as giving technicians expert knowledge about what maintenance is critical to perform. Knowledge of the phenomenon and environmental structure is leveraged by choosing predictors that will likely correlate well with the response, and an adaptive fitting procedure chooses the relevant set of predictors. Vigilance complements these techniques by exposing the severity of faults from the application's perspective. Vigilance's application-centric viewpoint is similar to [28], where the efficacy of a wireless sensor network for monitoring the seismic activity of a volcano is compared to standard seismology sensors.

Vigilance accurately characterizes missing data with a pdf by using a carefully constructed set of predictors that leverages spatial and temporal correlations, in conjunction with Polymars, an adaptive fitting procedure. Vigilance is able to propagate the uncertainty in missing data estimates through an application model, and therefore estimate how the uncertainty due to the absence of requisite inputs affects the certainty of a scientific application. Differences in maintenance requirements across applications and weaknesses in a deployment's design can be identified, and using Vigilance to monitor a deployment can reveal where collocating sensors may be most beneficial. Vigilance paves the way to formalizing the sensor deployment, deployment maintenance, and deployment upgrade processes, casting them as decision problems.

References

1. Ahmadian S, Ko T, Coe S, Hamilton M, Rhimiand M, Soatto S, Estrin D (2007) Heartbeat of a nest: using imagers as biological sensors. Tech. Rep. 1, Center for Embedded Network Sensing
2. Anonymous (2006) The comprehensive r archive network. URL http://cran.r-project.org/
3. Anonymous (2006) Crossbow mica2 wireless measurement system datasheet. URL http://www.xbow.com/Products/Product_pdf_files/Datasheets/Wireless/6020-0042-03_A_MICA2.pdf
4. Anonymous (2006) Decagon ec-5 soil moisture sensor. URL http://www.decagon.com/soil_moisture/ec5/
5. Anonymous (2006) The james san jacinto mountains reserve. URL http://www.jamesreserve.edu/

6. Anonymous (2008) Kepler project. URL http://kepler-project.org/
7. Dean J, Ghemawat S (2004) Mapreduce: Simplified data processing on large clusters. In: Sixth Symposium on Operating System Design and Implementation (OSDI'04)
8. Deshpande A, Guestrin C, Madden S, Hellerstein J, Hong W (2004) Model-driven data acquisition in sensor networks. In: Proceedings of the 30th International Conference on Very Large Data Bases (VLDB'04)
9. Efron B, Tibshirani RJ (1993) An Introduction to the Bootstrap. Chapman and Hall/CRC
10. Houghton RA (2003) Treatise on Geochemistry. Elsevier
11. Kooperberg C, Stone CJ (2004) Comparison of parametric and bootstrap approaches to obtaining confidence intervals for logspline density estimation. Journal of Computational and Graphical Statistics 13
12. Kooperberg C, Bose S, Stone CJ (1997) Polychotomous regression. Journal of the American Statistical Association 92(437):117–127
13. Li M, Ganesan D, Shenoy P (2006) Presto: Feedback-driven data management in sensor networks. In: Proceedings of the Third ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI'06)
14. Ramanathan N, Chang K, Kapur R, Girod L, Kohler E, Estrin D (2005) Sympathy for the sensor network debugger. In: Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems (SenSys'05)
15. Ramanathan N, Schoellhammer T, Kohler E, Estrin D (2008) Fixing faults in wireless sensing systems with confidence. Tech. rep., Center for Embedded Network Sensing
16. Ringwald M, Rmer K (2007) Deployment of sensor networks: Problems and passive inspection. In: The 5th Workshop on Intelligent Solutions in Embedded Systems (WISES'07)
17. Ringwald M, Romer K (2005) Monitoring and debugging of wireless sensor networks. In: GI/ITG Workshop on Systemsoftware for Pervasive Computing, URL http://www.vs.inf.ethz.ch/publ/papers/mringwal-monito-2005.pdf
18. Rossi LA, Krishnamachari B, Kuo CCJ (2004) Distributed parameter estimation for monitoring diffusion phenomena using physical models. In: IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON'04)
19. Rost S, Balakrishnan H (2006) Memento: A Health Monitoring System for Wireless Sensor Networks. In: IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON'06)
20. Schafer JL, Graham JW (2002) Missing data: Our view of the state of the art. Psychological Methods 7(2):147–177
21. Schoellhammer T, Greenstein B, Estrin D (2006) Hyper: A routing protocol to support mobile users of sensor networks. Tech. rep., Center for Embedded Network Sensing
22. Szewczyk R, Mainwaring A, Polastre J, Anderson J, Culler D (2004) An analysis of a large scale habitat monitoring application. In: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys'04)
23. Szewczyk R, Polastre J, Mainwaring A, Culler D (2004) Lessons from a sensor network expedition. In: Proceedings of the 1st European Conference on Wireless Sensor Networks (EWSN'04)
24. Tolle G, Polastre J, Szewczyk R, Culler D, Turner N, Tu K, Burgess S, Dawson T, Buonadonna P, Gay D, Hong W (2005) A macroscope in the redwoods. In: Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems (SenSys'05)
25. Vargas R, Allen MF (2008) Dynamics of fine root, fungal rhizomorphs, and soil respiration in a mixed temperate forest: Integrating sensors and observations. Vadose Zone Journal (7):1055–1064
26. Vargas R, Allen MF (2008) Environmental controls and the influence of vegetation type, fine roots and rhizomorphs on diel and seasonal variation in soil respiration. New Phytologist (179):460–471

27. Werner-Allen G, Johnson J, Ruiz M, Lees J, Welsh M (2005) Monitoring volcanic eruptions with a wireless sensor network. In: In Proc. Second European Workshop on Wireless Sensor Networks (EWSN'05)
28. Werner-Allen G, Lorincz K, Johnson J, Lees J, Welsh M (2006) Fidelity and yield in a volcano monitoring sensor network. In: Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI'06)

Chapter 7
Cane Toad Monitoring: Data Reduction in a High Rate Application

Wen Hu, Nirupama Bulusu, Thanh Dang, Andrew Taylor, Chun Tung Chou, Sanjay Jha, and Van Nghia Tran

Abstract This chapter describes our experiences developing wireless, acoustic sensor network systems for a high rate sensing application: monitoring amphibian populations in northern Australia. Our goal was to use automatic recognition of animal vocalizations to census the populations of native frogs and an invasive introduced species, the Cane Toad. This application falls within the large class of detection and classification applications based on acoustic signals, which also includes condition-based maintenance, vehicle classification, and in particular monitoring birds and animals. As most applications in this class, amphibian monitoring is challenging because it requires high frequency acoustic sampling (10 kHz), complex signal processing and calls for low cost and long-lived unattended system operation in a challenging environment, characterized by significant environment noise as well as flooding, extreme heat and humidity and occasional forest fires. These design and deployment challenges were addressed over several phases. An initial system addressed the challenges of weather-proof, unattended operation. Our second system focused on miniaturization and driving down system costs. Our final system enabled fast in-network frog classification at the motes themselves using compressed sensing. Our experience shows that compressed sensing works in practice and can be a powerful tool in developing and implementing not only cane toad monitoring applications, but other high rate sensing applications.

Keywords Wireless sensor networks · Acoustic sensing · High rate · Hybrid architecture · Machine learning · Lightweight classification · Compressive sensing

7.1 Introduction

This chapter explores the use of low-power, wireless acoustic sensor network technology for monitoring amphibian populations in remote areas of Australia's Northern Territory.

W. Hu (✉)
CSIRO, Brisbane, Australia
e-mail: wen.hu@csiro.au

E. Gaura et al. (eds.), *Wireless Sensor Networks: Deployments and Design Frameworks*,
DOI 10.1007/978-1-4419-5834-1_7,

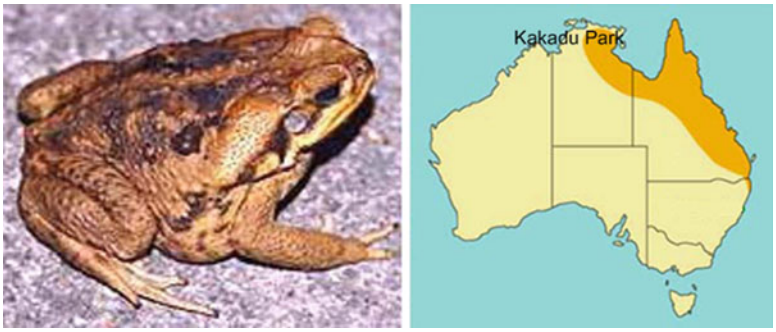


Fig. 7.1 The Cane Toad and its 2003 Australian distribution [17] used with permission copyright ACM 2009

The cane toad (*Bufo marinus*), see Fig. 7.1, was introduced to Australia in the 1930s in the belief that it would control pests in sugar cane crops [20]. Since the introduction of approximately 100 individual cane toads in the 1930s, they have progressively spread through north-eastern Australia and are estimated to number over 200 million in Australia. Their expanding distribution, density and ecological characteristics have raised grave concerns regarding their potential adverse impact on Australia's native fauna. Figure 7.1 illustrates their 2003 distribution. Of particular concern is the ecosystem of Kakadu National Park, a vast World Heritage area, recently colonized by cane toads [7].

A team of zoologists at the University of Queensland, headed by Dr. Gordon Grigg, wanted to study the impact of cane toads on the native fauna. In particular, they focused on the impact of cane toads on native frog species. The reasons for this are multi-fold. There are more than 20 unique frog species in the Kakadu National Park, considered by zoologists to be indicator species of an ecosystem's health. Secondly, native Australian frog species that share diets and habitats with the cane toad are potentially most susceptible to it. One way to characterize the impact of the cane toad on native frog species is to census frog species through acoustic observations. Male frogs sing, and the frog calls, referred to as vocalizations or aural signatures, are distinct for each frog species. However, some unique environmental conditions make it challenging to study cane toad impacts through manual, short field trips. The cane toads are currently active in Australia's remote territories. Most of the frog species are active during the monsoon season, and their activity depends on the rainfall. Since the rainfall is unpredictable, short field trips are not particularly useful. During the monsoon season, roads are often inaccessible. Moreover, extreme humidity and heat conditions make field work challenging. An ideal solution to this problem is a sensor network deployed in the study area that could operate unattended and monitor the impact of cane toads in the areas such as Kakadu National Park from acoustic observations.

In the late nineties, the zoologists approached Andrew Taylor, one of the authors of this chapter and an expert in artificial intelligence, to develop on-line algorithms for automated recognition of frog species, including the cane toad, based on

Table 7.1 Comparison of various systems

Versions	Pilot[29]	Second	Final
Goal	Automated acoustic census of amphibian populations	Miniaturization	Miniaturization
Challenges	Remote, hostile environment significant external noise	Data reduction	Data reduction
Contributions	Weather-proof operation robust classification algorithm	Signal capture on motes hybrid architecture	Lightweight classification

acoustic vocalizations. The goal was to build monitoring stations that could operate unattended and census frog species throughout the year. These monitoring stations would support a multi-year study on the impact of the cane toad on native fauna. The project team consisted of three zoologists and one computer scientist.

In this chapter, we describe the development of an acoustic sensor network system for cane-toad monitoring over several stages, focusing on the motivation for each iteration, and the design decisions made. A unique feature of this application is that the emphasis was always on in situ classification of frog species, rather than on collecting raw high rate acoustic data for future analysis as is the case in the volcano monitoring deployment described in Chap. 4. As far as the zoologists were concerned, the ability to perform data processing at the sensors was a desirable feature right from the outset. Each iteration increased the amount of acoustic data processed at a low power, resource-constrained embedded device, while reducing the amount of data transmitted to, or processed at a resource-rich, higher capability device. The various project stages are contrasted in Table 7.1.

In the pilot deployment starting 1996, Taylor et al., developed software to census frog populations from their vocalizations based on machine learning algorithms [29]. They deployed 12 independent frog monitoring stations, each about 25 km apart, in the Kakadu and the Roper valleys of the Northern Territory. Automatic monitoring of frog species consists of many resource-intensive tasks. The pilot deployment had purely resource-rich devices to monitor cane toads. Each of these monitoring stations contained a solar panel, a battery, power management electronics, a microphone with preamp, a temperature sensor, a rain gauge, and a *Pleb* single board computer [3]. These monitoring stations could operate unattended for a year in hostile conditions, such as flooding, cyclones, humidity and animal interference. The frog species classification software could recognize vocalizations of up to 9 frog species distinguishable to the human ear, and up to 22 frog species in total. However, these monitoring stations had no communications capability amongst themselves. Condition monitoring and data collection were only possible with expensive, typically annual, site visits. The high deployment cost severely restricted the study area.

In 2003, Nirupama Bulusu, Wen Hu and Sanjay Jha began a collaboration with Andrew Taylor to investigate whether it was possible to implement his frog census system on ultra low-power embedded devices such as the motes, that had recently become available. We wanted to focus on miniaturization as a way to drive down deployment costs. Not only was it challenging to implement classical signal

processing techniques on the motes, even sampling the acoustic signal proved to be a considerable hurdle. It appeared that the only way to use the motes was to perform raw data collection and transmit all the raw data back to a base station for further processing. However, communication was well known to be a major energy drain in wireless embedded devices. Thus, our primary design challenge in miniaturization was *data reduction*.

The second system we developed [16] was a hybrid system that consisted of both resource-rich and resource-impoverished sensors, where resource-impoverished sensors extended sensing coverage and were used for simple tasks such as collecting acoustic samples, and resource-rich sensors were used for resource-intensive tasks such Fast Fourier Transforms (FFTs) and greedy decision tree machine learning procedures. To enable the hybrid system, we developed algorithms to account for the sampling, processing, communication and energy bottlenecks of resource-impoverished sensors – (1) *high frequency sampling*, (2) *thresholding and noise reduction*, to reduce data transmission by up to 90%, and (3) *sampling scheduling*, which exploits the sensor network redundancy to increase effective sample processing rate. Machine learning based acoustic classification of frog species required sensors to sample the signal at the Nyquist rate and transfer the samples to higher-capability devices for processing. Unfortunately, this approach was communication intensive and not scalable because even with thresholding, at peak, the sensors sampled the signal at a high rate (10 kHz) and expended a lot of energy and bandwidth in transferring the samples.

The goal of our final system was to further reduce the data transmitted by low-power motes to extend system lifetime. Literature in the area of *compressive sensing* [5] showed that a few random samples could capture the signal structures well, provided that the signal was sparse in some domain. Building on this, for our third system, we designed a light-weight classification algorithm that was executed at the resource-constrained sensors (MicaZ mote class devices), overcoming the limitations of the earlier system. The main intuition for this approach was that animal vocalizations in general, and cane-toad vocalizations in particular, have simple repeated patterns without the variation prevalent in human speech. This implied that simple feature extraction should work for animal classification. Our approach was to estimate the signal envelope from randomly-sampled data at a much lower sampling rate than the signal Nyquist rate and match this envelope with the candidate signal envelope. It required a low sampling rate (100 Hz), small memory (2 kB) and was computationally feasible on resource-constrained sensors. It did not require precise timing, which would have been difficult to implement on resource-constrained sensor devices.

Throughout, our work built on lessons from previous sensor deployments for habitat monitoring [23, 28], health [26], education [27], structural monitoring [24], predictive maintenance [19], volcano monitoring [31] and precision agriculture [10]. Early deployments such as some of those cited above were mostly homogeneous systems and collected simple environmental parameters. More recent sensing systems acquire signals at much higher frequencies, perform complex signal processing, and in situ detection, classification and localization of sources. Like us, some

of these efforts rely on tiered architectures [12, 13] to tackle high rate applications. Our work can be considered complimentary to other systems which support distributed acoustic embedded acoustic sensing applications, such as Acoustic ENSBox [11], VoxNet [2] (see Chap. 4), EnviroMic [22] and Vango [14]. A variety of other techniques, systems and applications of embedded acoustic sensor networks have been reported in the literature and their features inspired various aspects of our work, as surveyed briefly in the reminder of this section. A common feature of these systems is their reliance on powerful nodes to support the data acquisition and extensive processing. In contrast, our experience shows that it is possible to implement high rate sensing and classification applications on low power embedded devices. Similar techniques could potentially be applied to other acoustic classification based sensing applications, such as vehicle classification, condition based maintenance, detection and classification of animal species. However, it is necessary to characterize the sparseness of the signal before random sampling methods can be applied. Additionally, lightweight algorithms must be tested to ensure robustness to noise.

From amongst the works reported, the following are of particular significance:

To reduce the number of transmissions, task decomposition and collaboration have been investigated in a bird monitoring application using acoustic sensor networks [30]. Like us, they try to reduce data and transmissions by preprocessing acoustic data at each sensing node.

Ali et al. [1] developed an acoustic localization algorithm for marmots, medium sized rodents, by combining bearing estimates to the marmot source from multiple sensors. As this is quite a complex algorithm, it has been implemented on powerful platforms, such as the Acoustic ENSBox and VoxNet.

EnviroMic [22] is a distributed recording system to enable scientists to collect bird calls in a study area over a period of time, that they could analyze later. Like us, EnviroMic focuses on miniaturized, low-power embedded devices (motes) for cost-effective monitoring. EnviroMic stores raw data at the motes themselves. If a mote has limited memory, its raw data can be transferred to another mote. Data is not transmitted to a higher capability device for storage, but retrieved by manually collecting the motes at the end of the study. Unlike our application, EnviroMic is not intended to support in situ classification.

Vango [14] is a programming framework to support data reduction in bandwidth constrained sensor networks, by performing in-network processing on low power motes. A program can be specified as a linear filter chain in Vango. This is complimentary to our work in that the matched filter developed in the third iteration of our system could be one of the filters provided by Vango to facilitate rapid deployability of mote-class devices.

Acoustic ENSBox [11] is a multi-sensor system in which each sensor hosts an array of 4 microphones, to support distributed acoustic sensing applications. ENSBox hardware can be used to replace Stargates in either our pure or hybrid system. Therefore, the acoustic ENSBox can be seen as a complement of our work.

More recently, the marmot localization system has been implemented on the VoxNet integrated hardware/software platform for acoustic monitoring applications,

as described in Chap. 5. The VoxNet platform would be very useful for pilot studies, in particular making it less cumbersome to collect and label training data for monitoring frog species. In contrast to marmot localization, our goal is to investigate which parts of application can be offloaded to inexpensive but resource-impoverished Mica motes.

7.2 Background: Pilot Deployment

In this section, we provide a brief overview of the pilot deployment [29], discussing system goals and challenges, hardware used, the frog vocalization recognition algorithm and lessons from the deployment. Our subsequent systems, which focus on low-power embedded devices, build upon this work.

7.2.1 Goals and Challenges

The goals of the project were twofold. The first goal was the design of a software system to enable automated census of amphibian populations. The second goal was the deployment of monitoring stations running this software, that could operate unattended through the monsoon season, eliminating the dependence on field work.

The first goal of developing a software system for automated census of amphibian populations proved to be quite challenging. The key idea was to use acoustic features in the time and frequency domains (see Fig. 7.4) to distinguish the vocalizations of different amphibians. These features included call rate, call duration, amplitude-time envelope, waveform periodicity, pulse-repetition rate, frequency modulation, frequency and spectral patterns. Frog vocalizations are much simpler than human speech but they had to be recognized in very difficult conditions with multiple competing uncooperative "speakers" which were distant from the microphone and with a variety of noise sources such as wind, rain and insects present. A second challenge was the collection of accurately labeled training data that could be used in automated classification algorithms.

7.2.2 Hardware

Twelve monitoring stations were deployed in the Roper and Kakadu valleys of northern Australia and were powered using solar panels. Each station consisted of a single board computer, a microphone, temperature, humidity and rainfall sensors. The Pleb, a single board computer, was built at the University of New South Wales, based on a 200 MHz StrongArm processor [3]. Hardware internals are shown in

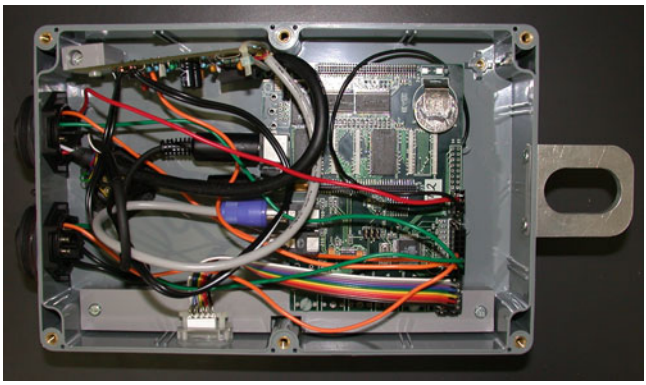


Fig. 7.2 Hardware Internals for the Pilot Deployment (Source: Andrew Taylor)

Fig. 7.3 Station mounted on a Tower (Source: Andrew Taylor)

Fig. 7.2. Data about detected frog species was logged onto flash memory once every 5 min. The stations were mounted on towers (see Fig. 7.3) or tripods to minimize external interference, and packaged to be weather-proof.

7.2.3 Frog Vocalization Recognition Algorithm

The demands of the difficult acoustic environment did not allow the recognition algorithm to segment or isolate individual vocalizations. Recognition consisted of three steps. The first step was generating a time-frequency spectrogram of the

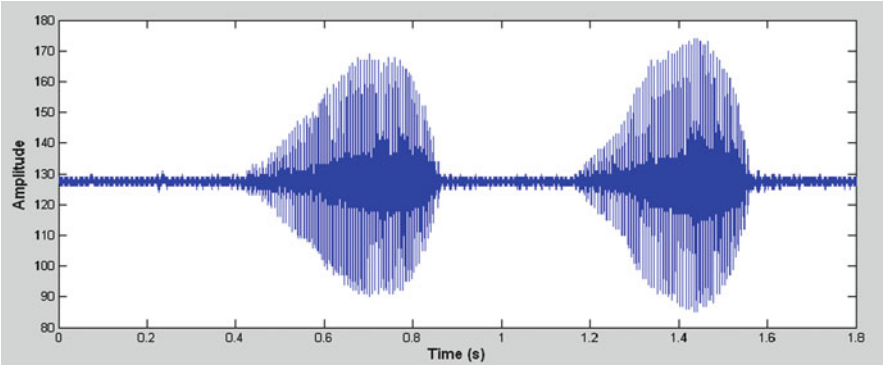


Fig. 7.4 The waveform graph of *Cyclorana cryptotis*. Amplitude (an unsigned ADC reading between 0–255) vs. time (s) [17] used with permission copyright ACM 2009

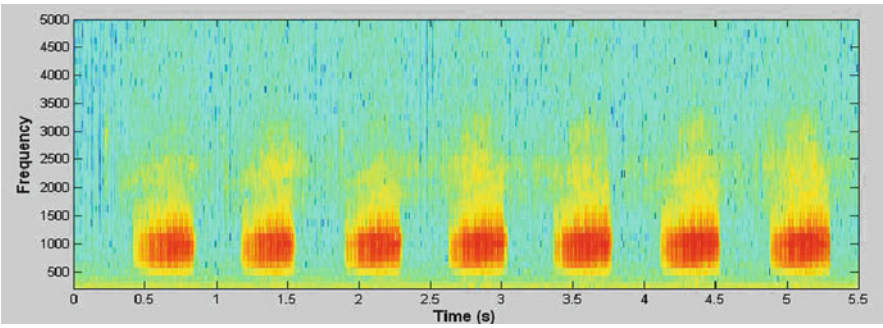


Fig. 7.5 The spectrogram graph of *Cyclorana cryptotis*. Frequency (Hz) vs. time (s) [17] used with permission copyright ACM 2009

acoustic signal. The second step was using the spectrogram as an input for a machine learning algorithm to classify frog species. The third step was a voting process to improve the classification reliability of the machine learning algorithm.

7.2.3.1 Spectrogram Generation

The input signal (the vocalization) was converted into a spectrogram of time-frequency pixels (see Fig. 7.5) by a Fast Fourier Transform (FFT) algorithm.

The spectrogram was broken into slices of 1 ms length. The frog vocalization recognition algorithm examined each slice of the spectrogram and tried to estimate the local peaks. The local peaks are frequency bins that had more energy than neighboring frequency bins. Attributes extracted from occurrences of these local peaks along with attributes extracted from the signal waveform were used to identify individual species of frogs. This was accomplished using classifiers, built by a machine learning algorithm, described next.

7.2.3.2 Machine Learning

Quinlan's machine leaning system, C4.5 [25], was used to build the classifiers. C4.5 is a supervised learning system, requiring training data. This training data was provided based on vocalizations of 22 individual frog species that were manually selected from high quality recordings gathered in earlier biological research. Vocalization of each species used a sound sample of 5–20 s. Additionally training data sounds of cricket species were introduced because crickets were prevalent in the deployment area. As cricket vocalizations share a few qualities with frog vocalizations, classifying them explicitly boosted the reliability of frog species classification.

7.2.3.3 Voting Process

Identification of frog species from individual local peaks only produced a classification accuracy of 50%. To increase the reliability of the system, a hierarchical voting process was used to identify the existence of each frog species (see Fig. 7.6). There were three levels of identifications in the system. For a specific species that had a vocalization lasting for 300 ms and wherein each vocalization consists of a number of mini nodes which were 30 ms long, the system worked as follows. Level 0 was 30 ms long. The identification of a species proceeded to the next level (level 1) of length 300 ms if the number of local peaks occurring within 30 ms exceeded a threshold value. Similarly, the identification process proceeded to level 2 of length 3 s if the local peaks occurring within 300 ms exceeded another threshold value. If a certain number of level 2 vocalizations were identified within 3 s, the species was identified reliably.

7.2.3.4 Evaluation

The algorithm was tested with extensive field data prior to the pilot deployment. The zoologists collected 29 recordings of frog calls in the field area, varying the recording length from 3–30 min, and the distance to the nearest frog from 2–70 meters. Initially, the algorithm was able to correctly classify 8 different species that were distinguishable to humans, and missed one species. In some cases, the calls of two ecologically similar frog species were indistinguishable to humans. These species were grouped together. Additional training data was necessary to remedy this problem.

Additionally, the algorithm performance during deployment was measured. The ground truth was established using expert zoologists. One zoologist visited the monitoring station on wet nights and found that the system detected 12 species observed by him, and missed only one species. It additionally detected 3 species that were not observed by the zoologist, as they were heard very sporadically.

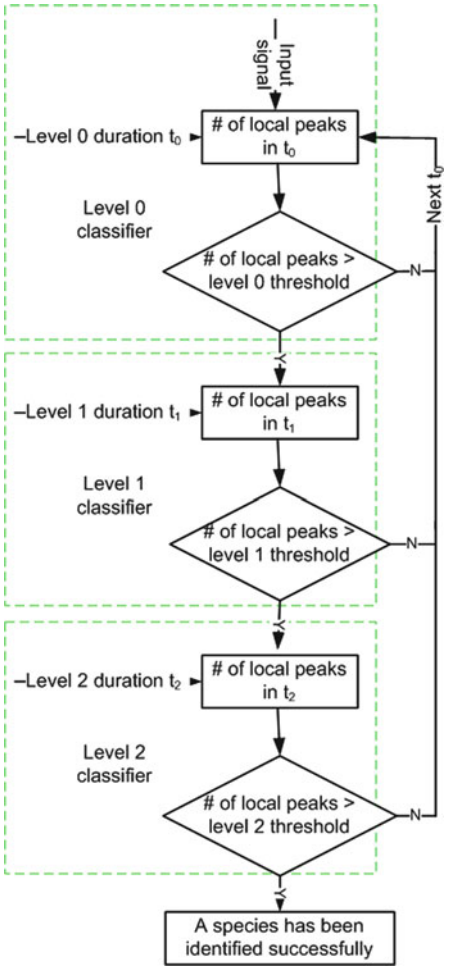


Fig. 7.6 The flow chart of a hierarchical-decision frog species classification algorithm [17] used with permission copyright ACM 2009

7.2.4 Lessons

The Roper and Kakadu valley deployment by Taylor et al. [29] successfully achieved unattended operation for more than an year, withstanding adversarial conditions ranging from human and animal interference to cyclones, monsoon heat and humidity, flooding, and dry season fires. All but one station were found to be working correctly during the first annual visit. Surprisingly, the microphones did not fail despite adversarial conditions. Even though there was frequent lightning in the areas, and the monitoring stations were mounted on tall poles, there were no ensuing failures due to lightning. Whenever there was continuous rain over several days (causing disruption to the solar power supply), the stations simply re-booted and resumed operation without losing their time base. The machine learning algorithm

initially misidentified two species, but after modification of the temporal segments used to recognize the species, performed robustly even in the presence of significant noise and interference.

However, the system had a few limitations. First, the system needs to be pre-trained to recognize the required sounds, from an existing library of recorded sounds. Training can be time-consuming. Secondly, the monitoring stations were very expensive, greatly limiting the scale of deployment. In addition to expensive single board computers, the monitoring stations incurred unexpectedly high costs for large solar panels and packaging, which run into several thousands of Australian dollars.

The latter problem motivated us to investigate whether low-power motes could lower costs. However, motes were very sharply resource-constrained with respect to sampling, processing and storage capability in comparison to single board processors. Not only was it difficult to generate a frequency spectrogram on a mote, even sampling an acoustic signal at 10 kHz was very challenging on the motes. In the next few sections, we describe the development of an acoustic sensor network system for cane-toad monitoring over several stages, reducing in each iteration the amount of data transmitted and increasing the data processed at the low power embedded device.

7.3 Iteration 2: Hybrid Sensor Networks

Following the pilot deployment, we developed a hybrid sensor network for the cane-toad monitoring application during the period July 2004 to December 2005. The hybrid sensor network consisted of resource-rich devices acting as cluster-heads, augmented by resource-poor devices to extend network coverage. This section describes the the goals and challenges, the hardware used and the system architecture, followed by performance evaluation and lessons learned from this iteration of the project.

7.3.1 Goals and Accomplishments

The goal of this iteration was to study whether the pilot system could be made more cost effective using miniaturized sensors based on ultra low-power embedded platforms. The accomplishments of this iteration included the development of a hybrid system prototype that can classify frog species and the addition of ad hoc networking capabilities to the cluster nodes, which enable on-line cluster-node reprogramming, reconfiguration, and real-time result delivery. A final accomplishment is the extension of network coverage by adding inexpensive but resource-impoverished nodes to the system and an evaluation of the hybrid system performance over short-term indoor and outdoor deployments.

7.3.2 Challenges

Classifying frog species using ultra low power embedded platforms was challenging because analyzing frog calls required:

- High Frequency Sampling. To differentiate the calls of cane toads from other 8 native frog species and other environmental noises such as the sound of rain or crickets, the cane toad monitoring system must be able to provide a sampling rate of at least 10 kHz. Note that the 10 kHz sample rate was an empirical result. This sampling rate is extremely challenging on ultra low power embedded platforms.
- Complex Signal Processing. To classify frog species using a machine learning algorithm, the first step is to use a 256-point Fast Fourier Transform (FFT) to produce a spectrogram in the frequency domain from the sampled acoustic inputs in the time domain. The FFT algorithm needs significantly greater computation power and memory than available on ultra low power embedded platforms.

These challenges have been met through a hybrid system prototype. The hybrid system uses the hardware described in the next section.

7.3.3 Wireless Sensor Hardware

We used two hardware platforms for our sensor network: Mica2 and Stargate. Mica2 (see Fig. 7.7) is the third generation of Berkeley mote manufactured commercially by Crossbow [6]. The mote had a 7.7 MHz Atmega processor running TinyOS [15] 1.x and 512 kB on-board flash memory, it could transmit at a maximum data rate of about 19 kB/s and was powered by two AA size batteries. We used the Mica2 sensors as our *resource-poor* sensors.

Stargate, also manufactured by Crossbow, is a high performance processing platform running the ARM-Linux operating system that offered much more resources than the Mica motes in terms of computation power, memory, energy and transmission capabilities. A Stargate had a 400 MHz Intel PXA 255 processor and 96 MB memory in total (64 MB SDRAM and 32 MB flash memory). It could be powered

Fig. 7.7 Mica2 and Stargate [17] used with permission copyright ACM 2009

by a Li-Ion battery and can support Wi-Fi (11 Mbps when using IEEE 802.11 b) transmissions. We used Stargates as our *resource-rich* sensors because they can be interfaced with motes and can therefore communicate directly with the Mica2 sensors over the wireless channel.

The next section describes two system prototypes that were built using the above hardware.

7.3.4 Cane Toad Monitoring Prototypes

7.3.4.1 Pure: Stargates Only

Since our frog-detection system involved many resource-intensive tasks, as a first step, it was natural to use the resource-rich Stargates to build such a system. A Stargate could achieve up to 44 kHz sampling rate, which was more than enough for our system. However, it could only process about 5% of the inputs sampled at 22 kHz in our initial implementation because of its slow floating point emulation. We addressed this problem by using an integer-only Fast Fourier Transform (FFT) implementation, which enabled all inputs to be processed at the 22 kHz sampling rate. This compared well the pilot deployment, which could only process 25% of the inputs [29]. Note that the minimum required sampling frequency was still 10 kHz; the use of 22 kHz sampling with the Stargate was simply due to availability.

Figure 7.8 shows the information processing architecture of the Stargate only system. The Stargate sampled acoustic data using a microphone via a Universal Serial Bus (USB) port. The sound spectrogram was then generated to convert input signals in time domain to frequency domain. The sound attributes, including the number and timing of local peaks, were extracted from the spectrogram and used as the inputs of machine learning classifiers. There was one classifier for each frog species. To increase the correctness and reliability of the recognition, a hierarchical recognition structure was employed, termed as voting process in the figure and described in Sect. 7.2.3.3. Note that the training (classifier building) process was done on a server machine prior to deployment. Then the classifiers were transferred and stored in Stargates.

Moreover, equipped with a wireless transmission channel, our Stargate devices could also communicate and coordinate with each other to form an ad-hoc network. This network provided real time feedback to the user when connected to the Internet. Furthermore, it could estimate the migrate directions of the cane-toad by analyzing the network-wide cane-toad existence snapshots at different times.

7.3.4.2 Hybrid: Stargates and Mica2s

The major problem of the pure system introduced in Sect. 7.3.4.1 was the high cost of the Stargate nodes. Therefore, we introduced a hybrid mixture of Stargates and

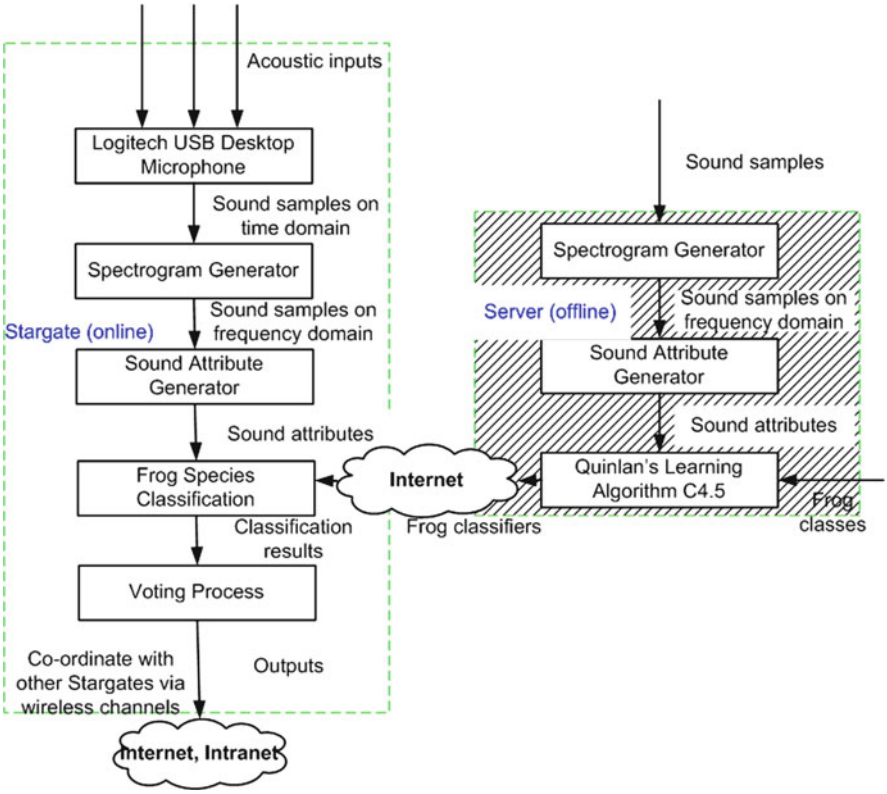


Fig. 7.8 The information processing architecture of the *Stargate only* system. The offline information processing at the server is performed prior to deployment [17] used with permission copyright ACM 2009

Mica2 motes to achieve a more cost-effective system. Mica2s could be scattered to collect acoustic samples. However, it would have been challenging (if not impossible) to implement resource-intensive tasks such as FFT and machine learning procedures on a tiny device such as the Mica2 that had a 7.7 MHz Central Processing Unit (CPU) and 4 kB Random-Access Memory (RAM). Hence, the resource-rich Stargates were used for this task. The Mica2 performed some preliminary processing to reduce the transmission sizes and environmental noise before it transferred the samples to the Stargate. Then the Stargate used these inputs to detect and classify frog species.

Figures 7.9 and 7.10 show the information processing architecture and the network architecture respectively of the Hybrid system. In the hybrid system, the Mica2s sampled acoustic signals, and compressed the samples before sending them to the Stargate via the radio channel. Upon receiving data from the satellite motes, the Stargate decompressed the received data before processing them. These classifications are relayed back to the server.

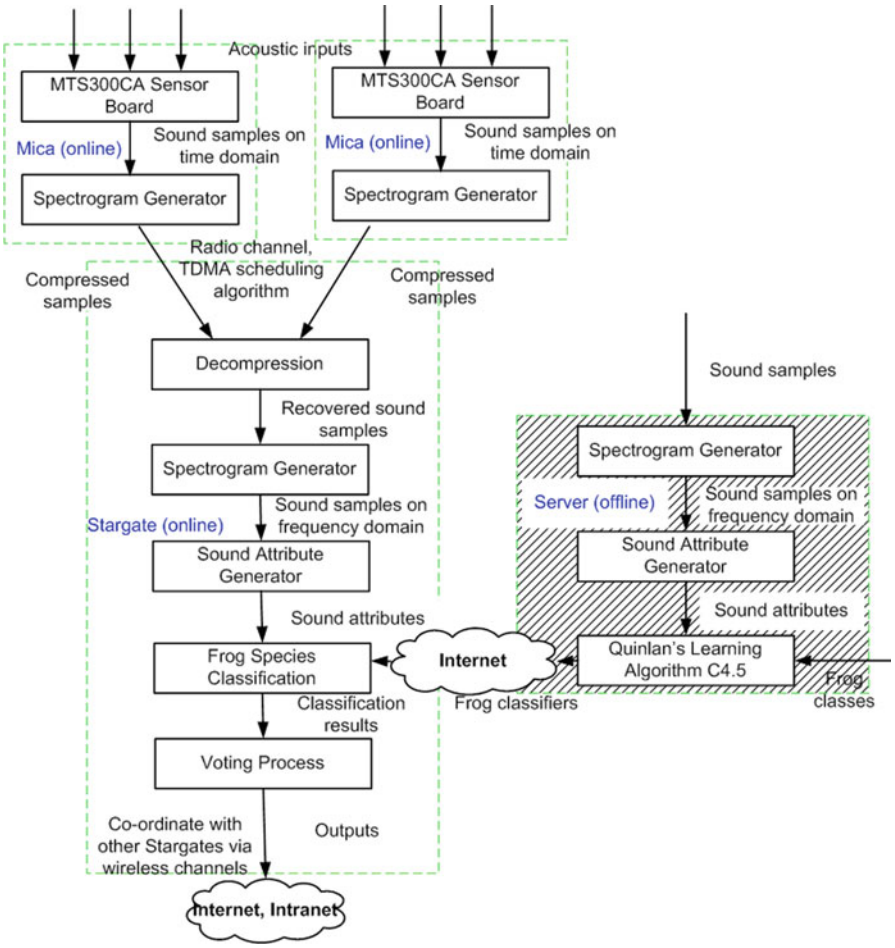


Fig. 7.9 The information processing architecture of the *Hybrid (Stargates and Mica2s)* system. The offline information processing at the server is performed prior to deployment [17] used with permission copyright ACM 2009

Network Architecture and Protocols. We used a cluster-based network architecture in the second iteration of the system. The Stargates act as the cluster heads and the Mica2 motes are the leaf nodes. Although the theoretical data rate of Mica2 nodes is 20 kB/s, we were able to achieve only a data rate of 13.632–14.036 kB/s in our experiments. The packet delivery rate was very high (approximately 99%) when the nodes were very close to each other (10 m). Since the bottleneck of the system was wireless bandwidth (see sampling scheduling in Sect. 7.3.4.2), the Mica2 motes were limited to one hop away from the cluster head. Therefore, no routing protocol was used. The communication schedule inside a cluster was controlled and signaled by the cluster head, i.e., the cluster head polled the Mica2 motes periodically. On the motes, we used Carrier Sense Multiple Access (CSMA), the default Medium Access

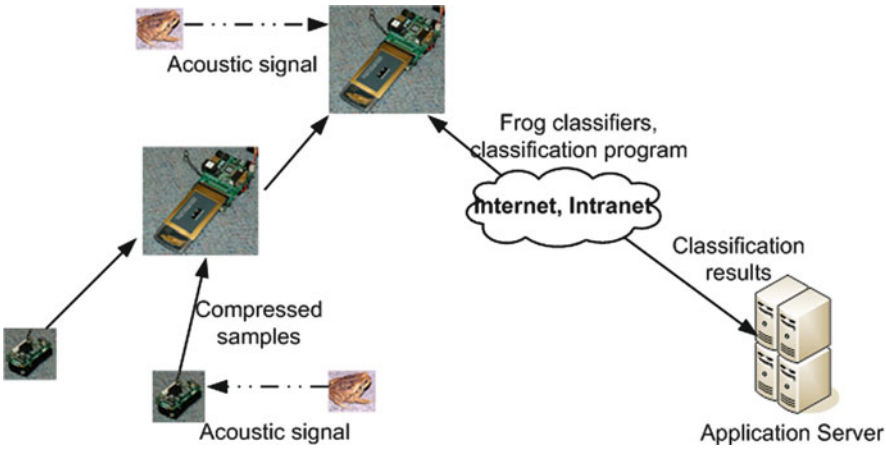


Fig. 7.10 The network architecture of the Hybrid system [17] used with permission copyright ACM 2009

Control (MAC) protocol in TinyOS 1.x, with centralized scheduling by the cluster heads (Stargates). There was no explicit time synchronization across the Mica2s, as we subtracted the recording period (15 s) from the message receive time to obtain the sampling time.

Device Packaging. We found the Pleb package [29] suitable for Stargates. We found HelioMote packaging [21] suitable for the Mica2 motes. Both packages were tested to be water-proof, and suitable for long term outdoor deployment.

We made one modification to the machine learning algorithm used in the pilot deployment. Our hybrid system built one classifier for each frog species vocalization and made a decision about the existence of each frog independently, which was different to Taylor's system that had one classifier for all frog species. To make the hybrid system effective, we further designed and implemented the following algorithms.

High Frequency Sampling. The Mica could sample at up to 200 Hz normally. With the original `HighFrequencySampling` component [18], the Mica could only achieve up to 6.67 kHz sampling rate after turning off the wireless radio while sampling. Because we needed a sampling frequency of at least 10 kHz, we modified the clock rate of the Analog-Digital Converter (ADC) on the sensor board to provide such a high sampling rate. We did not notice any adverse impact on the Micas by changing the clock rates.

Thresholding and noise-reduction. To reduce environmental noise and transmission sizes, we designed and implemented a simple yet effective algorithm. It divided the entire time period into a number of 1 ms time slices. Therefore, there were 10 samples in each time slice when sampling at 10 kHz. If the amplitude level of the whole period was under a threshold (for example, from -20 to $+20$), we called it a *silent/noise-only* period. For a silent/noise-only period, we used one special

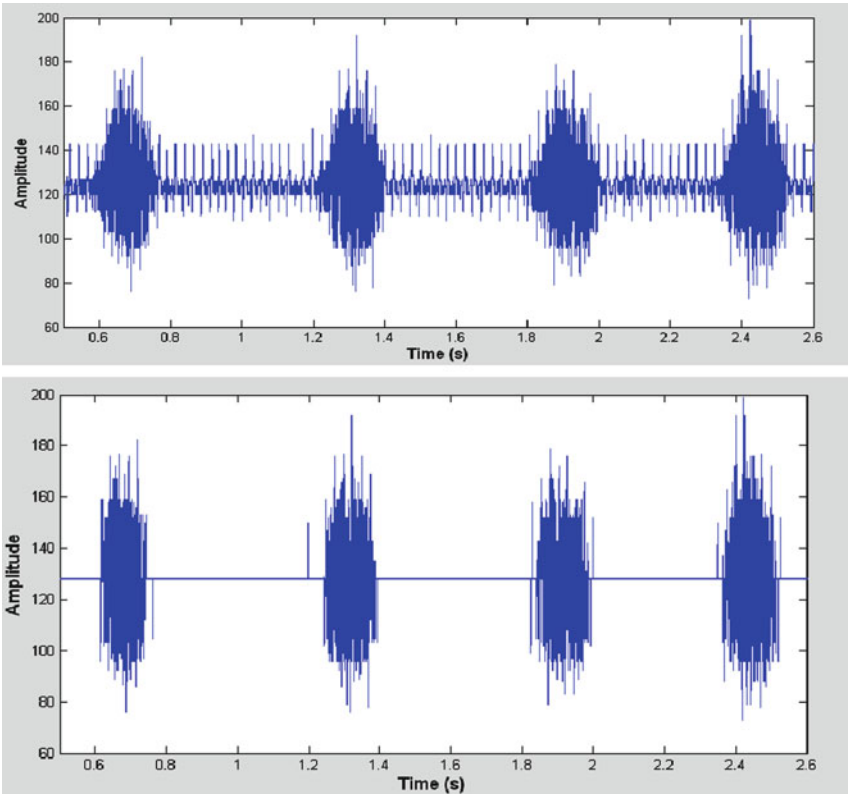


Fig. 7.11 The waveform graphs of *Cyclorana Cultripes* without (*top) and with (bottom) noise reduction. The samples were collected by Mica2s sampling on the field [17] used with permission copyright ACM 2009*

character of one byte to substitute for the whole period of 10 bytes. This reduced transmission sizes by up to 90%. When a Stargate received the packet, it replaced the special character with ten silent values to recover the original signal. The environmental noise was also reduced as shown in Fig. 7.11. The frequency signatures of the frogs' calls were preserved with a carefully chosen silent/noise-only threshold of 32.

Sampling Scheduling. The bottleneck of our hybrid system was the transmission link between Micas and a Stargate. With our thresholding algorithm, it took 30 s to transfer a 15 s segment of acoustic samples, which resulted in about a 30% process rate. To increase the process rate, we designed and implemented a scheduling algorithm which exploited the sensor network redundancy. Based on their locations, two Micas were grouped together if they detected the same acoustic signal. Then, the Stargate controlled the sampling and transferring periods of two Micas such that when one Mica was transferring, the other was sampling. Thus, the processing rate was increased to 50%, which is 60% more than that of a single Mica.

Our expectation was that the hybrid system could perform as well as a resource-rich pure system. This expectation was met, as the results in the next section demonstrate.

7.3.5 Evaluation

To evaluate the performance of our systems, we tested them over a range of scenarios. Our performance metrics included not only criteria such as transmission sizes and operational latency, but also application-determined criteria, in this case, whether the frog species was correctly identified.

- *Test environments*. In our experiments, the playbacks of 9 individual frog species calls and 7 different mixtures of frogs' calls were used as sound sources. Each mixture was created by mixing calls from 2–3 different species of frogs. For 6 of the mixtures, the cane toad was present; in the 7th mixture, cane toad was absent. Within a sound segment, multiple calls from each frog species selected in that particular mixture are present. Our Stargate system consisted of a Stargate with a Logitech USB Microphone that responded to 100–16 kHz frequencies. In the hybrid system, Mica2 used the standard microphone on MTS300CA sensor board. We tested the systems in both indoor and outdoor deployments. The indoor tests were conducted in our lab where external noise was minimal. The outdoor tests were conducted over half a day on a field with environment noises such as insect, bird calls and wind present. The Mica2s were powered using two AA batteries, and the Stargates were powered by four AA batteries. For the outdoor tests, the Mica2s and the Stargates were placed on boxes, at a height of approximately 10 cm from the ground. In both deployments, data logs were downloaded from the Stargate via WiFi.
- *Performance test results*. The test results are summarized in Table 7.2. The data in the table presents results from a series of 6 experiments for each test case. The results were identical across all the experiments. For IND, a trial was *Correct* if

Table 7.2 Test results for our two prototypes with respect to frog species identification [17] used with permission copyright ACM 2009

		Stargate		Hybrid	
		IND	MIX	IND	MIX
Indoor	Correct	9	5	9	5
	Wrong	0	2	0	2
	Cane toad: False positive or negative	0	0	0	0
Outdoor	Correct	9	5	9	4
	Wrong	0	2	0	3
	Cane toad: False positive or negative	0	0	0	0

IND – 9 types of individual frog's call
MIX – 7 types of mixtures of frog's calls

our system identified correctly the frog species; in case of MIX, a *Correct* trial meant all frog species in the mixture were correctly identified by the system. A *Wrong* trial had one or more false positives or false negatives. In all our MIX experiments, we were able to correctly identify the presence of cane toad even when we did not get the other species right. Both indoor and outdoor tests showed that our systems recognized the individual calls of 9 species of frogs successfully. Not surprisingly, it was more difficult to recognize the mixed calls of different frog species. The system gave incorrect results between similar species a few times. The pure Stargate system achieved one more correct recognition outdoors than the hybrid system since it operated at wider frequency ranges. Note that we also tested the pure system sampling at 11 kHz, and the performance results were similar to those of hybrid system. The Hybrid system performed better indoors than outdoors because of outdoor environmental noise. However, it never gave incorrect results for the cane toad species (our principal species, see rows 3 and 6 in Table 7.2) since the cane toad had a very different vocalization compared to the other native species.

- *Transmission sizes*. We studied transmission patterns for the hybrid system during a short-term outdoor deployment in a park, lasting half a day. In this deployment, there were two Mica2s and one Stargate. We collected the calls of frogs in field using Mica2s and stored them as raw data. Then, we used our thresholding algorithm to compress the raw data before transmissions. The results summarized in Table 7.3 show that the algorithm achieved 25–45% compression ratios under different scenarios. The lower bound of the compression ratio was 10% that occurred when the whole sampling period was *silent*. Since the frogs were only active during mid-night, the system operated within that period. The thresholding algorithm was more effective under long periods of silence.
- *Latency*. The Stargate only system provided real time user feedback. The hybrid system had a 45 s latency, which included 15 s sampling time, and 30 s transmission time. This latency was inconsequential for our purposes.
- *Cost*. The costs of the two prototypes have differed greatly since the cost of Mica was projected to drop dramatically. Therefore, we believed that the hybrid model was more suitable for the cane-toad monitoring application.

Table 7.3 Compression ratio observed for different scenarios [17] used with permission copyright ACM 2009

Frog(s)	Original Size	Compression Size	Compression Ratio
1	99,366	26,319	26.59%
2	99,622	25,561	25.66%
3	99,622	32,699	32.82%
4	99,544	36,688	36.86%
5	99,466	41,623	41.85%

1 – *Bufo marinus* call
2 – *Notaden melanoscaphus* call
3 – *Cyclorana cryptotis* call
4 – Mixed sound of 1 and 3
5 – Mixed sound of 1, 2 and 3

7.3.6 Lessons

The central design decision in this iteration was the use of a hybrid system architecture. We also built the pure Stargate only system to transition from the system described in Iteration 1 to a WiFi capable platform and to provide a direct performance comparison to the hybrid system. The hybrid architecture allowed us to augment resource-rich devices with low power embedded devices to extend sensing coverage. We were able to successfully sample the signal on motes and extend process rates using the sampling scheduling algorithms. An unexpected result from evaluating the hybrid system was that its classification performance was slightly less robust to outdoor noise than the pure system, even when it was sampling at comparable frequency and using the same classification algorithm.

The pure Stargate only system was ready to be deployed with a well-designed outdoor package, a good water-proof microphone with pre-amp to increase sensing range from approximately 3–100 m, an external antenna to increase transmission range, a high volume battery and a 200 mA solar panel with solar recharging circuit. However, the hybrid system needed further engineering efforts for robust, long-term deployment. In the hybrid system, the motes also needed a pre-amp.

Another major challenge for outdoor sensor deployment was energy conservation at the motes. The battery energy of the motes depleted very quickly. This was because the machine learning algorithm for acoustic species classification required motes to sample the signal at the Nyquist rate (10 kHz) and transfer the samples to Stargates for classification. Unfortunately, this approach was energy and bandwidth intensive because the motes expended a lot of energy and bandwidth in transferring the samples to a Stargate over the wireless channel. Adding solar panels to the motes to harvest energy could not completely remedy the problem, because of the irregularity of power supply, and because the frog species typically are most active at night. In retrospect, to prepare the system for deployment, we could have simply added large flash memories to the motes themselves to temporarily store data during periods of low energy availability. However, since raw acoustic data would have been recorded at a very high rate, memory requirements would have been very large. These problems motivated us to address the design challenge of signal processing on the mote itself. In the next section, we describe our final system which features lightweight acoustic classification on the mote.

7.4 Iteration 3: Lightweight Classification

In our final system design, our goal was to develop a lightweight acoustic classification algorithm that could be implemented at the resource-constrained motes. This had the potential to preserve the energy of the motes, by reducing the raw data transmitted by the motes to prolong the sensor network lifetime.

The Nyquist-Shannon sampling theorem posits that if a function $x(t)$ contains no frequencies higher than C Hertz, it is determined completely from its ordinates at a series of points spaced $1/2\,C$ apart. Thus, the Nyquist rate of sampling a signal is given by twice its highest frequency component. However, recent results in *compressive sensing* theory [5] have shown that a small number of random samples (much lower than the Nyquist-Shannon criteria) could capture the signal structures well, provided that the signal was *sparse* or *compressible*, that is, the signal contained many coefficients close to zero when represented in some domain (e.g., Fourier sinusoids, time domain).

Building on this result in compressive sensing, two proposed approaches [4, 9] have performed detection and classification on randomly sampled data, without actually reconstructing the signal. Boyle et al. [4] showed that we can classify signals from the histogram of the signal random samples. Unfortunately, the histograms were difficult to recover under noise. We also considered the smashed filter [9] to classify signals using random samples. The smashed filter is a variant of the matched filter, a popular technique in signal processing that calculates the correlation between a known signal and an unknown signal to verify if the unknown signal matches the known signal. However, it was computationally infeasible to use the smashed filter on resource-constrained devices for a large number of samples.

To overcome the limitations of previous approaches, we designed a light-weight classification algorithm that could be executed at the resource-constrained MicaZ mote class devices. The main intuition for this approach was that animal vocalizations in general, and cane-toad vocalizations in particular, have simple repeated patterns without the variation prevalent in human speech. This implied that simple feature extraction should work for animal classification. Our approach was to estimate the signal envelope from randomly-sampled data (without explicitly reconstructing the signal) and match this envelope with the candidate signal envelope. As shown in Sect. 7.3, Nyquist rate sampling at 10 kHz required the mote to suspend other tasks. Our approach differed from using a low pass filter or a uniform sampling rate in that we can capture a similar amount of information by using significantly fewer random samples than by using uniform samples. This approach required a low sampling rate, modest computation and memory resources, enabling resource-constrained sensors to classify high frequency signals. It also did not require precise timing, which was difficult on resource-constrained sensor devices.

Our classification algorithm had three main steps – random sampling, envelope extraction, and matched filtering (Fig. 7.12).

7.4.1 Random Sampling

We believed random sampling to be suitable for low-power sensor devices such as motes, because it is fairly straightforward to program the sensors to compute a random sequence of time a priori and schedule sample times according to the

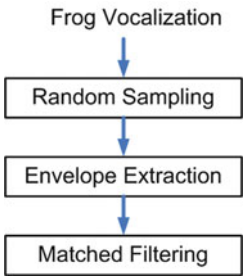


Fig. 7.12 Lightweight acoustic classification [8] used with permission copyright IEEE 2008

sequence. However, timing control can be difficult. If two sampling times in the random sequence are extremely close (say 10 and 10.00001 s), the sensor must sample at a very high frequency (100 kHz) for the short time period (between 10 and 10.00001 s). To overcome this, we generated random indices in a short time range (for example, (0, 1)) and scaled them to a much larger time range (for example, (0, 100)). The scale factor was the shortest time period between two samples.

The sampling time was chosen randomly in a short time period τ and dilated to the full time scale Γ. The sampling time was determined by the following equation:

$$sampleTime = randsequence + jitter,$$

where *randsequence* was determined by the equation:

$$randsequence = M \times randsample(n/M, m),$$

where n was the signal length, M was the down sampling factor, m was the total number of measurements, and M was the minimum time interval between each sample. Function *randsample* (a, b) picks b arbitrary numbers with an independent and identical distribution (i.i.d) from 1 to a. We selected m numbers randomly from the time scale τ, $[1, n/M]$ and dilated the numbers to the time scale Γ, $[1, n]$ by multiplying each number by M. Each number corresponds to the time when a sample should be taken. The sampling times were random from 1 to n and at least M time units apart. This ensured that we not only collected fewer samples but also sampled at a rate M times lower than the Nyquist rate.

To increase the sampling time randomness, we introduce *jitter*. We generate *jitter* using the equation:

$$jitter = s \times round(randn(m, 1))$$

where s was the scaling factor that controls how large the jitter was. Function *randn* (a, b) generates an $a \times b$ matrix with random entries, chosen from a normal distribution with zero mean and variance one. s should not be too large because it might create two samples that are very close together and require the sensor to sample at a high rate. s also should not be too small because after rounding, it might not create any jitter. We empirically choose $s = 2$ in this application.

7.4.2 Envelope Extraction

The envelope of a signal $x(t)$ was the boundary within which the signal was contained. The signal envelope could be estimated by applying a low-pass filter or smoothing the signal. We simplified the computation by just windowing the signal and taking the maximum absolute value of the samples in each non-overlap window.

We subdivided the recorded samples $x(t), 0 \leq t \leq n-1$ into K non-overlapping smaller segments as follows:

$$x_i(t) = x(iL + t)w(t)\ 0 \leq t \leq L-1, 0 \leq i \leq K-1$$

where $w(t) = 1$ for $0 \leq n \leq L-1$ was basically a rectangular window of duration L. L should not be too large to preserve enough points in the envelope. L also should not be too small to improve the robustness of envelop estimation. In this application, L can be varied from 10 to 100 without impacting the accuracy of envelope estimation.

The envelope was then estimated as:

$$y(i) = \max_{i=0}^{K-1} x_i(t)$$

Sensors could compute the envelope very efficiently because they only needed to examine the maximum value of the incoming samples in each window. Therefore, sensors could compute the envelope as they sampled the signal. In general, a frog often makes several calls continuously. Hence, we used a threshold ε to separate each call. Each envelope was normalized to have unit energy. The set of envelopes of frog calls was denoted as $y^m(i)$ where m was the discrete time index of the call. Figure 7.13 shows an example of frog calls and the envelopes. Using the same techniques, we also extracted the benchmark envelopes of all the frog species using the training signals. We denoted this set of envelopes as $Y_k^m(i)$ where k was the index of the frog species.

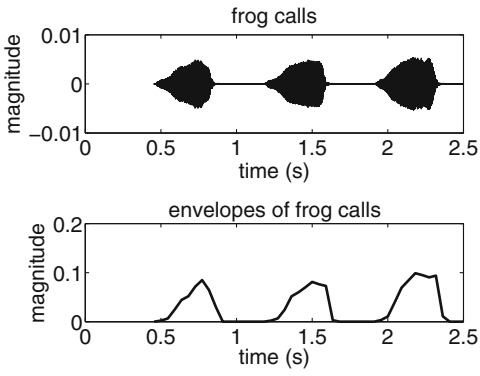


Fig. 7.13 Example of a frog call and three extracted envelopes [8] used with permission copyright IEEE 2008

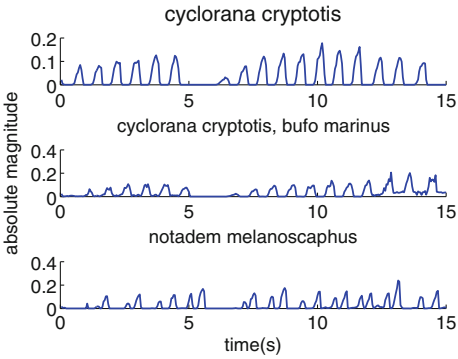


Fig. 7.14 Envelopes of toad calls [8] used with permission copyright IEEE 2008

7.4.3 Matched Filter

After the set of frog call envelopes $y^m(i)$ were recorded, we applied the matched filter of $y^m(i)$ with $Y_k^m(i)$ to detect the frog species. Figure 7.14 shows the call envelopes of different fog species, which were used as an input to the matched filter. The matched filter, used extensively in signal processing and communication to detect a signal, could be performed efficiently by convolution. The classified frog species k was defined as:

$$k = \arg_k \max \sum_{1}^{m} y^m(i) * Y_k^m(-i),$$

where $*$ denotes the convolution operator, which is defined as

$$(f * g)(n) = \sum_{m=-\infty}^{\infty} f(m)g(n-m),$$

where f and g were two signals; m is the discrete time index of the signals; n is the discrete time index of the correlation vector; and, k could be interpreted as the frog type whose calls have the maximum correlation with the benchmark calls.

7.4.4 Evaluation

We conducted several experiments indoors and outdoors during May 2008 to study the effectiveness of our algorithm. Here, we report on the results from indoor experiments conducted on May 15th, 2008. Depending on the frog type, we used between 5–20 values to represent a frog call envelope. The benchmark envelopes can be stored using less than 2 kB of memory. We recorded the songs from three different frog species and mixed them randomly. Each song is about 60 s. Then, we performed

random sampling on the motes, extracted the recorded signal envelopes and ran the matched filter. For each signal, we iterated 50 times by generating 50 random sample sets. We varied the down sampling factor from 5 to 200 in steps of 5. We measured the following metrics:

- True positive rate, the percentage of times that the frogs were correctly classified
- False negative rate, the percentage of times that the frogs were present but could not be detected
- False positive rate, the percentage of times that the frogs were absent but were detected

In general, we wished to have a high true positive rate and a low false negative rate. The low false negative rate was critical because we did not want to miss a frog. Ideally, the false positive rate should also be low. However, it was less critical if this rate was high because sensors could trigger a more capable server to verify the result. It was preferable to detect a frog incorrectly rather than miss a frog.

In the first set of results, we could see that even with a down sampling factor of more than 100, the true positive rates were above 90%. The true positive rates for *Bufo marinus* and *Notaden melanoscaphus* were always 100% (Fig. 7.15). These rates degraded slowly as the down sampling factor approached 200. In Fig. 7.16, the false negative rates start at about 5% and increase gradually to 20% as the down sampling factor increased from 5–200. Notably, the false positive rates were high: about 50% for *Bufo marinus* and *Cyclorana cryptotis* (Fig. 7.17). However, as we mentioned before, a false detection of a frog could be potentially verified by triggering a higher computation backend server to check the result.

7.4.5 Lessons

Our experience showed that random sampling could be successfully applied in practice to reduce the number of required measurements for a high rate sensing application, provided that the signal was compressible in some domain. Fortunately,

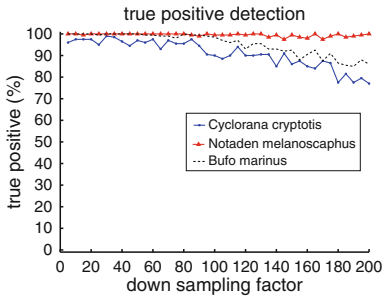


Fig. 7.15 True positive classification rate when down sampling factor varies from 5–200 [8] used with permission copyright IEEE 2008

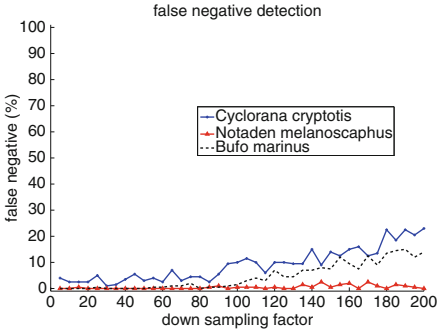


Fig. 7.16 False negative classification rate when down sampling factor varies from 5–200 [8] used with permission copyright IEEE 2008

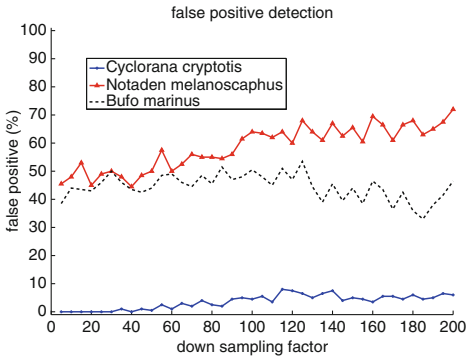


Fig. 7.17 False positive classification rate when down sampling factor varies from 5–200 [8] used with permission copyright IEEE 2008

this happens to be true not just for frog species, but for many animal signals (crickets, bird calls, etc.). With our lightweight classification algorithm, it was possible to process all data on the low power embedded device itself.

However, there was a trade-off between classification accuracy and the down sampling factor. We did not expect false positives to be as high as they were, relative to the machine learning algorithm used in previous iterations of the system. We can use more features in the signal, such as a complete frequency representation of the signal, which may require a higher sampling rate, to improve the classification accuracy. We also did not expect the down sampling factors for given detection rates to vary significantly across different frog species. In retrospect, both results seem fairly obvious. This is because the machine learning algorithm is adaptive to noise, and is as such inherently more robust than the matched filter. The down sampling factor depends to some extent on the compressibility of the source signal (e.g. frog call), or the minimum number of coefficients needed to represent the signal. Typically, the minimum number of coefficients can be empirically determined by analyzing recordings of the frog call. Since the source signal is different for different frog species, the down sampling factor can be different.

For any objects or animal species that must be detected using vocalizations, if their aural signatures are compressible, then the detection could ostensibly be performed on a mote, without using a single board computer. If high detection

accuracy is needed, then these detections should be verified using higher capability devices. We are currently investigating approaches that dynamically adjust the down sampling factor in response to the signal being measured, rather than using preset factors.

7.5 Conclusions

Cane toad impact monitoring based on aural call patterns is characterized by high frequency sampling, complex signal processing for in situ classification, wide-area sensing coverage and long-lived unattended operation requirements. A pilot deployment of single-board computer based monitoring stations in the northern territories of Australia was unexpectedly robust to weather conditions, however it was expensive as it featured powerful computing devices and expensive packaging.

In the second phase, we built a hybrid sensor network of low-power motes and more powerful Stargates for monitoring amphibian species. Our system could recognize the calls of several frog species in northern Australia. A thresholding and noise-reduction algorithm reduced data transmission sizes by up to 90%. To enlarge the sampling frequency for a given monitoring period, we designed a sampling scheduling algorithm which exploited the redundancy of sensor networks and increased the system process rate by up to 60%. System evaluation of indoor and outdoor deployments demonstrated the feasibility and usability of a hybrid systems approach in terms of minimizing cost while extending sensor coverage. However, for long term outdoor deployments, the variability of renewable energy supply makes the use of motes merely for collecting and transferring high rate data impractical.

In the latest phase, we developed a lightweight algorithm to classify cane toads at the motes themselves, addressing the above problem. It built on compressive sensing theory which showed that a small number of random samples could embed the signal structures well. In contrast to prior work, the algorithm could be readily implemented on sharply resource-constrained sensors such as the MicaZ motes, without any specialized hardware (e.g. DSP chips). It required only about 2 kB of memory for classification and allowed sensors to sample an acoustic signal at a rate 100 times lower, than the original 10 kHz Nyquist rate of the signal. Our experimental results on real world cane-toad vocalizations showed that we could classify cane toads with more than 90% accuracy and less than 5% miss rate. This algorithm clearly was not the optimal algorithm for detection and classification in general. However, this algorithm had the potential to significantly extend sensor network lifetime, by enabling motes to perform the classification themselves. It also minimized classification time to under a second, as opposed to the 45 s latency incurred in the hybrid system developed in the second iteration.

Our experience shows that the theory of compressive sensing has practical benefits and that miniaturization is possible for high data rate sensing applications, including acoustic applications, such as vehicle classification, condition based

maintenance, and detection of birds, cetaceans and bats. However, it is necessary to characterize the sparseness of the signal before random sampling methods can be applied. Ideally, the signal sparseness and sampling frequency should be adjusted dynamically. Additionally, lightweight algorithms must be tested to ensure robustness to outdoor noise.

The machine learning algorithms and lightweight classification algorithms used to classify amphibian species were robust. However, both these types of algorithms need to be trained using preliminary data which can be cumbersome. The development of machine learning algorithms that do not require pre-training and can categorize and store sounds for post-hoc identification would make these systems more readily deployable. Scientists can share bioacoustic libraries of categorized sounds (e.g. birds, frog calls, crickets) using sound wikis that can be used to monitor large proportions of fauna, potentially revolutionizing biodiversity monitoring.

We have shared our experiences designing and deploying an acoustic sensor network for cane-toad monitoring. We believe that hybrid architectures can be a cost-effective method for implementing high rate sensing applications. However, they are only scalable and cost-effective when the ratio of resource-poor sensors to resource-rich sensors is high. We believe that data reduction at resource-poor sensors, enabled by advances in compressed sensing, offers a lot of potential in improving the scalability of hybrid solutions.

In our case, the scientific objectives of the zoologists were known in advance. This allowed us to design a system based on intensive data processing at the sensors, rather than data recording. Starting with more powerful computing platforms allowed the development of robust detection and classification systems that were immediately useful to zoologists, and then allowed us to focus on how to migrate such functionality into low-power computing platforms that could be cost-effective.

References

1. Ali AM, Yao K, Collier TC, Taylor CE, Blumstein DT, Girod L (2007) An empirical study of collaborative acoustic source localization. In: IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks, ACM, New York, NY, USA, pp 41–50, DOI http://doi.acm.org/10.1145/1236360.1236367
2. Allen M, Girod L, Newton R, Madden S, Blumstein DT, Estrin D (2008) Voxnet: An interactive, rapidly-deployable acoustic monitoring platform. In: IPSN '08: Proceedings of the 7th international conference on Information processing in sensor networks, IEEE Computer Society, Washington, DC, USA, pp 371–382, DOI http://dx.doi.org/10.1109/IPSN.2008.45
3. Anonymous (2008) The pleb project. http://www.ertos.nicta.com.au/hardware/pleb/
4. Boyle F, Haupt J, Fudge G, Nowak R (2007) Detecting signal structure from randomly-sampled data. In: Proceedings of IEEE Workshop on Statistical Signal Processing (SSP), Madison, Wisconsin, pp 326–330
5. Candes E, Romberg JK, Tao T (2006) Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information. IEEE Transactions on Information Theory 52(2):489–509
6. Crossbow Technology Inc (2010) http://www.xbow.com

7. van Dam RA, Walden DJ, Begg GW (2002) A preliminary risk assessment of cane toads in kakadu national park. Scientist Report 164, Supervising Science, Darwin NT, Australia
8. Dang T, Bulusu N, Hu W (2008) Lightweight acoustic classification for cane-toad monitoring. In: Proc. 42nd Asilomar Conf. on Signals, Systems and Computers, CA, USA, pp 1601–1605, DOI 10.1109/ACSSC.2008.5074693
9. Davenport M, Duarte M, Wakin M, Laska J, Takhar D, Kelly K, Baraniuk R (2007) The smashed filter for compressive classification and target recognition. In: Proceedings of Computational Imaging V at SPIE Electronic Imaging, San Jose, California, pp 326–330
10. Estrin D, Girod L, Pottie G, Srivastava M (2001) Instrumenting the world with wireless sensor networks. In: Proceedings of the International Conference on Acoustics, Speech and Signal Processing, Salt Lake City, Utah
11. Girod L (2005) A self-calibrating system of distributed acoustic arrays. PhD Thesis, UCLA, Los Angeles, CA, USA
12. Girod L, Jamieson K, Mei Y, Newton R, Rost S, Thiagarajan A, Balakrishnan H, Madden S (2006) Wavescope: a signal-oriented data stream management system. In: SenSys '06: Proceedings of the 4th international conference on Embedded networked sensor systems, ACM, New York, NY, USA, pp 421–422, DOI http://doi.acm.org/10.1145/1182807.1182886
13. Gnawali O, Jang KY, Paek J, Vieira M, Govindan R, Greenstein B, Joki A, Estrin D, Kohler E (2006) The tenet architecture for tiered sensor networks. In: SenSys '06: Proceedings of the 4th international conference on Embedded networked sensor systems, ACM, New York, NY, USA, pp 153–166, DOI http://doi.acm.org/10.1145/1182807.1182823
14. Greenstein B, Mar C, Pesterev A, Farshchi S, Kohler E, Judy J, Estrin D (2006) Capturing high-frequency phenomena using a bandwidth-limited sensor network. In: SenSys '06: Proceedings of the 4th international conference on Embedded networked sensor systems, ACM, New York, NY, USA, pp 279–292, DOI http://doi.acm.org/10.1145/1182807.1182835
15. Hill J, Szewczyk R, Woo A, Hollar S, Culler DE, Pister KSJ (2000) System architecture directions for networked sensors. In: Proc. 9th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems, Boston, MA, USA, pp 93–104
16. Hu W, Tran VN, Bulusu N, tung Chou C, Jha S, Taylor A (2005) The design and evaluation of a hybrid sensor network for cane-toad monitoring. In: Proceedings of the Fourth Information Processing in Sensor Networks (IPSN/SPOTS), Los Angeles, CA
17. Hu W, Bulusu N, Chou CT, Jha S, Taylor A, Tran VN (2009) Design and evaluation of a hybrid sensor network for cane toad monitoring. ACM Trans Sen Netw 5(1):1–28, DOI http://doi.acm.org/10.1145/1464420.1464424
18. Kim S, Culler D, Demmel J (2004) Structural health monitoring using wireless sensor networks. Berkeley Deeply Embedded Network System Course Report
19. Krishnamurthy L, Adler R, Buonadonna P, Chhabra J, Flanigan M, Kushalnagar N, Nachman L, Yarvis M (2005) Design and deployment of industrial sensor networks: experiences from a semiconductor plant and the north sea. In: SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems, ACM, New York, NY, USA, pp 64–75, DOI http://doi.acm.org/10.1145/1098918.1098926
20. Lever C (2001) The Cane Toad. Westbury Publishing, West Yorkshire
21. Lin K, Yu J, Hsu J, Zahedi S, Lee D, Friedman J, Kansal A, Raghunathan V, Srivastava M (2005) Heliomote: enabling long-lived sensor networks through solar energy harvesting. In: SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems, ACM, New York, NY, USA, pp 309–309, DOI http://doi.acm.org/10.1145/1098918.1098974
22. Luo L, Cao Q, Huang C, Wang L, Abdelzaher TF, Stankovic JA, Ward M (2009) Design, implementation, and evaluation of enviromic: A storage-centric audio sensor network. ACM Trans Sen Netw 5(3):1–35, DOI http://doi.acm.org/10.1145/1525856.1525860
23. Mainwaring A, Culler D, Polastre J, Szewczyk R, Anderson J (2002) Wireless sensor networks for habitat monitoring. In: WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications, ACM, New York, NY, USA, pp 88–97, DOI http://doi.acm.org/10.1145/570738.570751

24. Mechitov K, Kim W, Agha G, Nagayama T (2004) High-frequency distributed sensing for structure monitoring. In: Proceedings of the First International Workshop on Networked Sensing Systems, Tokyo, Japan
25. Quinlan JR (1993) C4.5: programs for machine learning. Morgan Kaufmann Inc., San Francisco
26. Schwiebert L, Gupta SK, Weinmann J (2001) Research challenges in wireless networks of biomedical sensors. In: Proceedings of the 7th annual international conference on Mobile computing and networking, ACM, Rome, Italy, pp 151–165, DOI http://doi.acm.org/10.1145/381677.381692
27. Srivastava M, Muntz R, Potkonjak M (2001) Smart kindergarten: sensor-based wireless networks for smart developmental problem-solving enviroments. In: Proceedings of the 7th annual international conference on Mobile computing and networking, ACM, Rome, Italy, pp 132–138, DOI http://doi.acm.org/10.1145/381677.381690
28. Szewczyk R, Osterweil E, Polastre J, Hamilton M, Mainwaring A, Estrin D (2004) Habitat monitoring with sensor networks. Commun ACM 47(6):34–40, DOI http://doi.acm.org/10.1145/990680.990704
29. Taylor A, Grigg G, Watson G, McCallum H (1996) Monitoring frog communities: an application of machine learning. In: Proceedings of the 8th Innovative Applications of Artificial Intelligence Conference, AAAI, Menlo Park, CA, USA, pp 1564–1569
30. Wang H, Estrin D, Girod L (2003) Preprocessing in a tiered sensor network for habitat monitoring. EURASIP JASP (special issue of sensor networks) 4:392–401
31. Werner-Allen G, Lorincz K, Ruiz M, Marcillo O, Johnson J, Lees J, Welsh M (2006) Deploying a wireless sensor network on an active volcano. IEEE Internet Comput. (Special Sensor Nets Issue) 10(2):18–25

Chapter 8
ExScal: Dealing with Scale[1]

Vinayak Naik and Anish Arora

Abstract ExScal (for Extreme Scale) project deployed a 1,000+ mote network and a 200+ peer-to-peer ad-hoc 802.11 b network of stargates in a 1.3 km by 200 m remote area in Florida, USA during December 2004. In comparison with deployments of networks of mote and stargate-class devices fielded to date, the ExScal application remains relatively complex and large in size. The goal of the project was to deploy a dense wireless sensor network "tripwire" that detects, tracks, and classifies multiple intruders of different types, such as people and vehicles, in a long perimeter region. Networks such as this are envisioned to protect pipelines that are vulnerable to sabotage and borders between nations that are prone to illegal crossing.

The primary application requirements were as follows: (1) low cost over the duration of deployment, (2) accuracy and timeliness of detection, and (3) low human effort over the duration of deployment. The chapter describes how these requirements were met. It justifies the choice of sensing and communication modalities and packaging of nodes to meet the low cost over the desired lifetime of the network; it describes the topology of deployment and systems' software architecture chosen to meet the accuracy and timeliness of detection; finally, it describes the strategy used for node emplacement and the software tools components devised to handle faults, in order to lower the human effort. We report our processes of evolving the system from design to deployment emphasizing the role of simulation and testing in that process. We end the chapter by reporting the observed faults, their distribution, and the accuracy of intrusion detection achieved by the deployed application.

Keywords Multisensor systems · Computer networks · Routing · Network fault diagnosis · Network fault tolerance · Wireless sensor networks · Distributed computing · Signal processing · Embedded systems

[1]Based on "ExScal: Elements of an Extreme Scale Wireless Sensor Network", by Arora A et al which appeared in The 11th International Conference on Embedded and Real-Time Computing Systems and Applications.

V. Naik (✉)
Indraprastha Institute of Information Technology, Delhi, India
e-mail: naik@iiitd.ac.in

E. Gaura et al. (eds.), *Wireless Sensor Networks: Deployments and Design Frameworks*,
DOI 10.1007/978-1-4419-5834-1_8,

8.1 The ExScal Application and Its Demands

The ExScal project aimed to deploy a dense wireless sensor network "tripwire" that detects, tracks, and classifies multiple intruders of different types (such as people and vehicles) over a long region's perimeter. Applications of this concept are envisioned for protection of pipelines that are vulnerable to sabotage, borders between nations that are prone to illegal crossing, and areas abutting critical plants/thoroughfares that are vulnerable to terrorist threat. The primary requirements identified for this applications type are:

1. *Low cost over the duration of deployment*. This translates to selection of: sensing and communication modalities that have desirable range and enable low power operation, appropriate packaging, as well as node layouts that avoid nodal redundancy. Mission lifetime is 1–6 months.
2. *Accuracy and timeliness of detection*. To achieve accuracy and timeliness, we must have a low false alarm and low omission rates in detection, tracking, and classification. Since the physical terrain is not assumed to be constrained, the network must deal with breaches anywhere along the long perimeter. Response must be in near real-time, within a few seconds of intruder events, over the mission lifetime, even if intrusions are random, rare or ephemeral. Quality of service from the application is required even if some nodes in a region are misplaced or their components fail, during deployment or operation.
3. *Low human effort over the duration of deployment*. This applies to all surveillance phases, including the placement of the nodes as well as in the operation, monitoring, maintenance, and reconfiguration of the network.

The ExScal project built upon an earlier field demonstration, *A Line in the Sand* [1], where we manually positioned one laptop base station and 90 Mica2 "motes" (with magnetometer and micropower impulse radar sensors) over a 25 m by 10 m grassy area. A new architecture needed to be evolved to scale the system from 90 to 10,000 motes and to scale perimeter area by 1,000 to 10,000 times, while still meeting complex application requirements. The following architecture design principles were adopted.

1. To contain cost, we designed nodes that have sensing and communication ranges substantially larger than Mica2 motes; Sect. 8.2 describes these nodes: the XSM (for Extreme Scale Mote) sensor nodes and the XSS (for Extreme Scale Stargate) backbone communication nodes.
   The nodes were arranged in a planned topology, specifically a regular, hierarchical structure, to efficiently cover the region. Tier 1 of the hierarchy consisted of a grid of XSMs, Tier 2 consists of grid of XSSs, and Tier 3 consisted of one master operator node. Section 8.3 describes the topology in detail. Application services exploit knowledge of this topology for efficiently utilizing system resources.

2. To meet the desired quality of detection, we decomposed the ExScal application into multiple components that execute in different phases of operation. Section 8.4 describes these components. Decomposition simplifies the design,

allows us to configure and manage each component separately (at run time if need be), and reduces operational resource requirements. Importantly, it frees us from using common services for all components, instead we can use different services optimized for components needs. However, decomposition introduces a new challenge of interfacing components and also new failure scenarios.

3. For cost-effective human manageability, ExScal operated by command and control: Tier 3 initiates, monitors, and regulates the operations of all XSSs; in turn, each XSS likewise monitors and manages a section of (normally 20–50) XSMs. The operator maintained/reconfigured ExScal effectively based on the feedback obtained from the network. Section 8.5 describes the management aspects of the application. Autonomous functions, specifically, configurable recoverability for components, tolerance to several classes of faults (often by self-stabilization), and adaptation to certain classes of variable, non-uniform environments all support containment of human effort.

Thorough system testing was important due to the large number of system components. Each component needed to be tested both individually and when integrated with other components. Section 8.6 explains the role of simulation and emulation (testbed), and outdoor testing to gain confidence in the system. Section 8.7 describes some of the experimental results obtained.

Data and other literature on the project is available on the ExScal website, http://www.cse.ohio-state.edu/∼exscal.

8.1.1 ExScal Status and Some Project Facts

We started work on ExScal in September 2003 with a mandate to cover a 10 km by 1 km perimeter with 10,000 sensor nodes. To this end, we designed the two types of nodes and had 10,000 XSMs and 300 XSSs manufactured (these nodes are now commercially available from Crossbow). The footprint of the code we designed for ExScal was 200 kB for an XSM and 2 MB for an XSS. We designed ExScal scenarios for node configuration at the factory, for field marking, for node deployment at site, for network configuration in the field, for ExScal operation, and for network teardown.

We executed the afore-mentioned scenarios over a 2 week period in December 2004, during which we collected data on field marking accuracy, deployment yield, localization accuracy, sensing performance and variability, environment data (especially wind data collected via microphones), communications and network management performance at each tier, and intruder traces. Our experiments were conducted with 1,000+ XSMs and 200+ XSSs deployed over a 1.3 km by 200 m opening in a forest in Florida, USA. The scale of the final experiment was mandated by a change in the security policies of our sponsor, as a result of which, ExScal was transitioned into a new phase under a classified setting.

At the time of writing this chapter (2009), 4 years have passed since the deployment. Three major papers have been written: one is about the end-to-end system,

the second looks at faults and their effect on the performance of the system [2], and the third focuses on distributed sensing techniques [3]. The current book chapter is heavily based on the first paper. The authors have added sections on the system's software architecture of the system, and use of testbeds in debugging.

8.2 The Hardware Platform

XSM We designed the eXtreme Scale Mote (XSM) [4] for ExScal, especially to obtain (1) increased sensing range with respect to persons and vehicles as well as increased communication range (as compared with extant motes), and (2) long-lived, retaskable operation for timely detection of rare, random, and ephemeral events. Figure 8.1 shows a picture of the XSM enclosure and internals.

The XSM infrared and acoustic sensors were designed for low-power continuous operation and include asynchronous processor wakeup circuitry. Based on signal processing techniques further described in Sect. 8.4, the sensing ranges achieved by the detectors are as stated in Table 8.1.

The reliable communication range between on-the-ground XSMs typically exceeds 30 m in outdoor settings (although there is variability in this range based on ground conditions, humidity, etc.). The XSM lifetime approaches 1,000 h of continuous operation on two AA alkaline batteries. Recoverable retasking is addressed by

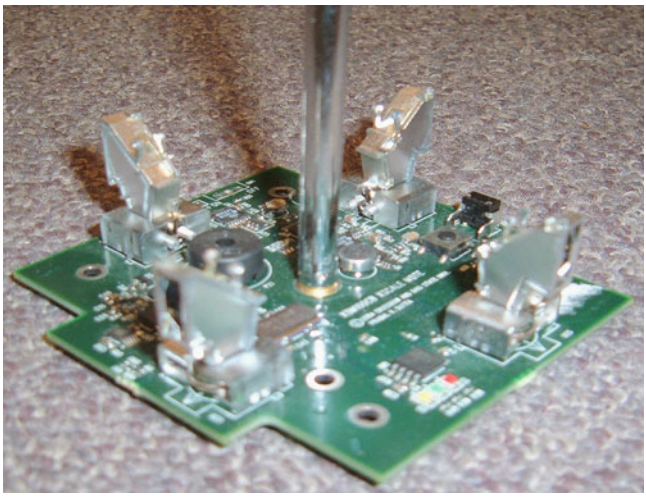


Fig. 8.1 The eXtreme Scale Mote. The XSM circuit board has a $3'' \times 3''$ footprint and the enclosure's dimensions are $3.5'' \times 3.5'' \times 2.5''$. The one-touch input (on/off switch) and one-listen output (buzzer) are mounted on the base next to the batteries. The XSM platform integrates an Atmel ATmega128 L microcontroller, a Chipcon CC1000 radio operating at 433 MHz, a 4 Mbit serial flash memory, quad infrared, dual-axis magnetic, acoustic sensors, and weatherproof packaging. XSMs are commercially available under the tradename of MSP410CA Mote Security Package

Table 8.1 Sensing ranges for intruders. SUV abbreviates Sport Utility Vehicle and ATV abbreviates All Terrain Vehicle. Original Table used with permission of IEEE [23]

Sensor	Intruder	Sensing range (m)
Magnetometer	SUV	7
PIR	SUV	30
	Person	12
Acoustics	ATV	5

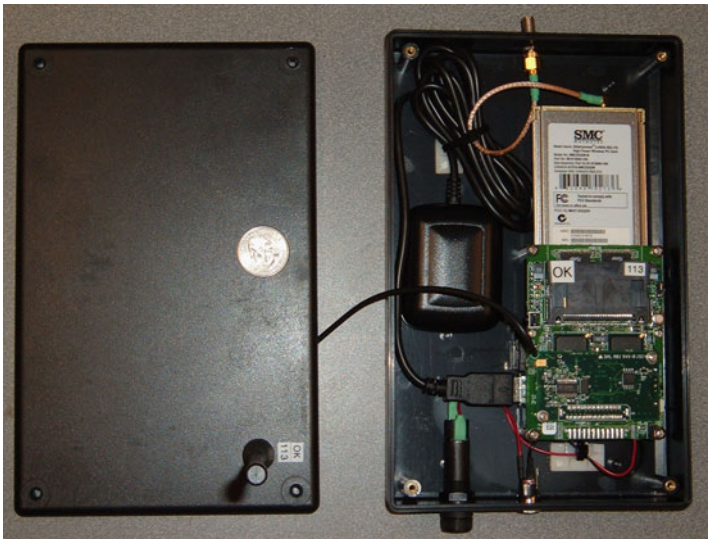


Fig. 8.2 The eXtreme Scale Stargate. Each XSS has an Intel 400 MHz XScale R processor (PXA255) with 64 MB SDRAM, 32 MB FLASH, type II PCMCIA slot, USB port, and 51-pin mote connector; packaging is watertight. Original Figure used with permission of IEEE [23]

using a grenade timer [4, 22] that periodically forces a system reset. We provide the rationale behind the use of grenade timer in Sect. 8.4.1.

XSS As its name suggests, the eXtreme Scale Stargate [5] includes a Linux-based Stargate computer [6]. It also includes a GPS unit, which connects via the Stargate daughter card USB port; the sensor we chose has a positioning error of up to 10 m. We used regression techniques and geometric algorithms to rectify the errors. The XSS node is pictured in Fig. 8.2.

XSSs communicate with each other via their 2532 W-B high power IEEE 802.11b card, which is connected to a 9 dBi antenna. The antenna is 1.82 m long. With this setup, we observed over 700 m reliable communications in the field at full power. XSSs communicate with nearby XSMs via the Chipcon CC1000 radio in a Mica2 that is connected to Stargate via a 51-pin connector.

The current required for various stargate operations is significant: 70 mA for processor operation, 90 mA for GPS operation, a total of 440 mA when in 802.11 b receive mode and 810 mA in the 802.11 b send mode. Given this requirement and the Stargate limitations for wake-up-on-radio and fast processor duty cycling, we

chose to power each XSS by a lead-acid battery that provides 6 V DC with total current draw of 105 Ah, and focused our attention on the energy efficiency of the XSS protocols/programs.

8.3 Topology, Coverage, and Deployment

Topology As discussed in Sect. 8.1, *ExScal* used a *planned* topology for node placement so as to efficiently cover the protected region. Note that since 7 m is the lowest sensing range for an intruder (see Table 8.1 – magnetometer-based detection of SUVs) it is straightforward to see that no deployment of 10,000 nodes over a 10 km by 500 m area or of 1,000 nodes over a 1.3 km by 200 m area can cover all points in the region.

Fortunately, *ExScal* application scenarios did not require sensor coverage at all points within the interior of the region; barrier coverage [7] is sufficient. That is, if intruders were detected multiple times soon after they enter the region – they could thus be classified and finely-tracked initially. This region is away from the asset. The intruders remained undetected only in bounded regions in the interior (near the asset) – they could thus still be coarsely-tracked within the interior. We therefore deployed sensors more densely at the boundary of the region than in its interior, where we assumed the high-value asset being secured lies (see Fig. 8.3). The "thick" line of sensors at the outer boundary of the region consisted of five rows of XSMs. For ease of deployment, alternate rows were staggered by half the spacing between consecutive sensors in a row so as to provide close to optimal coverage.

Coverage Considering Table 8.1 and Fig. 8.3, if any intruder crossed the thick line, it is detected by least five sensors. More specifically, an Sports Utility Vehicle (SUV) is detected by at least five magnetometer sensors and 30 PIR sensors; a person is detected by at least 10 PIR sensors; and an All Terrain Vehicle (ATV) is detected by at least 55 sensors that included magnetometer and PIR. The net result is that, even if some intruder detection messages are lost due to network unreliability, classification and fine-grain tracking of intruders is still possible when they entered the region through the thick line.

The interior of the region consisted of a grid of "thin" lines, each containing a single row of sensors. These enabled bounded-uncertainty tracking of intruders as they crossed from one thin line region to another.

In terms of reliable communication connectivity, this node placement ensured that each XSM could reliably communicate with 10–32 other XSMS in the thick line and 3–6 other XSMs in the thin lines. That said, since multi-hop reliability in the bandwidth-constrained Tier 1 network could be insufficient after 5–6 hops, we partitioned the Tier 1 topology into sections of 20–50 XSMs each and emplaced an XSS in each section to serve as a communication bridge between its XSMs and the Tier 3 base station. XSSs were thus spaced 90 m apart and each XSM could reach two XSSs within 5–6 Tier 1 hops. A total of 45 XSSs were needed for the Tier 2 communication backbone network.

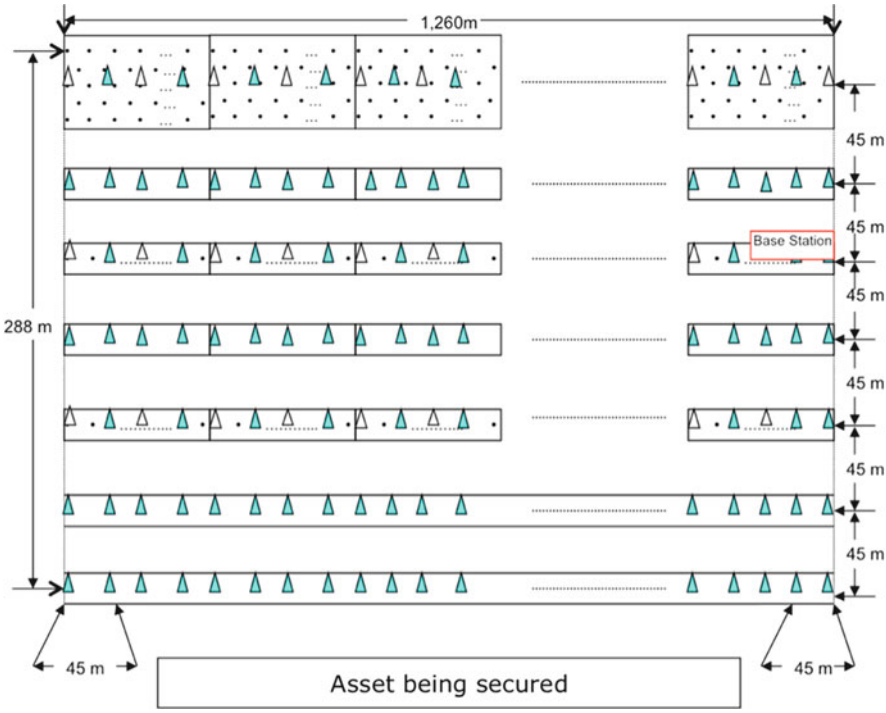


Fig. 8.3 *ExScal* Topology. Dots represent XSMs and triangles represent XSSs. Only the XSSs represented with empty triangles are used as communication bridges between the XSM network and the base station. Original Figure used with permission of IEEE [23]

Additional XSSs were manually positioned for executing experiments to validate that the XSS network scales to meet the original objective of the 10,000 node *ExScal* network. The full peer-to-peer ad hoc 802.11 b Tier 2 network consisted of the 203 XSS arranged in a 7×29 grid. After conducting radio connectivity tests, we found out that at full power, each XSS in this grid communicated reliably with 109–202 other XSSs; at low power, the numbers were substantially lower but were still sufficient to tolerate the failure of several XSS without partitioning the Tier 2 network.

Deployment For field ground truth measurement and node emplacement, after considering several alternatives involving laser range finders, walking meters, and ropes for triangulation, we settled on using commonplace surveying equipment, which not only provided us with submeter accuracy (within 0.2 m and with high likelihood within 0.1 m), but also saved both time and effort. We used a Leica Total Station 307, 3 Leica Reflectors mounted on prism poles, and a 45 m nylon rope with 9 m markings on it. A surveying expert helped with the marking process; see [8] for details.

The submeter accuracy we achieved in marking out the locations was not necessitated by the ExScal application, since our planned topology satisfied its coverage and connectivity requirements even if the actual separation between consecutive XSMs on the ground was off by 5 m (i.e., 14 m instead of the ideal 9 m). However, the submeter accuracy gave us finer ground truth so as to measure the accuracy of the localization process and the effect of less accurate placements on the quality of the results.

We completed marking all locations (983 XSMs and 203 XSSs) in 27 working hours with a team of eight people. Laying out the equipment took longer: about 24 working hours with 14 people. The grid topology not only enabled us to provide coverage in a cost efficient manner but it also enabled us to design energy efficient data transport protocols, such as logical grid routing and Spinkler, described in detail in Sect. 8.4.3.

8.4 The Software Architecture

As discussed in Sect. 8.1, *ExScal* met its quality of detection requirement by being decomposed into several components. Specifically, these are: a Trusted Base component, a Deployment component, a Localization component, and a Perimeter Security component.

8.4.1 The Trusted Base and Its Deployment Components

At Tier 1, each XSM had a trusted base program, *Nucleus*. Nucleus included a bootloader that executed every time a node starts and that offered an API to the running program by which to switch to another program binary (including Nucleus itself); the switch involved rebooting the XSM.

For convenience, we bundled with Nucleus a deployer response subcomponent as well as power management features. During deployment, we desired basic confirmation that each node was awake and functioning as we placed it on the ground. To this end, Nucleus exercised the sounder each time it was booted, and sent out multiple radio messages containing the node's unique identifier and network address. This served as feedback to the deployer as to when to move on and begin installation of the next XSM. Since the network deployment could take several hours, to avoid battery depletion in the interim, the power-saving sleep system in Nucleus was immediately enabled after the startup confirmation. This system used low-power listening [9], making it possible for the nodes to wake up only to receive radio messages and then go immediately to sleep.

Nucleus also included a subcomponent that provided node testing and network management functionality: specifically, it included Deluge [10] for dissemination of new programs and SNMS [11] for sending commands and collection of health/status information to all XSMs in a section via their Tier 1 network.

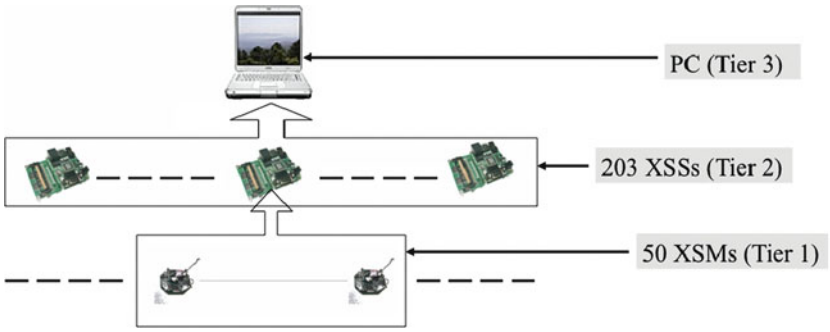


Fig. 8.4 Hierarchical network topology

Nucleus' dissemination component enabled sending commands to all XSMs in a section, e.g., to wakeup the section from sleep mode. A section is defined as a group of 50 XSMs, which were headed by one XSM as shown in Fig. 8.4. The wakeup command was sent as a normal message, but with a long preamble that would trigger a sleeping node to wake up. After awakening, the dissemination component in Nucleus would periodically retransmit the long-preamble wakeup message, to help wakeup the rest of the section. Nucleus' other network-based commands implemented sleep, reboot, and switch.

Once a Tier 1 section was woken up, its health was determined by querying over the node identification and status over the multihop network. Queries were injected by the XSSs associated with that section into the Tier 1 network using the dissemination component described above; this formed a routing tree rooted at the XSS; a separate data collection component then returned the results to the XSS.

Finally, Nucleus used the grenade timer [4] to provide protection against Byzantine failure in our task-specific components. A *Byzantine failure* is a fault, in which a component of some system not only behaves erroneously, but also fails to behave consistently when interacting with multiple other components. After being rebooted by an expiring grenade timer, the Nucleus bootloader always fell back to executing Nucleus. This ensured that after a failure, the nodes were left in a known state from which we could recover the system.

The binary bundle of Trusted Base and Deployment components was mutually exclusive to that of Location and Perimeter security component. Nucleus is the Trusted Base and Deluge and SNMS are its Deployment components. In other words, an XSM is either one of them at a time and not both. Since the bundle of Trusted Base and Deployment components contained only three components, which were Nucleus, Deluge, and SNMS, we do not present its architecture diagram.

8.4.2 The Location Component

This component assigned the grid position labels to Tier 1 and Tier 2 nodes. The grid locations were used by the Tier 1 routing protocol. Our design separated the

concern of manual deployment of nodes to grid locations from the concern of how nodes acquire grid positions. We used GPS to record spatial coordinates: at Tier 2 this was done by attached hardware, and Tier 2 copied the GPS data to Tier 3. For Tier 1 we did the same, however to economize on equipment, we manually recorded GPS data for each XSM and then uploaded results to Tier 3. Algorithms at Tier 3 then computed grid information to associate with each node identifier of the other tiers, and network protocols at these tiers delivered the output to each node. The grid information consists of association of the identifier to its grid coordinates.

Snap to Grid Imprecise physical deployment and inaccurate or missing GPS data complicated the transformation from GPS coordinates to grid positions. Errors of four or more meters could be expected for GPS readings at Tier 1, which made resolution of a grid with 9 m spacing and with (say) 1 m deployment error, nontrivial. Thus regression techniques and geometric algorithms needed to be developed. The resulting algorithm was called *snap to grid* (or simply, Snap).

The inputs to Snap were the ideal template of GPS coordinates for the grid, with a grid label for each node, and the set of recorded GPS data and corresponding node ID. The output of Snap was a list of grid labels and the associated node ID, which was subsequently used to determine routing information, later disseminated via Tier 2 and Tier 1 protocols. Briefly, Snap works by iteratively rotating and translating a bounding box of the input set of points until the box roughly aligns with the template; then by using linear regression to obtain a refined estimate of the slope of (say) one column of the input set, a further rotation improved the alignment. Finally, translation was improved by minimizing the sum of distances between input point and its nearest template point(s), again calculated iteratively using bounding boxes. The same technique also yielded the required matching between input point and template grid position.

Figure 8.5 shows typical input and output for a small portion of the grid. The plot on the left uses latitude/longitude coordinates, whereas the plot on the right uses synthetic grid-coordinates of Snap. This example has several points missing as well as errors in GPS position in the input.

The primary subcomponent of the Location component was Snap. It is executed at Tier 3. The dissemination of Snap's output was done by the Tier 1 and Tier 2 communication protocols, which are explained in Sect. 8.4.3 Since Snaps is a single program, we do not present the software architecture diagram of the Location component. The architecture of the communication protocols is explained in Sect. 8.4.3.

8.4.3 Perimeter Security Component, OpAp

OpAp stands for Operation Application. This is the application that obtained data from sensors, detected whether there was an intrusion, and, if there was an intrusion classified the intruder, and tracked it. Figure 8.6 describes the software architecture of the *OpAp* component at Tier 1. At Tier 1, *OpAp* included the magnetometer,

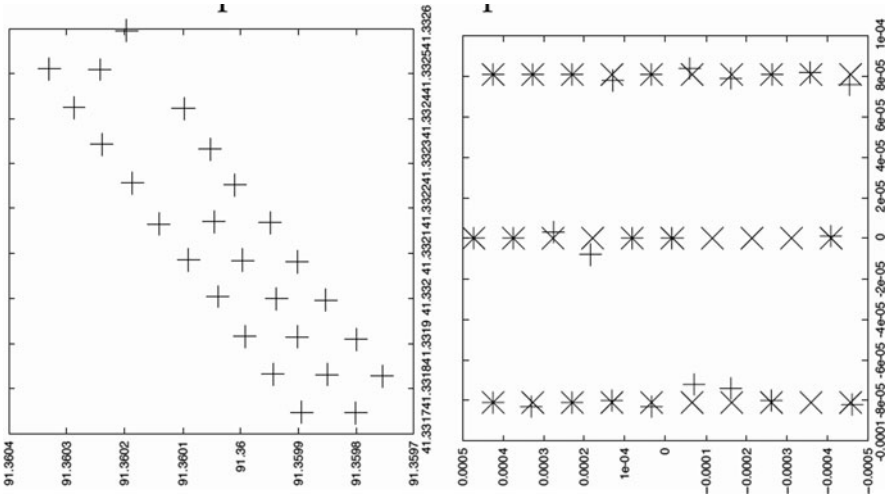


Fig. 8.5 *Left*: Input to Snap program. X and Y axes are latitude and longitude. Pluses represent reported GPS locations of the nodes. Missing locations are shown as blanks. *Right*: Output of Snap program. X and Y axes are synthetic grid-coordinates of Snap. The matching of input with template points are represented by crosses. Note that the missing locations in the middle row are filled in by crosses. Original Figure used with permission of IEEE [23]

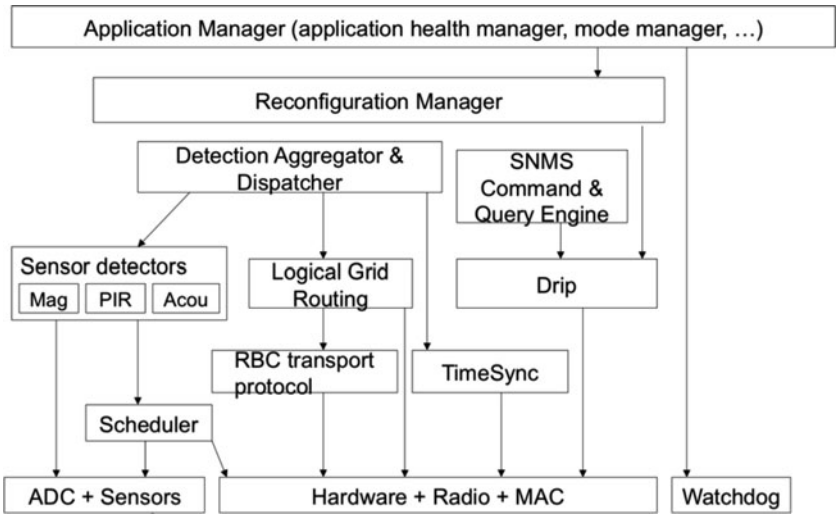


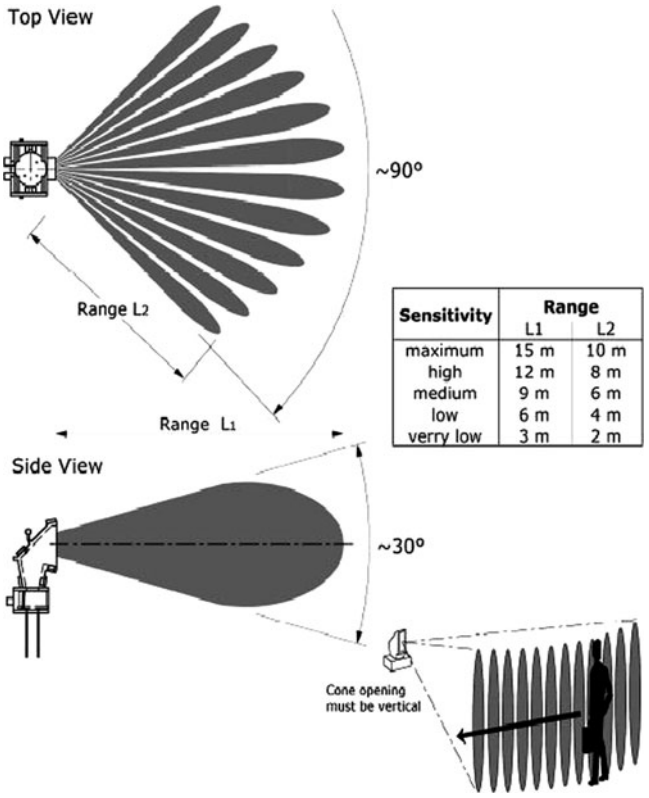
Fig. 8.6 Software architecture diagram of *OpAp*. The arrows indicate hierarchy. The software subcomponent at the root of the arrow is at the higher level in the hierarchy

infrared, and acoustic sensor chains that detected intruders; only one of these chains is described below, as an example. A sensor chain was a sequence of operations starting from sampling the ADC, filtering the noise, to raising a detection

event. Detection events, along with their timestamp (Tier 1 *OpAp* executed a time synchronizationprotocol that synchronized mote clocks to within 1 ms of each other) and grid location, were sent via the Tier 1 and then the Tier 2 network to the Tier 3 node, which classified and tracked intruders. We used Tsync time synchronization protocol where all the nodes synchronized their time to that of a root node. The code is available under contrib/minitasks/02/osu/timesync repository in TinyOS 1.x.

Tier 1 Motion Detection using Passive Infrared Sensors PIR sensors are commonly used for motion detection in automatic light switch and home security products; they are made of a crystalline material whose surface charge varies in response to the received infrared radiation emitted from warm objects such as the human body (Fig. 8.7).

For most indoor applications a simple analog detector circuit – typically a multistage amplifier and a two-level comparator – has satisfactory performance. However,



Sensitivity	Range	
	L1	L2
maximum	15 m	10 m
high	12 m	8 m
medium	9 m	6 m
low	6 m	4 m
verry low	3 m	2 m

Fig. 8.7 PIR sensor from Kube Electronics with integrated cone optics and 90 degree field of view. Four such sensors provide 360 degree coverage for the XSM. Original Figure used with permission of IEEE [23]

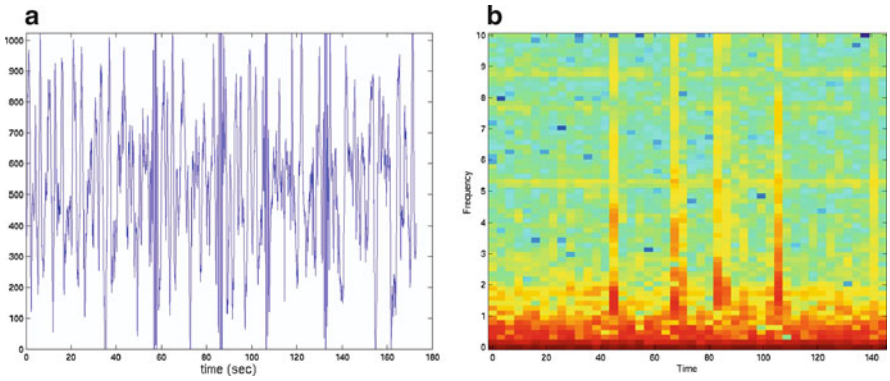


Fig. 8.8 *Raw PIR sensor values for an SUV (35 km/h).* (**a**) Time domain (X-axis represent raw sensor readings). (**b**) Frequency domain (X-axis represent frequency in hertz and Y axis represents time in seconds) The darker the color more the energy. Original figure used with permission of IEEE [23]

comparator-based detectors produce frequent false alarms in outdoor environments due to heat drifts and sunlight. Figure 8.8 shows raw signal values obtained from the PIR sensor for an SUV traveling four times across the field of view of the PIR sensor at 35 km/h. We observed that simple threshold based detectors caused false alarms even for very high threshold values. A frequency domain analysis of the same data revealed that target signature and the background variation occurred in non-overlapping bands, enabling reliable detection. A human walking at a moderate speed of 5 m/s occupied the 0.5–1.5 Hz band, a vehicle traveling 35 km/h signature was within the 1–5 Hz band, whereas background variations were confined to the 0–0.5 Hz band.

To increase detector robustness, we used a polyethylene film in the PIR windows, to reduce the effect of sunlight, and a digital bandpass filter to process raw sensor values to isolate the target energy from the slower background variations due to heat drifts. Raw signal values were first passed through a band pass filter with a pass band of 0.4–2 Hz. The energy of the filter output over a sliding time window was calculated by low pass filtering the instantaneous power. A low pass filter with a low cutoff frequency of 0.3 Hz was used to smooth detection events to prevent the PIR activity event from being broken into multiple events. Figure 8.9 depicts the output signal of the detector. We observed a 12 m reliable range for humans walking at a speed of 3 km/h and a 25 m reliable range for a midsize SUV traveling at 35 km/h.

Tier 1 communications *OpAp*'s performance requirements demanded that XSM detection packets be transported reliably and in real-time to the corresponding XSS. Nevertheless, we found that with the MintRoute's [21] distance-vector routing and queue management in TinyOS, only 33.7% of packets from XSMs are delivered to the XSSs on average. The causes for such low packet delivery rate included unreliable wireless links and high degree of channel contention in the presence of bursty

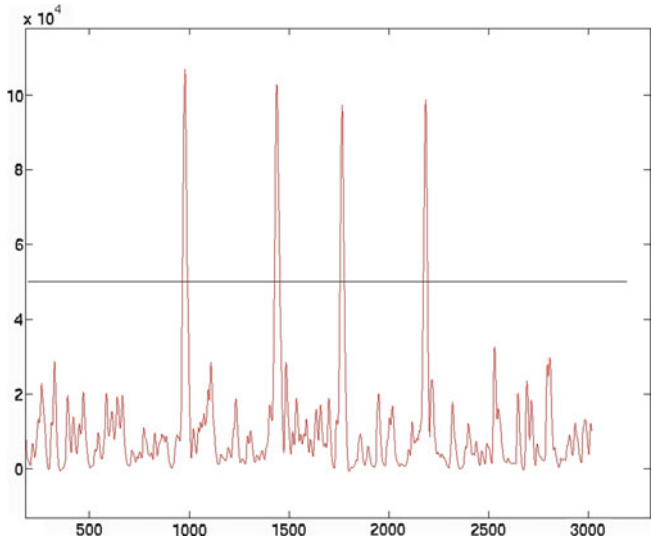


Fig. 8.9 Output signal of PIR sensor signal chain. X-axis represents time in seconds and Y-axis represents computed energy by the signal chain. The peaks above the threshold of 4.5×10^4 represent activity. Original figure used with permission of IEEE [23]

convergecast, where a huge burst of data packets needed to be transported reliably, in real-time, and simultaneously. The convergecast is a pattern of data traffic, where the destination of the routes is a unique node. In our case, the destination was the Tier-1 base station for each section of 50 XSMs.

To address the high packet loss rate, we used the Logical Grid Routing (LGR) protocol [12]. The LGR mapped the Tier 1 network onto a logical grid and only used links that are reliable in the presence of bursty convergecast. In LGR, each node locally determined, via information on its grid location, its potential parents and then distributed traffic uniformly across all the potential parents. LGR was fault-tolerant and recovered from faults quickly. With LGR, up to 81% of data packets were delivered from XSMs to the corresponding XSSs.

To further improve reliability, we used the Reliable Bursty Convergecast (RBC) transport protocol [13]. The RBC dealt with the loss of per-hop acknowledgments and retransmission-incurred contention as follows: to improve channel utilization and to reduce ack-loss, it used a windowless block acknowledgment scheme that guarantees continuous packet forwarding and replicates the acknowledgment for a packet; and to alleviate retransmission-incurred channel contention, it used differentiated contention control. It also had mechanisms to handle varying ack-delay and to reduce delay in timer-based retransmissions. Testbed experiments showed that LGR with RBC delivers on average 99% of the data packets from XSMs to the corresponding XSSs in real-time, sufficing for the requirements of reliability and timeliness in Tier 1 communications. LGR and RBC are suitable for any other perimeter security applications, which have grid topology.

Tier 2 communications The Tier 2 network was configured in IEEE 802.11 peer-to-peer ad hoc mode. Its architecture was composed of three reliable, power efficient transport services, namely Initd, Learn On the Fly (LOF) [14], and Sprinkler [15].

Initd–An Unstructured Broadcast Service: Initd was used to initialize the XSS network. Initialization consisted of the Tier 3 base station contacting each XSS and collecting their GPS locations. Initd used controlled diffusion to energy efficiently construct a one-shot tree rooted at the base station; the tree was used in the GPS location collection.

LOF–A Structured Convergecast Service: LOF was used to transport a message from any XSS to the base station. LOF exploited geographic location information to construct a shortest path tree rooted at the base station. The metric to construct the tree was link quality in terms of delay and the geographic distance advanced towards the root. To save power and to improve estimation fidelity, LOF used data traffic instead of beacon messages to estimate the link quality. Initially when the nodes boot up, there is a short period when the nodes send beacons. However, later link quality is solely estimated using data traffic. Given that the link quality changes over time, LOF probabilistically switches its forwarders to sample other routes.

Sprinkler–A Structured Broadcast Service: Sprinkler was used to disseminate bulk data (up to say 200 kB) to all XSSs. It exploited geographic information to construct a connected dominating set (CDS) and a transmission schedule for the XSSs in the CDS. The cardinality of the CDS was minimized to optimize the number of transmissions. The CDS XSSs used broadcasts to transmit messages; their schedule avoided the hidden terminal effect to ensure reliability and timeliness, without significant message retransmission.

One challenge in composing ExScal was interfacing of multiple software and hardware components. One example, where interfacing of two components failed is a connection between Initd and Sprinkler services at Tier 2. At each XSS node, Initd service gave the location of the node to Sprinkler service. In some of the cases, GPS on a node was unable to get a fix and hence the node was not located. In that case, Initd gave (0,0) as a grid-location to Sprinkler. Note that (0,0) was a valid grid-location for Sprinkler. When Sprinkler computed a CDS, it mistakenly placed unlocated nodes at (0,0) location. This led to an incorrect CDS, resulting in a failure while disseminating data. Therefore, while interfacing two components, it is important to verify the semantics of the parameters passed between those components.

Tier 3 *OpAp* logic To classify an intruder as person, SUV or ATV, *OpAp* measured for each sensor modality the *influence field* of the intruder on that sensor modality [1, 3], i.e., the area surrounding the intruder within which it was detectable by a sensor of that modality. Thus, for instance, with respect to an ATV, an SUV had a relatively larger magnetic influence field, a relatively smaller acoustic influence field, and a comparable PIR influence field.

To measure influence fields, *OpAp* aggregated detections over a time interval (typically 500 ms), since multiple XSMs may detect the intruder at a given location at different times, due to differences in hardware and synchronization. It clustered

spatially collocated detections of the same sensor modality in the interval, calculated cluster sizes, and classified accordingly. The component also maintained a history of decisions made in the recent past intervals, to increase or decrease the confidence of classification. Processing of intervals lagged real-time to accommodate network jitter, yet end-to-end classification of intruders was within 5 s. The system was able to classify and track multiple objects moving concurrently through the network as long as a minimum distance threshold separated them.

To track an intruder, *OpAp* estimated the centroid of a convex hull enveloping all of the nodes detecting that intruder. Depending on the type of intruder and its current estimated position, the tracking module also computed an expected region for the intruder location in the next time interval, based on the velocity of the type of the intruder. It then correlated the tracked objects from successive windows in order to construct a continuous track per intruder for the entire time it spends in the network. If the estimated location of an intruder did not lie in the expected regions of any of the currently tracked intruders, a new intruder was detected and a new track was created. When no new information was associated with a particular target for a certain interval of time, the track was removed.

OpAp also maintained health information of the XSMs by keeping track of frequent outlier nodes and nodes that did not respond; it used this information to refine the classification of intruders.

8.5 Management

ExScal used two distinct approaches for management. The first was a multi-tier command-and-control framework that allowed its operator to perform management operations from the Tier 3 base station: operator commands were disseminated using *Sprinkler* over the Tier 2 network to a management daemon running on each XSS. Based on the command type and its scoping rules, XSSs then either locally executed the command or invoked a Tier 1 management process, such as Deluge, SNMS, or *OpAp*-specific "dynamic reconfiguration". Examples of SNMS commands are sleep and wake-up. New configurations ranged from simple parameter updates to activating or deactivating modules such as sensor chains; an optimized version of the reconfiguration service at Tier 1 piggybacked configuration updates on routing heartbeat messages, thereby conserving network bandwidth. The results of commands, which were likewise obtained locally or aggregated from the XSM network, were then communicated by each XSS to the Tier 3 base station using *LOF*.

The second approach was one of local, autonomous management wherein each node used several local managers to detect and correct different types of low-level faults and maintained node consistency. For instance, each XSM used a "mode manager" to handle transitions between different component phases of *ExScal* and used local decisions to resolve any detected inconsistencies with its neighbors' modes based on policies specified by the network operator. Each XSM also used multiple component-level managers to monitor health predicates. For instance, the

XSM *OpAp* module that aggregated detection events from the sensor chains for dispatch to the base station used a "rate monitor" to count number of events. If the number of event per unit time crossed a pre-defined threshold, the module stopped sending messages. This prevented highly sensitive XSMs or XSMs in noisy environments from flooding the network with false detections. Finally, each XSM had a manager, which used outputs from the component health managers and from a watchdog timer based monitor to detect if the component was in a persistent failure state where it would repeatedly deadlock or its components would repeatedly misbehave. In such cases, the manager could choose to either run the component in a *safe* configuration by disabling certain modules, or simply return control back to Nucleus whereby the faulty component could be replaced. Again, such decisions were based on policies pre-specified by the operator. The difference between a watchdog timer and a grenade timer [22] is that the former can be delayed but the latter cannot be. Given the large number of components, which were implemented by a large number of programmers, the operator of the system did not have complete knowledge of the working of all the components. Hence, to recover from an extreme failure where a component was stuck, we used a watchdog timer. The grenade timer was used for the catastrophic failure where a component did reset a watchdog timer regularly but looped continuously.

In the ExScal network, we used synchronous duty-cycling, which means all the nodes wake up and sleep synchronously using a clock that is synchronized with a single global clock. The wake and sleep scheduling was done by the operator. A drawback of the synchronous approach is that the nodes have to exchange time synchronization packets, which consumes energy. A team at the Indian Institute of Science (IISc) is working on a project with the same application as that of ExScal. The project is using Telosb motes with a different version of PIR sensor. The IISc's project is using asynchronous duty-cycling. In asynchronous duty-cycling [16], the nodes do not all wake up and sleep at the same time. Although the asynchronous approach saves on time synchronization packets, the nodes may have to wake up for a larger amount of time while forwarding intrusion detection packets to the base station. Thereby, they may be spending more energy than those using a synchronous approach. In ExScal, our duty-cycling policy was based on a command-and-control framework.

8.6 Testing Prior To, and During the Final Deployment

Testing was done in simulation, using an indoor testbed, and with a scaled-down version outdoor. For simulations, we used TOSSIM [19] for Tier 1 and Emstar [20] for Tier 2 and 3. The most significant advantage behind using TOSSIM and Emstar is that the simulation code is same as that of the actual deployment code. This reduces time in writing code and also improves fidelity of the simulations. However, not all errors are caught during simulations since the architecture of the machine running simulated code is x86 hence different from that of XSM's Atmega and XSS's ARM

architectures. Bugs such as memory leaks or segmentation faults may not be caught through simulation. Additional testing with a testbed is thus needed to help catch these sorts of problems. Kansei was developed for this purpose.

Kansei is a testbed of 210 XSMs hooked individually onto 210 XSSs. The stargates are connected using both wired and wireless Ethernet. The wired channel is used to collect measurements of performance of wireless protocols at a central server. Kansei provided a testbed infrastructure to conduct experiments with 802.11 b networking and XSMs. It was conceived to test the middleware services for the Tier-2 network. Kansei exports a web interface on which experiments can be scheduled and the results retrieved.

The XSSs are arranged in a 15 × 14 grid, with an inter-node separation of 3 feet in X and Y axes, inside the Kansei testbed [17]. Each XSS is equipped with a SMC2532W-B high power IEEE 802.11 b card, which is connected to a 3 dBi antenna of length 1.82 m via a fixed attenuator of −20 dB [17] as shown in Fig. 8.10a. The nodes are raised from the ground at 4 feet. Each stargate is connected to the server via wired Ethernet. All the control and data traffic is communicated via wired Ethernet (green Ethernet cables in Fig. 8.10b) to avoid interference with experimental traffic. The network is configured in IEEE 802.11 b ad hoc mode. The frequency is set to 2.462 GHz (channel number 11). The environment contains another IEEE 802.11 network in an infrastructure mode, which is operating at 2.437 GHz (channel number 6). Kansei does not operate that network. According to IEEE 802.11 standard, the 2.462 GHz and 2.437 GHz frequencies (channel separation of 5) are non-interfering. This was corroborated experimentally by Robinson et al. [18]. The sensitivity threshold of the card was set to maximum. In other words, the signal at the receiver is not attenuated via software control.

Developers for Tier 2 used Kansei to debug the code. In the actual deployment, catching segmentation faults due to invalid memory access are hard because there is no console for the nodes. In Kansei, each node is connected to the wired Ethernet, which in turn is connected to a PC with a console. Thus, debugging the segmentation faults becomes easier. Figure 8.11 shows a screenshot of the console, running Emview-based visualization for Sprinkler. In case of a segmentation fault

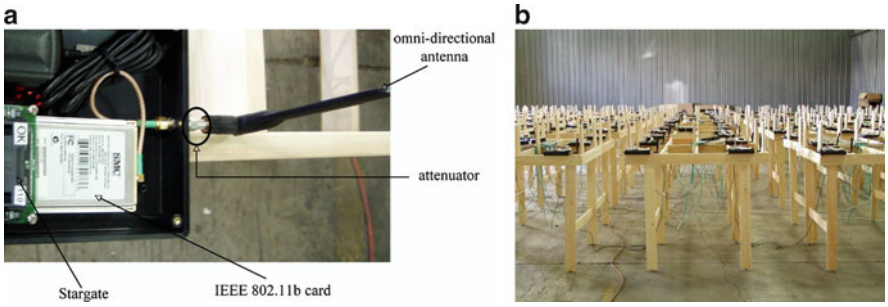


Fig. 8.10 The Kansei testbed. (**a**) Anatomy of a wireless node. (**b**) Arrangement of nodes in Kansei testbed

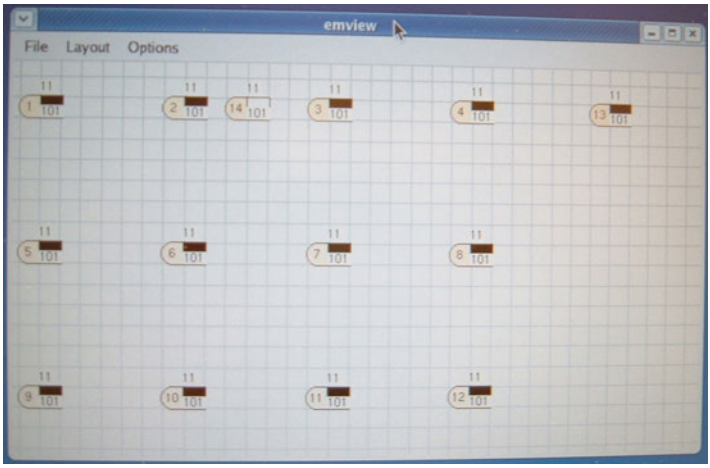


Fig. 8.11 Emview-based visualization of the network topology and progress of Sprinkler's data dissemination in Kansei. Each object consisting of a half-circle and a rectangle represent a node. The number in the half-circle is the ID of the node. The number in the rectangle is the number of packets received by the node. In this figure, all the nodes have received 101 packets. The number, above the object, represents the ID of the reprogramming session. In the figure, the session ID is 11

at a node, the node was highlighted on the screen. The visualization also provided information about which service experienced the fault and in which function call.

Upon design, the developers tested their code multiple times in the simulation and testbed environments before the final deployment. Simulation was used to test the changes quickly and the testbed was used to test more rigorously. In addition to these two testing environments, the developers performed testing in an outdoor environment. The scale of the outdoor deployment was smaller than that of the final fielded network. It consisted of about 50 nodes for Tier 1 and 10 nodes for Tier 2. However, the number of iterations for the outdoor deployments was far less than that of simulations and testbed. The primary goal of the outdoor testing was to understand the behavior of the system in terms of sensing accuracy and message reliability, when deployed in a real environment.

8.7 Results and Conclusions

We measured *ExScal* network yield for realistic intruder scenarios. The end-to-end routing yield was 85.27% (with Tier 1 at 86.72% and Tier 2 at 98.32%). We found the deployment faults (at 5.37%), localization faults (at 11.4%), and reprogramming faults (at 5.5%) to all be uniformly distributed across the region. In most cases, protocol designs were primarily responsible for tolerating the faults; policy-based managers dealt with nodes that experienced "lagger" and Byzantine contract-violation faults. The number of Byzantine sensors in the networks was 1%.

Reasons for sensors to become Byzantine were waving tall grass in the wind, extreme heat, or rain. If the battery voltage of the nodes fell below a critical value, the nodes observed arbitrary sensing values. We found the overall Tier1–Tier2 networked sensors reliability to be 73%.

In the absence of prior estimates of the faults, we conservatively selected network density to be twice the minimum required. Given the significant coverage and communication redundancy in our planned topology (cf. Sect. 8.3), the spatial uniformity of faults, and our architectural design principles (cf. Sect. 8.1), we found that with this yield the *ExScal* application was able to meet its requirements at the 1,000+ node scale, as was validated in the presence of persons, ATVs and SUVs traversing through its thick line and thin lines, and in management operation tests to reconfigure, change parameters, upload programs, and query health. We also studied the scaling of density, size, and path length in the Tier 2 network. Given the predictable patterns we observed, we believe that our hierarchical design will also meet *ExScal* requirements at the scale of 10,000 nodes. Our belief is based on the latency provided by the Tier-1 and Tier-2 network. For larger number of nodes, the increased latency in delivering messages at both the tiers would cause increased delay at the Tier-3 base station.

In summary, the take-away lessons for an extreme-scale sensor network project are as follows:

To contain cost, select nodes that have large sensing and communication ranges. Plan a topology – specifically a regular, hierarchical structure – which covers the regions efficiently. Exploit knowledge of this topology while designing the services for efficiently utilizing system resources.

To meet the desired quality of detection, decompose the entire system into multiple components which execute in different phases of operation. Decomposition simplifies the design, allows us to configure and manage each component separately (at run time if need be), and reduces operational resource requirements. Importantly, it frees us from using common services for all components; instead we can use different services optimized for components needs. However, decomposition introduces an additional task of checking the components agree on the semantics of the passed parameters.

For cost-effective human manageability, operate by command and control. The operator initiates, monitors, and regulates the operations. He maintains the system based on the feedback obtained from the network. Autonomous functions, specifically, configurable recoverability for components, tolerance to several classes of faults (often by self-stabilization), and adaptation to certain classes of variable, non-uniform environments all support containment of human effort.

To gain confidence in the correctness of such a large system, test the system on multiple platforms; simulation, emulation (testbed), and outdoor testing will all reveal different faults. The platforms are mentioned in the ascending order of cost to perform tests at a given scale and the accuracy of detecting errors.

Acknowledgments We would like to thank Rajiv Ramnath, Emre Ertin, Prasun Sinha, Sandip Bapat, Vinod Kulathumani, Hongwei Zhang, Hui Cao, Mukundan Sridharan, Santosh Kumar,

Nick Seddon, Chris Anderson, Ted Herman, Nishank Trivedi, Chen Zhang, Mikhail Nesterenko, Romil Shah, Sandeep Kulkarni, Mahesh Aramugam, Limin Wang, Mohamed Gouda, Young-ri Choi, David Culler, Prabal Dutta, Cory Sharp, Gilman Tolle, Mike Grimmer, Bill Ferriera, and Ken Parker for their help in realizing Project ExScal. The project was conceived in and funded by the DARPA NEST research program, with Vijay Raghavan serving as program manager.

References

1. Arora A et al (2004) A line in the sand: a wireless sensor network for target detection, classification, and tracking. Comput Netw J 46(5):605–634
2. Bapat S, Kulathumani V, Arora A (2005) Analyzing the yield of ExScal, a large-scale wireless sensor network experiment. In: Proceedings of the 13th IEEE international conference on network protocols ICNP. IEEE Computer Society, Washington, DC, pp 53–62
3. Bapat S, Kulathumani V, Arora (2005) A reliable estimation of influence fields for classification and tracking in unreliable sensor networks. In: Proceedings of the 24th IEEE symposium on reliable distributed systems (SRDS 05), pp 60–72
4. Dutta P, Grimmer M, Arora A, Bibyk S, Culler D (2005) Design of a wireless sensor network platform for detecting rare, random, and ephemeral events. In: 4th International conference on information processing in sensor networks (IPSN'05 – SPOTS Track), p 70
5. Arora A, Sinha P, Ertin E, Naik V, Zhang H, Sridhara M, Bapat S (2005) Exscal backbone network architecture. In: Proceedings of the 3rd international conference on mobile systems, applications, and services (MobiSys 2005)
6. Intel Stargate specifications. www.xbow.com/Products/Product_pdf_files/Wireless_pdf/6020–0049–01_B_STARGATE.pdf
7. Kumar S, Lai T H, Arora A (2005) Barrier coverage with wireless sensors. In: Proceedings of the 11th annual international conference on mobile computing and networking (ACM MobiCom), pp 284–298
8. Hazelton NWJ (2004) Report on mark placement on node deployment
9. Polastre J, Hill H, Culler D (2004) Versatile low power media access for wireless sensor networks. In: Proceedings of the 2nd ACM conference on embedded networked sensor systems, pp 95–107
10. Hui JW, Culler D (2004) The dynamic behavior of a data dissemination protocol for network programming at scale. In: Proceedings of the 2nd ACM conference on embedded networked sensor systems, pp 81–94
11. Tolle G, Culler D (2005) Design of an application-cooperative management system for wireless sensor networks. In: Proceedings of the 2nd IEEE workshop on wireless sensor networks, pp 121–132
12. Choi YR, Gouda M, Zhang H, Arora A (2004) Routing on a logical grid in sensor networks. Technical Report UTCS TR-04–49
13. Zhang H, Arora A, Choi Y R, Gouda M (2005) Reliable bursty convergecast in wireless sensor networks. In: Proceedings of the 6th ACM international symposium on mobile ad hoc networking and computing, pp 266–276
14. Zhang H, Arora A, Sinha P (2005) Learn on the fly: quiescent routing in wireless sensor networks. Technical Report OSU-CISRC-4/05-TR20
15. Naik V, Arora A, Sinha P, Zhang H (2007) Sprinkler: a reliable and energy efficient data dissemination service for wireless embedded devices. IEEE Trans Mobile Comput 6(7):777–789
16. Sebastian J, Keshavamurthy S, Anand SVR, Hegde M, Kuri J, Kumar A Status report on WSN implementation on COTS devices. Technical report TR-DRDO-IISc.WSN-2009.1
17. Naik V, Ertin E, Zhang H, Arora A (2006) Wireless testbed bonsai. In: Second international workshop on wireless network measurement (WiNMee) held in conjunction with the 4th international symposium on modeling and optimization in mobile, Ad Hoc, and wireless networks (WiOpt)

18. Robinson J, Papagiannaki K, Guo X, Diot C, Krishnamurthy L (2005) Experimenting with a multi-radio mesh networking testbed. In: 1st Workshop on wireless network measurements (WiNMee)
19. Levis P, Nelson L, Welsh M, Culler D (2003) TOSSIM: accurate and scalable simulation of entire TinyOS applications. In: Proceedings of the 1st international conference on embedded networked sensor systems, pp 126–137
20. Girod L, Ramanathan N, Elson J, Stathopoulos T, Lukac M, Estrin D (2007) Emstar: a software environment for developing and deploying heterogeneous sensor-actuator networks. ACM Trans Sensor Netw 3(3):13
21. Woo A, Tong T, Culler D (2003) Taming the underlying challenges of reliable multihop routing in sensor networks. In: Proceedings of the 1st international conference on embedded networked sensor systems, pp 14–27
22. Stajano F, Anderson R (2000) The grenade timer: fortifying the watchdog timer against malicious mobile code. In: Proceedings of 7th international workshop on mobile multimedia communications
23. Arora A et al (2005) ExScal: elements of an extreme scale wireless sensor network. In: Proceedings of 11th IEEE international conference on embedded and real-time computing systems and applications, pp 102–108

Chapter 9
Glacier Monitoring: Deploying Custom Hardware in Harsh Environments

Kirk Martinez and Jane K. Hart

Abstract This chapter describes experiences with designing and deploying sensor networks in a glacial environment. In the final version of the system, the nodes were designed to be sufficiently robust for the harsh environment, to be small enough to fit into a borehole and to have a design lifespan of 4 years on a battery. The nodes were a completely custom design based on a PIC microcontroller and supporting sensors to measure and report pressure, case strain, conductivity, pitch, roll and an LED and photodiode to measure reflectivity. The nodes reported data via an RF transceiver to a basestation based on the Gumstix ARM/Linux module. Over the course of 4 years of development, the design of the system evolved and design changes were evident in nearly every element of the system, from the platform used at the base station to the design of the sensor node to the choice of radio frequencies.

Keywords Deployment · Power management · Environmental sensing · Glaciers

9.1 Motivation for the Glacsweb Project

The aims of the Glacsweb project were to aid the understanding of subglacial processes, especially to investigate their links with climate change, as well as developing the next generations of environmental sensor networks [11]. The project developed all its own custom sensor network hardware and software. This was partly due to the lack of suitable off the shelf hardware and the need for annual development cycles but it also helped us to fully understand every element of the complex systems involved. The Glacsweb project deployed groups of nodes (called here *probes*) under temperate valley glaciers in Norway and Iceland. The progression of the system developments led to new data streams that had never been produced before in terms of both time-span and the range of sensors used. Between 2003 and 2006, 30 nodes were deployed in Briksdalsbreen, Norway. Subsequently, a further 8 nodes were deployed in Skalafellsjökull, Iceland in 2008.

K. Martinez (✉)
University of Southampton, Southampton, Hampshire SO17 1BJ, UK
e-mail: km@ecs.soton.ac.uk

E. Gaura et al. (eds.), *Wireless Sensor Networks: Deployments and Design Frameworks*,
DOI 10.1007/978-1-4419-5834-1_9,

The study of glaciers is a vital part of the research into understanding the Earth's response to climate change. The water stored as ice in ice caps and valley glaciers is used, for example, to provide drinking water, hydroelectric power and supplying rivers. Global warming is melting glaciers at a rapid pace, which will cause considerable problems in the future [14,20]. Glacier behavior is controlled by the interaction of ice and sediment, both within and underneath the glacier. These interactions are complex and have been the subject of many studies by glaciologists (e.g., [1, 18]). What was needed was an alternative to wired sensors [5,7,15], which would provide longer term and embedded sensing [9,10]. Typical wired glacier sensors contain one sensor per cable. The set of sensors include: water pressure sensors, strain gauges on a rod, and tilt cells or a reel-spool movement detector (a multi-turn potentiometer wound with wire to a fixed point). Nonetheless, if a fist-sized object could contain sufficient sensors to measure these parameters with a wireless link, it could replace the wired counterparts. The longevity of sensors would hopefully be improved due to the lack of wire-break risk, leading to both more data collected and more realistic subglacial sensing. Thus, in summary, the main probe requirements were to measure tilt, temperature, pressure, strain, conductivity over a period of years.

Associated with these requirements, the main glaciological objectives were:

- To produce a long term record of water pressures changes in the ice and subglacial sediment (till) in order to investigate subglacial and englacial hydrology.
- To produce the first ever record of three-dimensional probe movement, which can be used to test models of fundamental flow of inclusions within a viscous medium and demonstrate the rate of movement throughout the year.
- To carry on the first ever study of till temperatures within the till, in order to examine heat balance.
- To enable an investigation of the relationship between water pressure, till strength (case strain, resistance), and till temperature in order to understand till sedimentation, and till rheology.

In terms of sensor networks research, the requirements of the system led to a wide range of research questions including:

- How would the telemetry work? What frequencies are suitable in ice?
- What communications system would be needed to handle the varying conditions?
- Which sensors could be used? Which were most appropriate?

Fig. 9.1 Panoramic photograph of Skalafellsjökull, Iceland, site of the 2008 deployment, Photograph by K. Martinez

- How can a node survive on battery power for many years in such conditions?
- Would the electronics design have to be compromised to place it within the probe cases?
- What materials could be used to make the probe cases?
- How would the command and control be designed to ensure the safety of the nodes and data?

All these research issues led to developments in the lab and during fieldwork. Some would only be solved by full deployments while others could be tested well in the lab. It was already clear at an early stage that designing and building for a long time then deploying a perfect system was not feasible. This was mainly due to the unknown nature of the environment. After several development cycles we have learnt generically useful skills and techniques that enable sensor network deployments. Without the benefit of this knowledge, WSN researchers may find deployments extremely frustrating and prone to failure.

9.2 System Design

Our system comprises low power sensor network nodes placed in and under the ice, a base station on the surface, and communication links back to the UK.

The operating requirements are shown in Table 9.1, and our prototyping was based on these considerations. The sampling rate of the probes was initially fixed at once per four hours, partly because changes were expected to be slow but also to save power. Measurements from previous wired sensors were used in order to estimate a sensible sampling rate. With better power management, the last system deployed in Iceland was able to increase the sampling rate to once every hour. Although a complex adaptive sampling algorithm had been developed for the project [16], this was deemed too risky to be deployed in the field.

Repeated access to the glacier was not feasible, thus our approach was to plan a major deployment round every summer, with minor 2–4 person trips in either autumn or spring. A typical deployment cycle would involve design, testing and production of an incremental change to the systems within one year. These field visits were particularly important for highlighting radio communication issues but

Table 9.1 Operating requirements

Operating air temperatures	−20 to +30°C
Temperature in ice/till	−1 to +1°C
Radio frequencies required to transmit through 100 m temperate ice	Less than 500 MHz
Maximum diameter of probe case to fit borehole	Less than 0.1 m
Maximum depth of ice	100 m
Maximum ice pressure	900 kPa
Maximum water pressure	1,200 kPa
Battery life	4 years

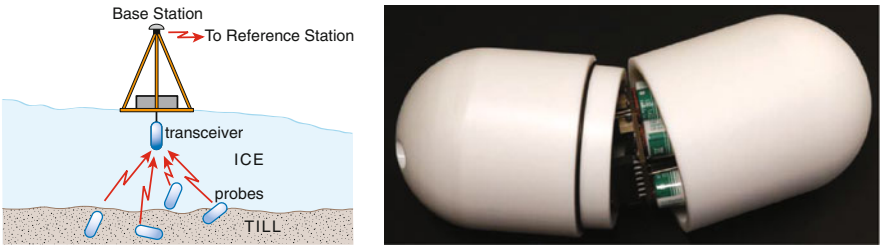


Fig. 9.2 The overall architecture with a single base station on the glacier surface controlling the probes in the ice/till. A long-range radio modem linked the base station to a PC in an Internet café 3 km away. Image used with permission copyright Springer 2009 (from [9])

also physical issues such as the effect of high pressure on the probe case. Core software could often be kept the same; the main changes typically involved the radio network. Similarly, some hardware modules such as power supplies, transceivers or amplifiers could often be reused. This yearly cycle forced us to be selective about which system components were the focus of each development cycle, otherwise key components may not have been completed on time. Unlike a purely laboratory-based system, once the fieldwork logistics started in spring, the deadlines were fixed and unmovable.

We expected long radio-disconnection periods but the users (glaciologists) wanted every data sample, even if it was delivered months later. Therefore, we used a large ring buffer (6,000 readings) for the data in each probe in order to store data for up to a year. The base station used the same principle, storing all datasets on compact flash, even after they were transferred to the UK.

GPS time synchronization of all the subsystems meant that it was easier to coordinate the system as it provided a globally synchronized clock source. Broadcasting the GPS time to all probes daily was used as a way to keep synchronization within seconds and hence save power by narrowing safety margins on wake-up schedules. While probe clocks remain synchronized, the disparity between base station time and probe time could drift by around 1 min per year.

Power management was the key to satisfying the requirement for long-term system life. Since a daily data transfer was acceptable, the radios on every unit were completely off most of the time and limited time windows were given to those tasks that used them (e.g., 1–3 min per day, maximum 3 retries per packet). The base stations only power-on units such as radios or GPS during their use.

9.2.1 Deployment

Fieldwork logistics should not be underestimated. In Norway, we took advantage of the well-established track up to the glacier, produced by the tourist industry. In the first years, it was even possible to occasionally use a horse and cart to carry heavy

equipment up the mountain. Unfortunately this mode of transport was replaced with battery driven vehicles because the increased summer temperatures lead to increased glacier melting leading to increased discharge in the connecting waterfall, which frightened the horses. An unpredicted outcome of "global warming"!

The probes were deployed by drilling holes in the ice, down to the bed, using a hot-water drill. The drill equipment weighs 200 kg and needs to be either airlifted by helicopter on steep glaciers or driven on, if access allows. The drill requires a constant supply of water, which means that summer is the only possible deployment time for probes. Nevertheless, other times of the year were found to be reasonable for base station developments. By drilling for a further 10–15 min at the bottom, a hole was formed in the sediment into which the probes could be placed. Some probes were placed in the ice by drilling a shallower hole. The boreholes could be empty of water or half full, presenting the users with difficult deployment scenarios, as the probes may not have survived a drop down the hole. The solution was to lower them down loosely attached to fishing line.

The base station needed to be stable on the glacier, which is constantly moving (about 15 cm/day) and melting (in summer this could be 20 cm/day). Our design was to build a pyramid structure that cut into the ice and needed no anchors (as these would just melt-out). In practice, a rope anchored down a deep hole (15 m) could also be used as a backup. On one occasion, when the base station fell into an opening crevasse, it was held by the combination of the rope anchor and the RS232 cable attaching it to the wired radio.

Melkevol's Internet café at the bottom of the valley proved to be an ideal place to link a small PC to the ISDN line (64 kbit/s) in order to provide a communication link between the glacier-based WSN and the Internet. Initially, we tested 2.4 GHz radio modems but discovered that this frequency is particularly poor in mountainous regions with no line of sight. We then tried 500 mW 466 MHz modems and found them to be more suitable to the conditions. Although ISDN is slow by modern standards, the bandwidth bottleneck, when dialing in from the UK, was the 9,600-baud radio modem link between the Internet café and the base station. Nonetheless, there was an unexpectedly long latency on the ISDN link, which hampered any interactive programming or debugging over this link. A later upgrade to ADSL in the Internet café made reverse connections from the UK much easier (and cheaper). Nonetheless, tuning the PPP link on the radio modems took much longer than expected and the link could sometimes fail. PPP is a standard point-to-point protocol, which simplified our work by allowing TCP/IP protocols to be used over the serial connection but which is complex to configure. When connected remotely, the programmers always had to consider what the consequences would be if the radio link went down for the day when only half way through a delicate operation, due to the unstable nature of the PPP link.

9.2.2 Probe Evolution

When we started developing in 2001, there was no commercial hardware usable for the probes. We also felt that building a completely custom design would let us learn

more about the system issues. Later on, when commercial WSN nodes became common, it was still preferable to use our own hardware since it had power control and full sensor interfaces such as amplifiers and bridges (for strain and conductivity). The radio systems available on commercial nodes are also not suitable since they typically use a frequency that is too high (e.g., 2.4 GHz) or have insufficient power (e.g., 10 mW). We used either our own power amplifier with a transceiver or a high power unit (e.g., 100 mW) since tests with 10 mW radios showed them to be limited in range to tens of meters through ice.

The diameter of the probes was determined by the typical width of the boreholes. Eventually the electronics was miniaturized sufficiently for the probe cases to shrink accordingly and hence fit down the hole more easily. The probe case designs were tested in a high-pressure water testing facility in the National Oceanographic Centre in Southampton, so the probe enclosures could be tested for water leaks. In practice there was a considerable amount of craft required to place the sensors in the cases, seal behind them with potting compound and finally sealing the probe cases. The choice of polyester for the cases was due to its RF properties, low water absorption and strength (see Fig. 9.3).

The probe electronics improved with each version in terms of power consumption. Reducing the sleep current was the main objective and this was eventually reduced from 10 to 1 μA. Subtle changes were needed to reduce energy leakage and lower the noise from the power supplies so that sensors were not affected.

When we started developing the probes there was no established operating system available for WSN nodes and so we developed our own. This had the advantage that we understood what every line of code was doing and could test each module easily. Nonetheless, the bespoke operating system took time to create and lacks any inherited modules obtained from an open source package such as TinyOS or Contiki. The overall structure of the code was maintained for a number of years because its features and risks were well known. It was tightly coupled to our custom packet structure and its command structure as well as its design methodology.

Each probe has a range of sensors, which had to be selected carefully. The original temperature sensor, which was a standard package with a wide operating range, did not provide fine enough resolution given that it would spend most of its life at

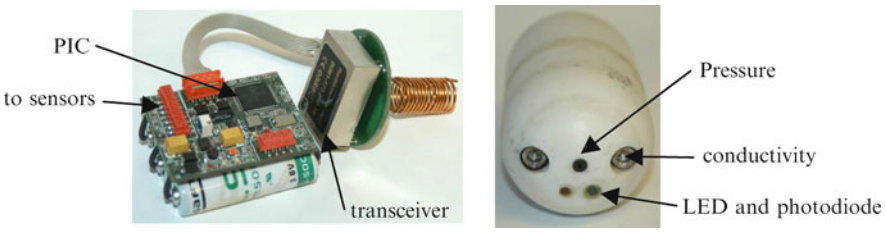


Fig. 9.3 *Left* hand figure shows the probe electronics (from 2006) with the Radiometrix transceiver and custom antenna. *Right* hand figure presents the end view of a probe showing the resistance bolts, pressure gauge (*centre*) and LED/sensor (*bottom*)

around 0.5°C. The coarse resolution meant that the data steps were large (0.06°C) in comparison with the small temperature variations. A custom thermistor design would require more electronics and calibration. The probe tilt was measured using a MEMS accelerometer. A 3-D visualization program was produced to show the probe's motion. Without this, the raw acceleration data would be difficult to interpret. The conductivity sensor simply used a pair of stainless steel bolts, which was a more robust design than using an off-the-shelf conductivity sensor. A pressure gauge was selected that would operate at pressures up to 50% higher than the predicted maximum under the ice. Case strain was measured using strain gauges, which were part of a Whetstone bridge. These were glued to the inside of the case and protected from the other electronics with a plastic cover. In practice, the strain gauges were the least reliable of all the sensors (with a 50% failure rate), presumably due to their delicate wiring. A light reflecting sensor was added to later probes to provide more information about the nature of the material the probe was embedded within. This consisted of a LED and photodiode. A side-benefit of this sensor was the fact that the probe showed some activity when sampling, which made prototypes easier to debug.

All of the sensors required calibration, mainly to determine their zero offset. A practical change that was made over the years is placing precision resistors on the PCB rather than using many tuning potentiometers for sensor amplifier settings. Using precision resistors meant that the systems required separate calibration readings to be stored but avoided the space use and risk of potentiometers on board. The calibration data and conversions could be carried out on the final server or in the probes. The latter approach makes browsing the incoming data easier and secures the calibration data in the probes. So from the user's perspective that is better, however developers may want access to the raw data for debugging purposes. In practice, both raw and calibrated data were stored on the base station in separate files.

Although published figures for ice suggested losses of only 10 dB over 100 m [6] even at 868 MHz we found that transceivers at that frequency were not usable. Switching to 433 MHz we achieved a range of 35 m and later at 173 MHz we obtained over 100 m range. Measurements showed that when the glacier surface was wet most of the radio losses were caused by the top 20 m of ice. This led to a decision to place an additional transceiver at least 20 m down a borehole attached by an RS232 cable [9]. The transceiver for several deployments was standardized around the Radiometrix BiM unit [17] tuned to 173.25 MHz but powered at 100 mW rather than the default 10 mW. Compared to modern transceivers this unit is primitive – requiring careful driving of its analogue input stage and bit balancing (we used Manchester encoding then later FEC – Forward Error Correction). The transceiver interface was trickier than normal systems because the digital input was directly driving the FM deviation and the output was analogue and needed a fast comparator in order to capture data properly. A small, round PCB was used to mount the antenna and provide a small ground plane for it (see Fig. 9.3).

Many of the features of the radio communications were designed to maximize the performance of the system according to the user's needs while balancing system risks. For example only three retries due to packet errors were allowed in order to

prevent excessive amounts of energy being used while attempting to transmit only one day's data. In this environment, poor communications are likely to be caused by unfavorable environmental conditions and thus communications performance should eventually improve when the weather changes. When a new radio design was produced it would only be completely tested during the deployment, due to the difficulty of accessing a glacier at other times of the year. For example, the tuned 173 MHz helical antennas made in 2006 were tuned only after an experiment on the glacier transmitting data between two test holes.

In order to simplify the task of building the first systems, we used a star network topology, which was quick to build and easy to debug although it reduced the range of the network. Implementing one of the published multi-hop, ad-hoc network routing algorithms was considered too risky for the first deployments. Power management, robustness and sensor integration were the main elements emphasized initially. Adaptive behavior was also minimized in the base station initially in order to ensure higher reliability and ease of control. These pragmatic choices led us to have a working system within two years with only one main developer. The architecture later evolved to include ad-hoc networking [4] and a more advanced radio design. A TDMA-based ad-hoc protocol was designed around the fairly static environmental conditions. Sadly, 2006 was the year when the whole of the front part of the glacier broke off and all of the equipment was lost [3]. This meant the new protocols could not be evaluated in-situ. Some principles, such as: long-term store and forward design to preserve the data, using GPS time to synchronize all systems, and using script languages for the main control code have been maintained throughout.

9.2.3 Base Station Evolution

A reliable base station is a vital component of a sensor network. Power consumption considerations led us to initially build a complex PIC-based base station PCB that contained all the functionality needed to switch power, wake-up and communicate. However many of the peripherals that had to be used, dGPS, radio modem, weather station, required software that was not suited to the PIC. Also the expectation of producing a perfect system and not needing to update its software was clearly overambitious. Fortunately low power ARM-based boards running embedded Linux had emerged which would solve the software issue. The increased sleep power these require would be compensated for by large batteries and wind/solar generators and later designs had comparable sleep requirements.

Figure 9.4 shows the evolution of the base station design. A design lesson we learnt was to have switchable LEDs for debugging and terminals that are easy to use in cold weather. The Bitsy [2] Unix system provided a platform for software development that could be mirrored on a laptop. This made it easier for a developer to build and debug an application separately, especially for serial-connected peripherals such as the GPS, weather station and GSM. Unfortunately the sleep power was fairly high (120 mW) and the build environment lagged behind developments on other embedded Linux platforms. This led to the use of the Gumstix [8] platform

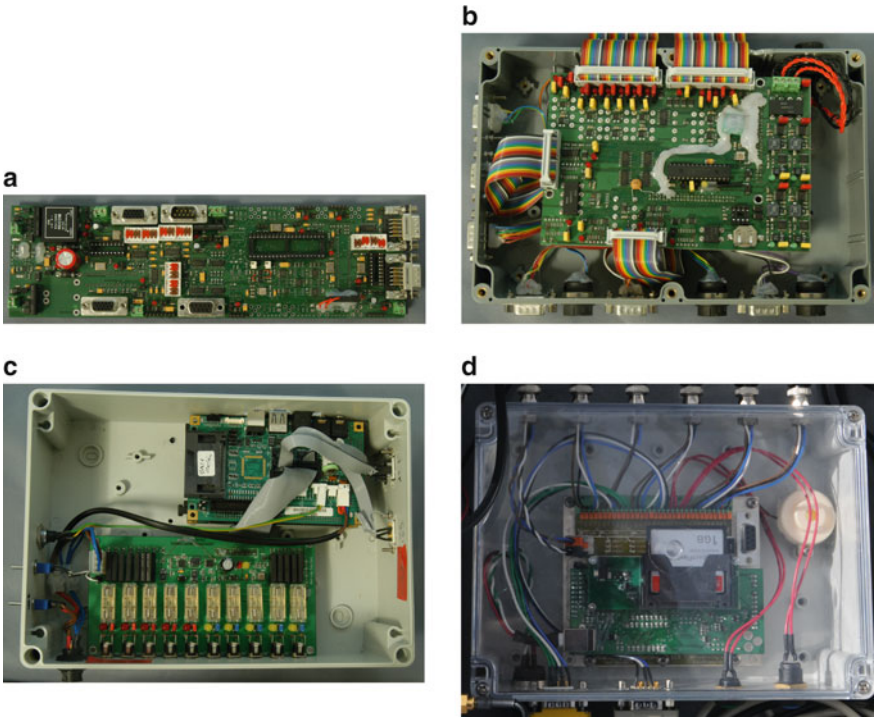


Fig. 9.4 (**a**) Early PIC-based base-station (2004), (**b**) Deployed PIC-based base station (2005) showing an increased use of screw terminals and common connectors. (**c**) Bitsy-based basestation (2006) with modular switched power-supplies, (**d**) Gumstix-based system (2008)

for the Linux part even though sleep/standby was not a solved issue for the ARM. A microcontroller was used to control power to the peripherals. The latest base station deployed in Iceland used our custom MSP430 add-on PCB [12, 13] that also controlled power to the Gumstix. Rather than using a standby mode of the ARM (as in the Bitsy), the microcontroller powers-up the Gumstix, which, while using more energy in start-up, results in a low sleep power (72 μW) and hence longer life in this situation. In Norway, the wind power in the winter was enough to maintain high battery charge. In Iceland snow prevented any charging by burying the blades of the wind generator and preventing them turning, so the lower sleep power was essential.

One lesson learnt from prototyping on a different platform (e.g. laptop) is that small differences in hardware, even in serial port handling, can cause problems once the code is run on the embedded platform. Issues with "sleeping" level translators (RS232 devices designed to save power but which can cause problems) and USB-to-serial converters were difficult to diagnose. The overall power consumption of a sleeping base station was once measured as 12 mA, which was entirely due to the commercial solar regulator used. Although debugging LEDs could be switched off, the internal circuit of the regulator was not designed for low power situations.

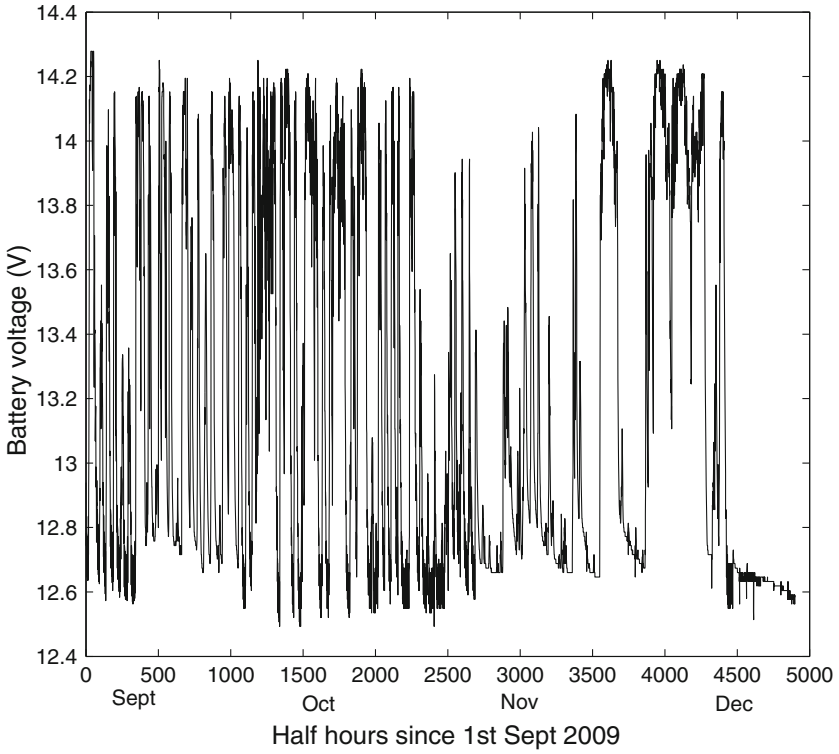


Fig. 9.5 Battery voltage of the base station in Iceland

Testing in the laboratory is best carried out on the real hardware even if it is more time consuming.

The graph showing base station battery voltage in Fig. 9.5 demonstrates how the voltage rises and falls diurnally due to solar charging and has bursts of high values on windy days due to wind power charging. The right half shows the different behavior due to synchronizing with the reference system's once-per-day GPS sampling, which causes regular dips. These data show that a daily average of the battery voltage is needed in order to reduce the noise in the estimate of the usable energy reserves.

The experience from successive deployments provided clear lessons, which, if heeded, can drastically improve the performance and reliability of future systems. Enclosures and cases are one area where years of experience already exist (e.g., in the housing of weather stations and seismic monitoring systems). Connectors, which are an unfortunate necessity when solar panels, antennas and external sensors are used, are another area where learning from commercial systems and past experience is essential. We found that "Pelican" cases and large, over-specified (in terms of maximum rated voltage and current load) connectors (e.g., Amphenol MS3102) were an essential part of the project's risk-reduction and have been used extensively

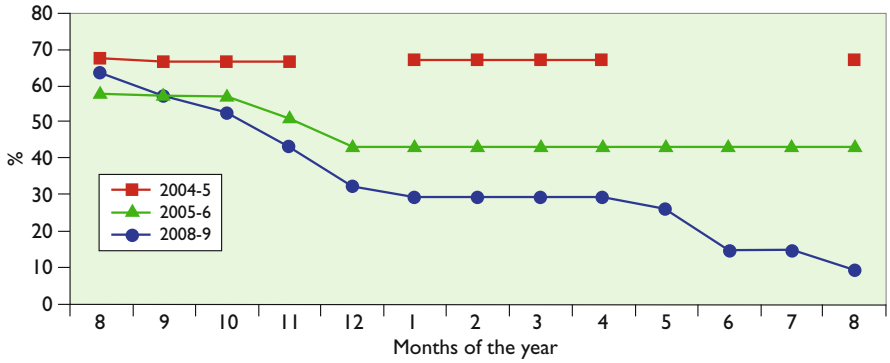


Fig. 9.6 Percentage of expected data (per month) obtained each year: 2004–2006 Briksdalsbreen; 2008–2009 Skalafellsjökull

in commercial products. Another lesson was labeling the external connectors to allow a non-expert to interact with the system ("disconnect the GPS", for example). We typically enclosed the core base station electronics in a smaller box within the main case, to protect it from rain. This requires double the number of connectors for external modules but is far safer and allows the main electronics to be removed safely for debugging elsewhere.

Figure 9.6 presents rate of decay of the deployed systems, which shows that the probes are slowly lost over time. However, it can be seen that the success rate has improved each year. The actual reason for the disappearance of probes is unknown as there is typically very little information that changes drastically before they vanish. There are no sudden sensor reading changes to indicate that a probe is about to be crushed or moved away. Battery power was discarded as a failure issue as the predicted lifetime is long and the measured voltage is always high. It is more likely that they are suddenly crushed or washed away by a subglacial stream.

9.2.4 WSN Advantages for the User

Previous glacier sensing used wired instruments. However wired sensors cannot move freely in their environment and it is difficult to know how this affects the data. The cables are also prone to damage within a shearing glacier. Moreover, wired deployments require large quantities of cable, which were heavy to carry as well as difficult to manage on the ice surface. Radio linked sensors have range restrictions in a star-configuration but are easier to manage. Ad-hoc networking clearly makes probe positioning easier than wired sensors as well as reducing the risks of restricted radio range. WSN nodes can be placed in locations that would be almost impossible with other techniques, for example in a subglacial shear zone where a cable would soon be damaged.

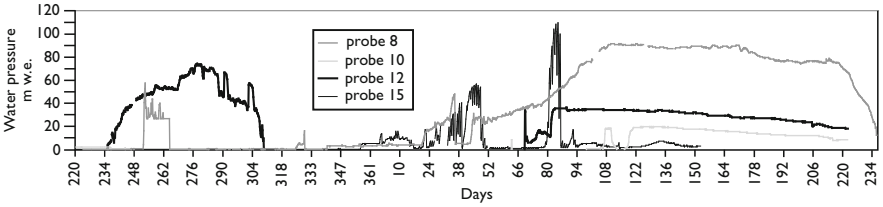


Fig. 9.7 Probe water pressure data from Briksdalsbreen till probes

The deployments reported on in this chapter provided long-term records stretching over several years. Figure 9.7, for example, shows the pressure data for three probes acquired during two years. Simply acquiring continuous data over such a long period is a major improvement because several years and sites can be compared.

Overall, the Glacsweb project was very successful and the results can be summarized as follows:

- It was demonstrated that the deployed WSN system works very well. In 2004–2005, a total of 859 days of probe data (36,078 sensor readings) were received and one probe lasted for the whole year. After further development in 2005–2006, a total of 1,208 days of probe data (50,715 sensor readings) were received, and three probes lasted for the whole year. In Iceland 784 probe-days (169,000 readings) have been collected between August 2008 and December 2009.
- It was shown that annual water pressure changes were more complex than reported elsewhere in the literature. The probes showed a similar pattern of low winter water pressures, a two-stage spring event associated with the opening of the glacier hydrological system, and high summer water pressure (Fig. 9.6) [19].
- The amount and nature of clast (probe) movement within the till was quantified for the first time, and this data was used to demonstrate how obstacles move within a sheared viscous material, which has important implications for other environments such as debris flows and shear zones as well as till fabric studies [9, 10, 18].
- We were able to measure till temperatures over the whole year for the first time. We were able to compare our results with theoretical modeling to show two important processes occurring at different times of the year: (a) melt-water inputs cooled the till, (b) high friction (deformation) increased till temperatures.
- We were able to demonstrate that during high water pressure, the till behaved as a linear Bingham viscoplastic flow. This data is required for glacier modeling, as well as being a vital contribution to the ongoing debate concerning till rheology. Winter low water pressures were associated with low water contents and high till strengths resulted in brittle deformation; high spring water pressures, high water contents and low tills strengths resulted in basal sliding, bed separation and hydrofracture; the summer intermediate water pressures were associated with ductile deformation. It is the superimposition of these processes over the year that produces the dynamic range of structures observed in both modern and ancient subglacial sediments.

9.3 Summary of Recommendations and Observations

Wireless sensor networks are best developed and deployed in close collaboration with the domain experts. Any first deployment should be expected to reveal hidden issues and may well seem unsuccessful but it is a necessary step to producing a working system. In developing WSN systems fit for in-field deployment, a wider range of expertise is needed than might be expected, including mechanical engineering, testing, calibration, rapid prototyping, physics, and good humor. Planning for unexpected fieldwork is essential because the natural environment is so unpredictable and these trips also allow for incremental improvements or debugging.

For the deployments reported here, there is still an issue of usability (as with most sensor networks), partly due to the timescales involved in Glacsweb: the configuration and management of the system was rather unfriendly to end-users, in comparison with data loggers and off-the-shelf telemetry systems already used by the environmental monitoring community; the off-the-shelf systems can be configured by end-users with the aid of a manual. It is hoped that through long-term deployments together with real users involvement, WSN technology can advance to the stage where it is as commonly used.

Acknowledgments The authors would like to thank Inge and Gro Melkvoll for their help with logistics and all the members of the Glasweb team: Dr Royan Ong, Elisa Anderson, Tom Bennellick, Hannah Brown, James Cheshire, Kim Dowset, Dr Ahmed Elsaify, Jeff Gough, John Hunt, Natalie Jarman, Daniel Miles, Dr Paritosh Padhy, Celine Ragault, Alistair Riddoch, Stuart Rimmer, Dr Kathryn C. Rose, Robert Spanton, Sarah Stafford, David Vaughan-Hirsch, Dr Richard Waller, Sarita Ward, Mathew Westoby and Dr Gang Zou. Thanks also go to Bib Smith in the Cartographic Unit, School of Geography for cartography. This research was funded by the Royal Society, DTI NextWave programme and ESPRC.

References

1. Alley RB, Blankenship DD, Bentley CR, Rooney ST (1986) Deformation of till beneath ice stream B, West Antarctica. Nature 322:57–59
2. Applied Data (2009) BitsyX - PXA255. http://www.applieddata.net/products_bitsyX.asp. Accessed July 2009
3. BBC news (2006) Report on retreat of Briksdalsbreen glacier. BBC News, http://www.youtube.com/watch?v=CY8AagMh_1M. Accessed Nov. 2009
4. Elsaify A, Padhy P, Martinez K, Zou G (2007) GWMAC – A TDMA Based MAC Protocol for a Glacial Sensor Network. In: 4th ACM PE-WASUN, October 22–26, Chania, Crete Island, Greece
5. Engelhardt H, Kamb B (1998) Basal sliding of Ice Stream B, West Antarctica. J Glaciology 44(147):223–230
6. Evans S, Smith BME (1969) A radio echo equipment for depth sounding in polar ice sheets. J Phys E 2(2):131–136
7. Fischer UH, Clarke GKC (2001) Review of subglacial hydro-mechanical coupling: Trapridge Glacier, Yukon Territory, Canada. Quat Int 86:29–44
8. Gumstix, www.gumstix.com - Accessed Nov. 2009

9. Hart JK, Martinez K, Ong R, Riddoch A, Rose KC, Padhy P (2006) An autonomous multi-sensor subglacial probe: design and preliminary results from Briksdalsbreen, Norway. J Glaciology *52*(178):389–397
10. Hart JK, Rose KC, Martinez K, Ong R (2009) Subglacial clast behavior and its implication for till fabric development: new results derived from wireless subglacial probe experiments. Quat Sci Rev 28:597–607
11. Martinez K, Hart JK, Ong R (2004) Environmental sensor networks. IEEE Comput 37(8): 50–56
12. Martinez K, Basford P, Ellul J, Spanton R (2009a) Gumsense - a high power low power sensor node. In: 6th European Conference on Wireless Sensor Networks
13. Martinez K, Hart JK, Ong R (2009b) Deploying a Wireless Sensor Network in Iceland. In: Lecture Notes in Computer Science, Proc. Geosensor Networks, vol. 5659, pp. 131–137
14. Nick FM, Vieli A, Howat IM, Joughin I (2009) Large-scale changes in Greenland outlet glacier dynamics triggered at the terminus. Nature Geoscience, 2, pp. 110–114
15. Murray T, Porter PR (2001) Basal conditions beneath a soft-bedded polythermal surge-type glacier: Bakaninbreen, Svalbard. Quat Int 86:103–116
16. Padhy P, Dash RK, Martinez K, Jennings NR (2006) A utility-based sensing and communication model for a glacial sensor network. In: 5th Int. Conf. on Autonomous Agents and Multi-Agent Systems, Hakodate, Japan. pp. 1353–1360
17. Radiometrix BiM: http://www.radiometrix.co.uk/products/bim1.htm accessed Nov. 2009
18. Rose J, Hart JK (2009) Quaternary Glaciodynamics. Quat Sci Rev 28:577–579
19. Rose KC, Hart JK, Martinez K (2009) Seasonal changes in basal conditions at Briksdalsbreen, Norway: the winter–spring transition. Boreas 38:579–590
20. Vaughan DG, Sponge JR (2004) Risk estimation of collapse of the West Antarctic ice sheet. Clim Change 52:65–91

Chapter 10
Adding the Human Element: Experience with a Wireless Patient Monitoring System

Dorothy Curtis, Esteban Pino, Thomas Stair, and Lucila Ohno-Machado

Abstract This chapter describes how the design and deployment of a wireless sensor network for patient monitoring was affected by human factors. The wireless sensor network was deployed and studied in the waiting area of an emergency department at an urban academic hospital in the United States. The broader application of this work is in the area of disaster management. The challenges were in the integration of this new technology into a pre-existing environment for caring for patients with minimal disruption. These challenges included operation restrictions mandated by the hospital's Institutional Review Board, choice of an appropriate localization subsystem, limited power, concerns about fire hazards, theft, and hospital security during the eighteen-month deployment. Issues such as user acceptance, patient mobility and expectations of multiple stake holders are also discussed.

Keywords Emergency department decision support · Wireless sensor networks · Ambulatory patient monitoring · Body sensor networks · Medical sensors · Personal digital assistants · Motes · Location tracking · Multiple stakeholder involvement.

10.1 Introduction

Wireless sensor networks bring monitoring to applications and phenomena that have been difficult or costly to monitor until now. One of these areas is monitoring patients waiting for medical care in busy emergency departments at hospitals. While some wireless sensors networks face challenging technical problems up front, such as high data rates or difficult weather conditions, placing a wireless sensor network on humans in a medical environment brings a different sort of challenge: the addition of the Human Element. Patients can be unpredictable and caregivers can take issue with the deployment at any time.

D. Curtis (✉)
Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139, USA
e-mail: dcurtis@csail.mit.edu

E. Gaura et al. (eds.), *Wireless Sensor Networks: Deployments and Design Frameworks*,
DOI 10.1007/978-1-4419-5834-1_10,

The hospital environment is complex and stressful: caregivers are focused on dealing with potentially life-threatening medical problems and do not want new technology to interfere with the performance of their duties; patients are interested in receiving care promptly. This environment is also protected by hospital rules and any experimental procedures or devices need to be approved by an Institutional Review Board (IRB).

The Scalable Medical Alert and Response Technology (SMART) system [6–8, 16, 17, 19, 22] monitors patients in the waiting area of an emergency department of a hospital. The SMART system, which monitors up to ten patients at a time, was deployed in the Emergency Department at the Brigham and Women's Hospital in Boston for eighteen months, between June, 2006, and December, 2007: 172 patients were monitored and 91 patients completed surveys. These surveys show that the system was well accepted.

10.1.1 Waiting Patients Need to Be Monitored

SMART is designed to provide continuous monitoring capabilities useful for assisting caregivers in recognizing important changes to patient health. Even in hospitals, generally considered safe, well-attended environments, patients spend time in areas where significant changes to their status may go unnoticed. Recent news articles have reported patients dying in the waiting rooms of emergency departments due to deteriorations not caught in time by hospital staff [1, 3, 4]. Further, caregivers have been chagrinned to find patients unconscious in restrooms. SMART aims to ensure that patients in hospital environments are noticed and located when their health deteriorates to the point that they need immediate care, even when they are not immediately visible to the Emergency Department (ED) staff.

To illustrate where SMART fits into an emergency department (ED), we describe how a patient who has just arrived interacts with the ED. First the patient describes his complaint, in a few words, to a clerk at the Registration Desk. Then the patient waits to be triaged by a nurse who briefly evaluates his condition and assigns him an Emergency Severity Index (ESI) [21, 23], which ranges from 1 to 5. An index of 1 indicates that the patient should receive a bed and attention from a caregiver immediately. Other values indicate that the patient should remain in the Waiting Area until a caregiver and a bed become available. Typically patients with "shortness of breath" or "chest pain" are assigned index numbers of 2 or 3. While these numbers indicate a high degree of urgency, the patient may in effect remain in the waiting area for several hours before a caregiver and a bed become available. While waiting, patients typically walk around the waiting area, visit the restrooms, or step outside for a smoke or some fresh air. During this time, however, a patient's status may deteriorate and the busy triage staff may not notice. Even if the staff receives an alarm concerning the patient's vital signs, they still need to locate the patient immediately. These are the patients and staff for whom SMART was designed.

10.1.2 Brief Overview of SMART

The SMART system itself consists of several subsystems: the patient monitoring subsystem, the central computer, the caregiver subsystem, and the location subsystem. The patient monitoring subsystem collects vital signs data from patients and forwards them to the central computer, which is called SMART Central. SMART Central performs several functions: (1) collects and saves data; (2) analyzes patient data for conditions needing caregiver attention; (3) maintains location information for patients and caretivers; (4) generates alarms to caregivers showing the patient's problem and location. The SMART system also includes Personal Digital Assistants (PDAs) for caregivers. With these PDAs, caregivers can receive alarms, indicate whether they will respond to these alarms, list the roster of patients wearing SMART pouches, and display patients' vital signs and locations in real-time and at earlier times. By maintaining location information on both patients and caregivers, alarms can be sent to the caregiver nearest the patient with a problem. Figure 10.1 shows the flow of data through the SMART system.

The patient monitoring subsystem, the SMART pouch, is worn by the patient, as shown in Fig. 10.2. Figure 10.3 shows the pouch which contains a PDA, a sensor box, ElectroCardioGram (ECG) leads and an oxygenation level sensor (SpO_2). The SMART operator, an Emergency Medical Technician (EMT) who administers the

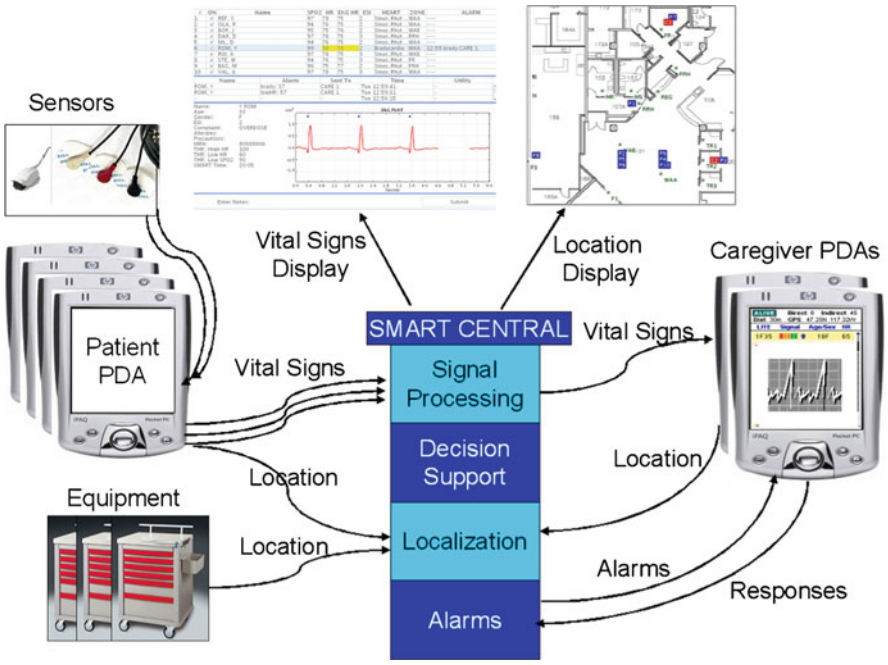


Fig. 10.1 Data flow through the SMART system

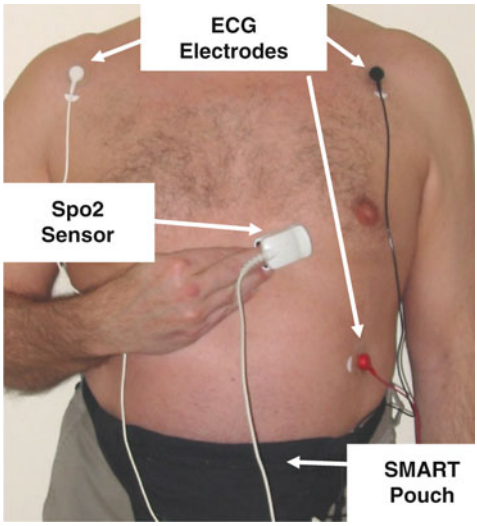


Fig. 10.2 Patient wearing a SMART pouch

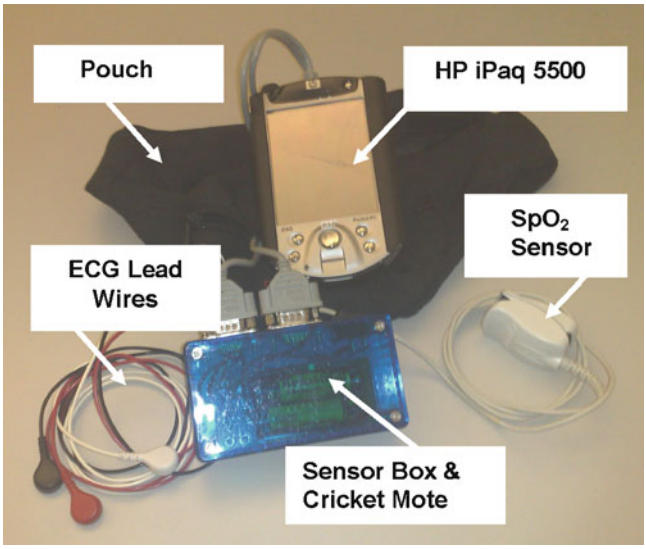


Fig. 10.3 Internals of a SMART pouch

SMART system, attaches the ECG leads to the patient's chest (or arms) and waist and the SpO_2 sensor to one of the patient's fingers. These sensors are connected to a Cricket Mote-based [5] sensor box which is, in turn, connected to the PDA. The PDA collects and timestamps the signals from the sensors and transmits them wirelessly to SMART Central.

The SMART system also includes a location tracking subsystem. This allows SMART Central to know where caregivers, patients and equipment are located and to assign an alarm to the caregiver closest to the patient.

The SMART system, which currently monitors up to ten patients at a time, was deployed in the Emergency Department at the Brigham and Women's Hospital in Boston for eighteen months, between June, 2006, and December, 2007: 172 patients were monitored and 91 patients completed surveys. These surveys show that the system was well accepted.

10.2 Application Goals and Requirements

The proposal to build the Scalable Medical Alert and Response Technology (SMART) system was developed in response to a Broad Area Announcement (BAA) by the National Library of Medicine (NLM), US National Institutes of Health (NIH). This announcement sought proposals in the area of application of next generation communications technologies to healthcare issues. This BAA was announced shortly after the September 11, 2001 attacks on the United States when systems to assist in disaster management were a top priority.

While supporting caregivers at a disaster site was our initial motivation, we ultimately decided to build a system that would be useful in urban hospital emergency departments. While new equipment can be introduced at disaster drills, these are limited in scope and time. Deploying the SMART system in an emergency department allowed us to study the feasibility of monitoring a number of patients wirelessly in a more controlled environment than at a disaster site. At times an overcrowded urban hospital emergency department approximates a disaster site when there are too many patients for the caregivers to monitor. An additional benefit occurs by having caregivers use the SMART system on a daily basis: emergency department caregivers gain familiarity with the system and will be able to use it effectively at a disaster site.

There are many stakeholders in this environment: patients, caregivers, the hospital IRB, engineers, and system designers. In the beginning, we believed we had representatives for these stakeholders on the team and that we understood their requirements. The patient wishes to receive good care from the hospital. The caregivers wish to provide the best possible care to all the patients at the hospital, but have limited time. The Institutional Review Board (IRB) at the hospital is also a proxy for both patients and caregivers: it seeks to protect the patients from harm and to maintain the current level of care provided to patients. The engineers want to build a reliable, maintainable system. The system designers want to see if this system can be integrated into the emergency department and whether it will improve care for the patients in the waiting area. However, we found, in practice, not all stakeholders were aware of all requirements when the study started.

The initial requirements were that

- The design of the system and study protocol be acceptable to the hospital's Institutional Review Board (IRB).
- The system integrate well with the existing procedures of the Emergency Department and not interfere with the Emergency Department's existing operations, either by distracting caregivers or by overloading their network.
- The patient monitoring system work within limited power and weight, be acceptable to patients and preserve their mobility.
- The system maintain position information for patients and caregivers, so that caregivers can find patients who are out of sight, such as in a restroom, when their status deteriorates and can allocate resources base on location.

10.3 Component and Sub-System Selection

The components that were needed were (1) patient monitoring node with integrated sensors for vital signs, (2) a caregiver node, (3) a location subsystem and (4) a central processing node. Since the goal of our project was to evaluate the feasibility of monitoring patients wirelessly in a specific context, we wanted to use reliable components. Using off-the-shelf, consumer or commercial grade hardware components reduced cost, shortened design time, and helped convince the hospital representatives that our devices were safe for use with patients. This latter point is particularly important for systems used in medical contexts, because there is a potentially lengthy approval process required for devices that come into contact with patients. A custom design might have resulted in a more lengthy review process. In designing the software, we preferred components with which the development team had experience to save on design time. For wireless communication we chose 802.11 b because the infrastructure has low cost and is commonly available and easily deployed.

10.3.1 Patient Monitoring Node Selection

For the patient monitoring node, we needed a wearable platform that had the ability to connect to multiple sensors, and had sufficient battery lifetime to collect patients' vital signs data and forward them wirelessly via 802.11 b to the central computer. We chose the commercially available HP iPaq Model 5,500 PDA (personal digital assistant) for collecting patients' vital signs data and forwarding them to the central server. This platform came with an ARM-based Intel XScale 400 MHz PXA255 processor, 128 MB of RAM, and 802.11 b wireless networking built-in. To simplify integration with the sensors, we used the optional "backpack" that provided extra battery power and slots for two PCMCIA cards. For an operating system, we chose to install the "Familiar" version of Linux [9]. We chose this over Windows CE

because several developers had familiarity with this operating system on the HP iPaq platform from prior work in Project Oxygen [14], and it enabled us to modify the operating system if that became necessary. For software development, we wanted a strongly typed higher level language. Initially we used Java on the iPaq, but after encountering memory limitations, we switched to Python.

To collect the patients' vital signs, we needed to choose sensors and integrate them with the iPaq. To this end we designed a sensor box that connected via serial ports to the patient monitoring node. The sensor box was based on the Cricket Mote, commercially available from Crossbow. We chose the mote initially because we planned to use Crickets for our location subsystem. It also provided a "daughter card" for interfacing with other sensors. Although the Cricket includes an integrated low power, short-range radio stack based on the Chipcon CC1000, we elected to use the 802.11 b interface on the iPaq to build our network. Using 802.11 b had two distinct advantages: it could make use of commercially available wireless infrastructure components, such as routers and devices with built-in 802.11 b support, and it eliminated the additional complexity of a multi-hop network. Further it was estimated to be much more reliable than a multi-hop network.

The patient monitor included two medical sensors: SpO_2 and ECG, which we interfaced via the sensor box. We used an SpO_2 sensor from Nonin to detect the oxygenation level in a patient's blood. Because we could not find a suitable ECG sensor with an acceptable price, form factor and open interface, we designed and built our own as a daughter card for the mote. The challenge in designing our own ECG sensor was acceptable signal quality. Our initial design based on a single op amp with two lead wires connected to the patient needed to be upgraded to use two op amps and three lead wires to achieve acceptable signal quality.

10.3.2 Caregiver Node Selection

The caregiver module had fewer requirements than the patient module: the primary requirements were support for a GUI, portability, and wireless communication with the SMART Central computer. While the display on the patient node was optional, it was a requirement for the caregiver node. By using the same HP iPaq PDA for the caregiver node as for the patient node, we retained a similar programming environment and hardware spares could be used for both purposes.

10.3.3 Location Subsystem Selection

We used the following criteria to evaluate potential location subsystems:

a. Approximate price of $25/tag, with a higher acceptable unit cost for the detection infrastructure.
b. Tags for people should be small, lightweight and easily worn; tags for equipment should be easily attached.

c. Room level location granularity.
d. Ease of installation, both indoors and outdoors; high availability; low maintenance.
e. Scalability: minimum: 10 patients, two caregivers, no equipment; maximum: 1,000s of patients, 100s of caregivers, 100s of pieces of equipment.
f. Caregivers should be able to turn off location tracking during breaks.

As originally proposed, there were two location subsystems, one for people (patients and caregivers) based on Cricket [18] and another for equipment based on RFID for several reasons. RFID looked like a good choice for equipment tracking because the tags are very low cost, whereas Crickets are preferable for tracking people because they can also be extended to support the medical sensors.

Although Cricket was an expensive solution, its cost could be amortized over the Cricket Mote's dual use for both location and ECG sensing. This also contributed to wearability because it eliminated the need for an additional component inside the SMART pouch. The installation can be quite easy, both indoors and outdoors, because the beacon infrastructure needed to support the Cricket system is wireless and battery powered. There is a trade off here between ease of installation and maintenance: if the beacons are run on batteries, the batteries must be replaced every three weeks.

Since the Cricket location architecture is based on passive receivers, it is scalable to an arbitrary number and density of receivers. Beacons, the active component of the system, are deployed at a fixed density that roughly corresponds to the granularity of location. Passive receivers also facilitate the protection of the caregivers' privacy: by default the central computer does not receive localization information. The software at the receiver can choose to share this information with others. This could be important for caregivers who might choose not to share their location during breaks.

After a period of evaluation, we decided that the Cricket Location system did not meet our criteria for accuracy: the receivers occasionally indicated that a nearby patient was far away. We considered several possible algorithms for filtering outliers, both receiver and server-based. However, we were unable to implement a solution based on Cricket, due to limited development resources.

By this point, we had reviewed several other indoor location systems, including a successful demonstration of the Sonitor Indoor Positioning System in the waiting area of the hospital's emergency department [20]. This demo showed that the system worked well and met many of our criteria: the tags were small and lightweight; the accuracy was reasonable; the system was commercially available; and a company was there to stand behind the installation. The Sonitor location infrastructure is composed of installed receivers that can operate via either a wired or a wireless network, although they require wired power. The room-level accuracy that Sonitor achieves is due to the fact that its ultrasound transmissions do not penetrate walls, but it is probably not a good choice for outdoor use. The system scalability was adequate for our immediate needs, but its use of emitter-based tags makes it architecturally less scalable than the Cricket system.

Initially we had planned to demonstrate equipment tracking using inexpensive RFID ($5–$25, vs. $200 for a Cricket tag), with the more expensive detectors (∼$5,000) installed at the doorways. However, because the ED has an "open plan" layout and the doorways were wide (10–15 feet), this was not feasible using the detectors available at the time, because of their limited detection range. The projected cost was raised further, since two RFID detectors are required to infer the direction a tag is going through a doorway. Given the challenges associated with equipment tracking, we decided to focus on people tracking, as it was more critical to our application.

10.3.4 Central Server Selection

A standard, commodity PC running Linux provided a reasonable basis for SMART Central. On SMART Central we used ORNetDB [13], a streaming database written in C++ that had been developed for another project. We modified this system to support multiple patients. ORNetDB defined the protocols for use with the PDAs and the Java-based user interface software on SMART Central.

10.3.5 Building Through Integration

The SMART system had many components and a limited development budget: its goals were attainable in large part through the use of off-the-shelf components. We encountered some limitations in using this approach. For instance, our initial choice for SpO_2 sensor used Bluetooth to communicate with the patient node. However, the Bluetooth implementation in Familiar Linux was not stable and we lacked the resources to expend sufficient effort to fix the Bluetooth driver in the operating system. These stability problems in the Familiar driver were eventually fixed, but we did not take advantage of it because by then we had committed to use the sensor box approach. Many components were independent of the hardware platform we used: for example, because the vital signs sensors were serial-port based, we could debug and evaluate them attached to an office PC. The patient nodes operated independently of each other, and all connected directly to the hospital WiFi infrastructure, rather than using cooperative multi-hop protocols as is common in many other wireless sensor networks (WSNs). In retrospect, choosing a different platform would have given us more flexibility in choosing larger batteries. The HP iPaq 5,500 is compatible with only specific models of a "smart" battery.

In choosing the location subsystem, we were challenged by the infancy of the field: there was no obvious choice. So we did a standard component selection: we made a list of criteria and invited several vendors in to present their products. We tried to visit working installations of the systems or asked the vendor do on-site demonstrations. Although this was time-consuming, it facilitated good assessments of the availability and the accuracy of the systems.

10.4 Software

The SMART system software comprised a broad set of components, including medical signal analysis and event detection, decision support, location tracking, an administrative web interface, and end-to-end sensor data acquisition and logging. In this section we describe these components and how they fit together into a coherent system. A large part of the complexity arose because the system needed to integrate several different types of hardware: Motes, iPaqs running Linux, the Sonitor infrastructure, and a Linux PC. Although care was taken to minimize the number of different operating environments involved, software needed to be written in three different environments: TinyOS on the Mote platform; Familiar Linux on the iPaqs for the patients and caregivers; and Fedora Linux, Postgres, Apache, and Python on the SMART Central PC.

The SMART software can be considered as four main components: two separate end-to-end systems that connect SMART Central to the patient and caregiver respectively, a location system implemented by Sonitor, and SMART Central, which includes the logging, analysis, and decision support features as well as a web front end.

The patient monitoring system required software on the iPaq to collect three data streams: ECG data from the Mote-based ECG, SpO_2 readings from an off-the-shelf sensor serial interface, and battery level. Once sensed, this data was time-stamped and forwarded to SMART Central, where it was processed in real time and stored into a database. For the Mote-based ECG sensor, custom developed firmware sampled the ECG sensor at 200 Hz and reported the data back to the iPaq via a serial port. The clocks on the PDAs and SMART Central were synchronized with each other using the standard NTP protocol [11].

The caregiver module connected to SMART Central and displayed real-time information to the caregiver. This information included (1) the roster of patients being monitored; (2) the display of a patient's current vital signs, including the ECG tracing in near real-time; (3) the display of a patient's vital signs since the beginning of monitoring; (4) incoming alarms. The caregiver module also handled the caregiver's response to an alarm, if any: the caregiver could (1) "accept" the alarm; (2) indicate that he was "busy"; (3) forward the alarm to another caregiver; or (4) do nothing. To prevent unauthorized access to patient data, the caregiver module needed to automatically lock the screen if the caregiver had not interacted with the PDA for a period of time.

SMART Central acted as an intermediary between the patient and caregiver systems and served to interpret and log patient data. Raw and derived patient data, caregiver alarms and responses, and the location history for both patients and caregivers was logged to a Postgres database. This database also recorded patient demographic information and anonymous survey information, input via a web application written for an Apache web server. This web server also updated a browser's map to display the locations of the patients and caregivers.

Because of performance limitations of the Postgres database for streaming data, we used an internally developed streaming database called ORNetDB to

process near real-time streams of vital signs data and provide those streams to other modules. Part of the ECG signal processing was accomplished inside the ORNetDB directly, including signal quality estimation (noisy vs. no signal vs. acceptable quality), heartbeat detection, and classification of heart rate. ORNetDB also provided data streams to other modules: the caregiver module on the caregiver iPaq, the process which logged the raw data into Postgres, and the SMART Central decision support program. The protocol by which raw sensor data were transported from the patient monitoring tags to SMART Central was also implemented as part of ORNetDB.

SMART Central processed the raw location data and used it to assign positions to patients and caregivers and to support decision-making. The SMART Central decision support program consisted of modules to

a. Further analyze the ECG signal to classify each patient's heart rhythm into one of the following: normal sinus, bradycardia (slow), tachycardia (fast), irregular, ventricular fibrillation, ventricular tachycardia, asystole, "noisy", or "leads off".
b. Process the SpO_2 signal: check it against thresholds and check for disconnection.
c. Display near real-time vital signs data to users at SMART Central and at the caregiver's PDA.
d. Decide whether any patients' vital signs were beyond thresholds or otherwise needed attention (e.g., "leads off"); select a caregiver to alert; send alarms; handle caregivers' responses or their failure to respond to alarms.

For the purposes of demos and debugging, we also wrote "playback" programs to replay patient data into the standard data processing flow.

10.5 Preparations for Deploying SMART in the Hospital

The deployment process in a hospital environment differs substantially from the process in many other contexts where WSNs are used, and much of this difference is associated with the regulatory environment that hospitals impose. Unlike other environments where development and debugging can be done iteratively "in the field", health care systems in hospitals must be implemented and characterized *before* they are allowed to be applied to real patients. Consequently, much of the early design and development could only be done in controlled test environments and with development team members as test subjects.

Before the system could be applied more broadly, any studies needed to be approved by the hospital's Institutional Review Board (IRB). This board meets periodically to review written proposals for research to be conducted at the hospital, to ensure patient safety, privacy, and ethical pursuit of research while maintaining the hospital's level of service to patients. Only after approval from the IRB could the system be fully tested with real patients in the ED.

As a part of this process, a biomedical engineer affiliated with the hospital must approve all electronic equipment that will be used on patients. He easily approved

the HP iPaq PDA because it was considered "consumer electronics." The sensor box was also readily approved because it was powered by standard AA batteries. One caveat was that the HP iPaq PDA was not to be plugged into wall power while the patient was wearing it, to reduce potential electrical hazards. The SpO_2 sensor developed by Nonin [12] had already been approved by the FDA. The biomedical engineer reviewing the equipment did recommend that we protect the electronics from fluids, so we housed the sensor box and iPaq in a water-resistant pouch. To prevent the spread of disease among the patients being monitored, we also developed a process for cleaning the outside of the pouches.

The plans submitted to the IRB included the patient consent form and a detailed description of the research and study plan. These plans included a description of possible negative outcomes to the patients from participating in this study and our plans for handling such situations; for example, we expected that some patients would develop a rash on their skin due to the adhesive on the electrodes. The physicians involved in the study were required to provide telephone contact information for dealing with these problems.

As a result of the IRB review, the original plan for deploying the system needed to be redesigned. The IRB determined that providing PDAs to the existing caregivers might distract and overburden them and thus interfere with the hospital's current level of service to patients. To resolve this problem, we hired a dedicated EMT (Emergency Medical Technician) known as the "SMART Operator" to supervise the SMART Central system. The SMART Operator obtained consents from the patients and handled all alarms.

The Operator was also required to watch the patients' raw ECG tracings and other vital signs at SMART Central, since, by virtue of the additional patient monitoring, the hospital would be held responsible if the patient's condition deteriorated and an alarm was *not* issued due to system malfunction. Additionally, the SMART Operator provided a point of contact for the team and reported any problems and successes to the development team.

The IRB was also concerned about interference with the hospital's network, so they required that we deploy our own wireless network. We deployed an off-the-shelf wireless router to allow the iPaqs and Sonitor detectors to connect to SMART Central. Because the router was not connected to the Internet, we did not expect that anyone would attempt to use it. However, this resulted in an unexpected failure mode: occasionally there were laptops in the waiting room that were statically configured to connect to the 192.168.0/24 subnet. These connections sometimes consumed router resources that the patient iPaqs expected to use. For our immediate purposes we handled this problem by changing the subnet that our router and clients used. However, this points to a more general problem: if a system such as SMART were life-critical, the network would need to be secured against use by devices not part of SMART.

To facilitate maintenance of the sensor boxes in the SMART pouches, the final version had two improvements (see also [2]): (1) external connectors for all the cables allowing them to be detached and replaced without opening the box; and (2) "hatches" for the AA batteries allowed the batteries to be replaced without opening the sensor box.

10.6 The Hospital Deployment Experience

The SMART system was deployed for 18 months at the Brigham and Women's Hospital in Boston, Massachusetts, USA, in the waiting area of the Emergency Department. The objective of the deployment was to study the feasibility of using a wireless patient monitoring system in such an environment. The system as configured was able to monitor ten patients at the same time. Caregiver PDAs were available but not normally used because of the IRB's concerns about distracting the caregivers. The PDAs connected to our WiFi router by 802.11 b wireless networking; SMART Central was installed on a standard PC and connected to the router by wired Ethernet. Location support for the deployment was provided by the Sonitor infrastructure, made up of 15 ultrasound detectors that connected to SMART Central by Ethernet. Some Sonitor receivers were connected directly to the router using wired Ethernet, while others were connected wirelessly using ASUS wired-to-wireless Ethernet access points.

This deployment lasted approximately 18 months, from mid-June 2006 through December, 2007. During this time, we monitored 172 patients who presented with "chest pain" or "shortness of breath", and collected 152 h of data for off-line analysis. Three patients were reprioritized on the basis of the SMART system's monitoring.

10.6.1 Installation and Operation of the Location System

Sonitor performed the deployment of their indoor location system at the hospital. We encountered both installation and operational problems with the Sonitor system. During installation, the ED personnel were concerned about dust from inside the ceiling drifting down onto the waiting patients while power and network cabling was run through the ceiling. To minimize impact on patients, this work was done early in the morning. After the system was operational, we encountered some power supply failures. The current required to support both the Sonitor detector and the wireless adapter exceeded the specification of the power supplies, causing some of them to fail. Sonitor quickly identified the problem and replaced the power supplies.

A second operational problem arose during the deployment, when a few of the detectors went missing. It is unclear whether the devices were stolen or whether they were removed by hospital personnel who were unaware of their purpose. This might have been prevented by adding labels that indicated that they were part of an authorized study in the hospital, as suggested in previous work [2]. As the study went on, we considered replacing the missing components, but in the end we chose not to replace them. Ultimately we found that, while this location system was good enough to track patients at room level accuracy, it was less operationally critical that we anticipated: since we were monitoring between zero and four patients per day (usually only one at a time), it was easy enough for the SMART Operator to track the patients visually. In a larger scale deployment, location would be more critical.

10.6.2 Battery Life and Power Management

To achieve a reasonable lifetime (>3 h) for the HP iPaq, we disabled the screen on the patient monitors. Because there were no problems requiring the use of the display during deployment, this was acceptable. The three hour limit was reasonable for patient monitoring in the emergency department's waiting area, as the patients were usually summoned into the ED proper about an hour after receiving a SMART pouch. There was only one instance where we needed to replace a patient's pouch because of low battery during monitoring.

A more complex problem with power emerged from the fact that the PDA only reported valid battery levels if the battery was fully charged when the PDA booted up. This problem was not discovered during the development process, because the PDAs were always kept plugged in when they were not in use. However, at the hospital, leaving the PDAs plugged in was considered a fire hazard. Because they could not be left charging, the reported battery levels were not reliable. The SMART Operator handled this situation by plugging in several PDAs when he arrived so that he could be certain that they would be "mostly" charged up when given to a patient.

10.6.3 Choosing a Location for SMART Central

We chose the location of the SMART Central computer with two goals in mind: (1) allow the SMART Operator to visually monitor the patients and (2) keep the system in a secure place while SMART was not in use. A hall off the waiting area seemed like an ideal spot. The hall was locked with a door, so the system would be secure when not in use and the door could be open when that SMART Operator was present.

The concern with fire hazards at the hospital arose again when there were too many papers in the area around SMART Central. The solution here was to buy a small filing cabinet. The lesson is that in a context dedicated to some purpose, all potential problems that might detract from that purpose must be minimized: the environment must be cleaner and safer than an "average" environment.

10.7 Discussion

The SMART project encompassed a variety of challenges, from the mobility of the patients to the appearance of new stakeholders during deployment, as well as the technical challenges of minimizing false alarms.

10.7.1 Hospital Deployment Life-Cycle

When developing a system for deployment in a hospital environment, pre-deployment tasks need to be budgeted carefully. If everything seems ready, the deployment may happen sooner than appropriate. In certain contexts, prototypes can be useful for getting feedback from the stakeholders, however they may be misleading because they may not reflect deeper-seated engineering issues.

Once the system is deployed, there are several barriers to iterating any part of it. For hardware on the patient, any change to the device would need to be re-approved by the biomedical engineering department and by the IRB. Even something as minor as adding a question to the survey form entails re-approval by the IRB. Changes are also to be avoided because the deployment is part of a study and changing the parameters of the study part-way through may invalidate previous datasets. Reporting the results of a study on a statistically significant number of patients is an important consideration for deploying a wireless sensor network in a hospital environment.

10.7.2 Preparing for the Reality of the Hospital Environment

We have previously discussed some of the impediments to deployment that are unique to specialized operating environments such as hospitals. In particular, the IRB process held the system to a higher standard prior to a real deployment. Apart from the particular test involved in the study itself, all of the testing needed to be done ahead of time so that the hardware and procedures to be used could be finalized prior to the IRB.

Testing with healthy volunteers is an important part of this testing process. While the IRB is still involved, some of its usual concerns do not apply because healthy volunteers are not considered as vulnerable a population as patients. In our early test deployments with healthy volunteers, we needed to develop crowd management techniques, because the healthy volunteers tended to be more numerous and less subdued than real patients. For the final healthy volunteer session, we asked the volunteers to arrive at fifteen minute intervals, so that we could meet with each one individually to get their consent, give them a pouch, place the electrodes and collect their demographic data. This proved far more manageable than having the healthy volunteers start up their own patient PDAs and place the electrodes themselves.

Testing with healthy volunteers helped us debug the system and some of the procedures for deployment, but it did not catch all the problems. While much of the system can be tested this way, the healthy volunteers behave quite differently and have very different sensor profiles compared with the real patients. First, the movement profile of healthy volunteers produced less noise in the ECG signals, which ultimately produced fewer false alarms than we saw with real patients. Second, readings taken from healthy subjects should not trigger the alarms and decision support parts of the system, so much of the system would not generally be exercised.

To better understand this process, we report on the development of the ECG processing algorithms, beginning with a prototype and ending with a system that could be deployed. The ECG processing algorithms were prototyped in MATLAB [10] by a graduate student specializing in biomedical signal processing. Several test data sources were used to test and validate the MATLAB implementation: (1) Physionet data [15]; (2) data collected with our ECG sensor from a patient simulator programmed to produce electrical signals consistent with various heart arrhythmias; and (3) data collected from some of the people working on the project.

After testing the algorithms themselves, they needed to be re-implemented in order to be integrated into SMART Central. We tested the integrated version in deployments using healthy volunteers. While these healthy volunteers did not have any cardiac arrhythmias, they provided a test of the ECG analysis algorithms running in real time on the real hardware.

However, real patient data turned out to be different from our data collections. The major evidence of this was that the system produced many more false alarms than we expected based on healthy volunteers. These were largely due to noise in the ECG signal due to patient motion. We were able to reduce the false alarm rate after reviewing the algorithm and implementation. Reviewing the implementation revealed several bugs that were repaired. We removed more false alarms by modifying the algorithms to fuse the ECG information with that provided by the SpO_2. We considered other approaches for detecting patient motion, such as via the location sensor or via an accelerometer within the sensor box. However, the particular location sensor we were using was not sensitive enough for this purpose and we lacked the development budget and time to explore the accelerometer option.

Ultimately, the final version of the system still had too many false alarms. We learned that it is important to collect a significant amount of "real data" as soon as possible. We recommend submitting an early application to the IRB so that data from real patients can be collected early on to help the developers produce a robust system. If this can be done early enough in the process, the hardware design can be modified if necessary (e.g., adding additional sensors). Whenever false alarms were a problem, the SMART Operator handled them by reviewing the raw data that produced the results and only passing them along to the ED staff when the findings merited it.

10.7.3 Working with Stakeholders During the Deployment

As has been discussed, there were many stakeholders involved in both the design and the deployment of SMART Central. Through our regular, weekly meetings, the engineers stayed in contact with the physicians who gave feedback on the features of the system. This practice in general was helpful in keeping all the team members aware of the status of the project. It did lead the development team, however, to focus on developing user interface software. This software is easy to demo and helps the user community better understand the vision we are presenting. Maintaining strong ties with the physicians required a non-trivial amount of time.

The Institutional Review Board (IRB) represented the interests of the patients and the existing practices of caregivers in the emergency department. We initiated our testing with healthy volunteers with an initial proposal restricted to healthy people. Thus, we deferred the proposal involving interaction with patients until most of the system was working.

Beyond the IRB, from time to time emergency room staff members would advocate for themselves or the patients: there were complaints about potential fire hazards due to charging the PDAs, excess paper in and around SMART Central, and, as described above, concern about toxic dust falling from inside the ceiling while cables were being run. After about six months of operation, the security department (a previously unknown stakeholder) notified us that the door to the back hall where SMART Central was located, needed to be kept locked. The SMART Operator adapted to this request by staying in the back hall with the system and opening the door once in a while to check on the patients, although he was not too happy about this request because he was be isolated from his patients and relegated to a small closed space.

In general, the stakeholder landscape in a hospital environment is complex, and it is difficult to identify all of the potential stakeholders ahead of time and resolve all potential issues. Many times, it is difficult to envision the full impact of a deployment until it is underway.

10.8 Future Work: Disaster Management

At a disaster site, a sensor system cannot rely on fixed infrastructure, and requires that the disaster management team deploy their system quickly so that they can provide service to the victims of the disaster. The main issue at a disaster site is that the victims far outnumber the caregivers, which implies that patient monitoring devices should be substantially cheaper than the other parts of the system. Since there are few caregivers, their first priority is to stabilize and triage the victims and then arrange transportation for those victims who need it. Between the victims who need immediate care and the mostly well victims are those in need of monitoring: they are not urgently ill, but their status is worrisome and might deteriorate. Having a portable, wireless patient monitoring system that includes a location system is a reasonable way to keep track of these patients while they wait for transportation, so caregivers can find them easily if their condition deteriorates.

10.9 Conclusions

Deploying a wireless sensor network into a sensitive or specialized environment such as a hospital is challenging. Beyond collecting and analyzing data, the WSN must be acceptable to the human subjects who will wear the monitoring devices

and the use of the WSN has to fit in an environment with an existing workflow and comply with constraints that the development team may have not anticipated. The hurdles between design and deployment make it difficult to collect "real data" early in the project.

Identifying stakeholders and communicating with them effectively was critical to the success of the project. During the deployment we stayed in contact with the emergency department staff, communicated with the hospital's Institutional Review Board, and complied with requests from the hospital security group and the triage staff.

In building WSN systems, one aspect is choosing the appropriate technologies to be used. While some technologies are mature and used by consumers, others are still evolving. In this project, we attempted to choose and use location systems even though the technology was not mature at that time.

The administrative challenges we faced in deployment are not fundamental roadblocks to eventual widespread use of wireless patient monitoring systems, but they complicate the development process. These issues also favor different development strategies. Whereas the design of many WSNs is focused on physical resource constraints such as energy or small size, systems deployed in a hospital environment must instead be optimized for logistical and administrative constraints. For example, a less efficient solution based on consumer electronics may be much easier to get through the regulatory process than an experimental system that is extremely power-efficient.

In conclusion, the SMART system has shown potential to identify life threatening situations and to alert caregivers automatically. Although the system produced more false alarms than were desirable, we believe that the false alarm rate could be improved using a larger patient dataset and minor algorithmic changes. In addition, a case can be made that there will always need to be a human in the loop to control the system and to check its results. By developing and testing user interface components such as the SMART Central GUI, we have taken steps toward addressing that need.

Given an aging population, we expect that the need for pervasive systems like this one will become essential to delivering high quality healthcare with minimum cost in non-disaster situations.

Acknowledgments This work was approved by the Human Subjects Research Committee at Partners Healthcare Inc (parent institution of Brigham and Women's Hospital). Parts of this work have been previously published in Curtis et al. [7]. We acknowledge funding support from the National Library of Medicine. NIH (N01LM33509). The SMART project would not have been possible without the efforts of our team members: Robert A. Greenes, Jason Waterman, Jacob Bailey, Eugene Shih and John Guttag. We would like to thank the following collaborators for their efforts in earlier phases of this project: Greg Harfst, Robert P. Fischer, Pankaj Sarin, Jennifer Morrissey, Ediza Giraldez, Robert El-Kareh, Hari Balakrishnan, Michel Goraczko and the Cricket team, HP® CRL, Michael Blechner, Staal Vinterbo, and Jennifer Carlisle. The indoor location system would not have been deployed without the support from Terry Aasen and the Sonitor® team. We also thank Stephen J. Nelson and the Stratus Center at Brigham and Women's Hospital for the opportunity to collect data from their simulation center, and Craig Schaffert for his comments on this paper.

References

1. ABC News Illinois woman's ER wait death ruled homicide Sep 17, 2006. [accessed 2010 Feb 1] DOI=http://abcnews.go.com/GMA/Health/story?id=2454685&page=1
2. Barrenetxea G, Ingelrest, F, Schaefer, G, Vetterli, M (2008) The hitchhiker's guide to successful wireless sensor network deployments. Proceedings of the 6th ACM conference on embedded network sensor systems. SenSys '08:43–56. DOI=http://doi.acm.org/10.1145/1460412.1460418
3. CBC News Woman dies in emergency waiting room, Oct 11, 2002, [accessed 2010 Feb 1] DOI=http://www.cbc.ca/canada/britishcolumbia/story/2002/10/11/bc_emerg021011.html
4. CNN, Tape shows woman dying on waiting room floor, July 1, 2008 [accessed 2010 Feb 1] DOI=http://www.cnn.com/2008/US/07/01/waiting.room.death/index.html
5. Crossbow® Technologies, Inc. home page. Wireless sensor networks;. Available from: http://www.xbow.com [accessed 2010 Feb 1]
6. Curtis DW, Bailey JM, Pino EJ, Stair TO, Vinterbo S, Waterman J, Shih EI, Guttag JV, Greenes RA, Ohno-Machado L (2009) Using ambient intelligence for physiological monitoring. J Ambient Intell Smart Environ 1(2)
7. Curtis DW, Pino EJ, Bailey JM, Shih EI, Waterman J, Vinterbo S, Stair TO, Guttag JV, Greenes RA, Ohno-Machado L (2008) SMART – an integrated, wireless system for monitoring unattended patients. J Am Med Inform Assoc 15(1):44–53
8. Curtis DW, Shih EI, Waterman J, Guttag JV, Bailey JM, Stair TO, Greenes RA, Ohno-Machado L (2008) Physiological signal monitoring in the waiting areas of an emergency room. Proc IEEE BodyNets 2008
9. The Familiar Project http://familiar.handhelds.org/[accessed 2010 Feb 1]
10. Matlab. Available from http://www.mathworks.com [accessed 2010 Feb]
11. Mills DL (1991) Internet time synchronization: the Network time protocol. IEEE Trans Commun 39(10):1482–1493
12. Nonin® home page. Nonin® Medical, Inc.; c2007: http://www.nonin.com [accessed 2010 Feb 1]
13. ORNetDB. Streaming database software developed for internal use at CSAIL, MIT
14. Oxygen http://oxygen.csail.mit.edu [accessed 2010 Feb 1]
15. Goldberger AL, Amaral LAN, Glass L et al (2000) PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals. Circulation 101(23):215–220
16. Pino EJ, Curtis DW, Wiechmann E, Ohno-Machado L (2009) A real-time diagnostic support system for ambulatory patients, in preparation
17. Pino EJ, Ohno-Machado L, Wiechmann E, Curtis DW (2005) Real-time ECG algorithms for ambulatory patient monitoring. AMIA Annu Symp Proc 604–608
18. Priyantha NB, Chakraborty A, Balakrishnan H (2000) The Cricket location-support system. Proceedings of the 6th annual international conference on Mobile computing and networking (MOBICOM):32–43. Cricket home page: http://cricket.csail.mit.edu [accessed 2010 Feb 1]
19. The SMART website: http://smart.csail.mit.edu [accessed 2010 Feb 1]
20. Sonitor® home page. Sonitor® Technologies Inc.; c2007 http://www.sonitor.com/[accessed 2010 Feb 1]
21. Tanabe P, Travers D, Gilboy N, Rosenau A, Sierzega G, Rupp V et al (2005) Refining emergency severity index triage criteria. Acad Emerg Med 12(6):497–501
22. Vinterbo S, Pino EJ, Curtis DW, Boxwala A, Ohno-Machado L (2009) An ultrasound based patient position tracking system, in preparation
23. Wuerz R, Milne L, Eitel D, Travers D, Gilboy N (2000) Reliability and validity of a new five-level triage instrument. Acad Emerg Med. 7(3):236–242

Editors

Elena Gaura received her BSc and MSc in Electrical Engineering from the Technical University of Cluj Napoca, Romania in 1989 and 1991, respectively, and her PhD from Coventry University, UK, in 2000. By the time her PhD was awarded, she was serving as a Senior Lecturer in Computer Science at Coventry University. She was further appointed as inaugural director of the Coventry University's Cogent Computing Applied Research Centre in 2006, position she continues to hold. She was awarded a chair in Pervasive Computing by Coventry University in 2009.

Over the course of her career, Elena has accrued a sturdy academic reputation in the area of smart sensing systems in general and wireless sensor networks (WSNs) in particular. She is an active disseminator of research both to the academic community and the industry, and is a frequent organiser of Networked Sensing events. Elena is a member of several sensing and microsystems advisory bodies, including the EPSRC College of Peers, UK, and chair of the UK Wireless Intelligent Sensing Interest Group (WiSIG).

Presently her research is with the development of deployable Wireless Sensor Networks for real-life applications with a focus on system design and integration, closed loop actuation and integration of decision engines within poorly resourced WSN systems. Whilst driving towards the ultimate aim of designing autonomous systems capable of large scale field sensing, her research work is heavily biased towards resolving well specified industrial problems using WSNs.

Michael Allen received the BSc (Hons. 1st class) and PhD in Computer Science from Coventry University in 2005 and 2009 respectively. His PhD work investigated acoustic localization in real-life WSN applications, and was advised by Professor Elena Gaura. Throughout 2007, he was a visiting researcher at the Center for Embedded Networked Sensing (CENS) at UCLA, under the guidance of Professor Deborah Estrin. During this time he worked with WSN systems for acoustic sensing, specifically collaborating on the WaveScope and VoxNet projects with Dr Lewis Girod at MIT. He is currently a postdoctoral associate at the Singapore-MIT Alliance for Research and Technology (SMART) where he collaborates on a WSN-based research project for real-time monitoring of the water distribution system in downtown Singapore (WaterWiSe@sg). His research interests include the development of high data-rate wireless sensing systems and the on-line data processing algorithms they use, as well as source and self localization algorithms in WSNs.

Lewis Girod is a Research Scientist at the MIT Computer Science and Artificial Intelligence Laboratory. He has been working in the area of wireless and embedded networked sensing since 1998. He joined CSAIL in 2006 after completing his PhD at UCLA in the Center for Embedded Networked Sensing (2005, advisor Professor Deborah Estrin). He has numerous academic publications in wireless sensor networks and broad experience implementing vertical networked sensing applications in academia and industry. He also holds a BS in Mathematics (1994) and and MEng in EECS from MIT (1995).

James Brusey is a Senior Research Fellow at the Cogent Computing Applied Research Centre at Coventry University. He received a BSc degree in Computer Science and a PhD from RMIT University (Melbourne, Australia) in 1996, and 2003, respectively. His PhD thesis entitled Learning Behaviours for Robot Soccer won the Australian Computer Science Association Award for best PhD thesis in 2004. Prior to coming to Coventry, he spent four years as a Senior Research Fellow in the Institute for Manufacturing, a division of the Cambridge University Engineering Department. Funding during this period included EU Framework 6 project, Cambridge-MIT Institute funding, and funding from the Auto-ID Centre. His current research interests include: Bayesian approaches to state estimation for Wireless Sensor Networks, middleware and design patterns for WSNs, and sensor node placement optimisation.

Geoffrey Challen (né Werner-Allen) is a Ph.D. Candidate in Computer Science at the Harvard University School of Engineering and Applied Sciences, advised by Matt Welsh. His research addresses the systems and networking challenges necessary to enable high-fidelity sensing applications, focusing specifically on maximizing the usage of the limited resources available to sensor network nodes. Working with geoscientists, he has helped perform three sensor network deployments on active Ecuadorean volcanoes. He built and maintains MoteLab, a wireless sensor network testbed used by researchers worldwide. Geoffrey is a 2009 Siebel Fellow, and a Resident Tutor at Eliot House.

Index