DATA MANAGEMENT IN ENVIRONMENTAL MONITORING SENSOR NETWORKS

by
Jayant Gupchup

A dissertation submitted to the Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy.

Baltimore, Maryland
February 2012

Abstract

Data gathered from multi-month to multi-year battery-powered environmental monitoring sensor networks present numerous challenges. This thesis explores three problems. First, design issues related to loading, storing and data integrity are studied in detail. An end-to-end system addressing these tasks by decoupling deployment-specific and deployment-independent phases is presented. This solution places a strong emphasis on the ability to trace the origins of every collected measurement for provenance and scientific reproducibility. Second, we explore the problem of assigning accurate global timestamps to the measurements collected using the motes' local clocks. In deployments lacking a persistent gateway, a data-driven approach is employed to assign timestamps to within 10 parts per million. Building on these experiences, we developed a novel low-power approach to accurately timestamp measurements in the presence of random, frequent mote reboots. The system is tested in simulation and on a real deployment in a Brazilian rain forest. It is able to achieve an accuracy in the order of seconds for more than 99% of measurements even when a global clock source is missing for days to months. Lastly, this thesis explores a generic data-driven approach to reduce communication costs in order to increase network lifetime. In particular, spatial correlation among sampling stations is leveraged to adaptively retrieve data from a small subset of informative sensors rather than all instrumented locations. Soil temperature data collected every half hour for four months from 50 locations is used to evaluate this system. The method achieves a factor of two reduction in collected data with a median error of $0.06°C$ and 95th percentile error of $0.325°C$.

This work is part of the Life Under Your Feet project developed at the Hopkins Inter-Networking Research (HiNRG) and eScience Labs at the Johns Hopkins University. At the time of writing, the data collected for this project is available at http://www.lifeunderyourfeet.org/

Dr. Andreas Terzis *Advisor*	Associate Professor Department of Computer Science The Johns Hopkins University
Dr. Alex Szalay *Primary reader*	Professor Department of Physics and Astronomy and Computer Science The Johns Hopkins University
Dr. Carey Priebe *Secondary reader*	Professor Department of Applied Mathematics and Statistics The Johns Hopkins University

To my grandmother,

who battled Alzheimer's for many years and came out on top.

Her serenity and positive attitude will always be a tremendous source of inspiration.

Acknowledgements

First and foremost, I would like pay my gratitude to my academic advisors: Prof. Alex Szalay and Dr. Andreas Terzis. I appreciate all the opportunities I have got at JHU - none of it would have been possible without their constant support, guidance and timely (and well-deserved) kicks in the rear.

It has been an absolute privilege to work with Alex and Andreas. It is not easy to describe Alex but I read a quote that summed up Alex pretty well. "There are dreamers and there are doers, and then there a few special individuals that have the rare gift of being both" - anonymous. His breadth of knowledge and ability to do a good job at whatever he takes up never ceases to amaze me. Andreas' ability for long-term vision is unparalleled to anything I have seen before. He has taught me to question and evaluate the motivation behind taking up any task and to always keep the end in sight. He has been a great source of support even when my chips were down. I feel really honored to have worked with these two gifted individuals and I cherish their advise tremendously.

Working with Dr. Kathy Szlavecz was an enriching experience. Her questions gave Life Under Your Feet some real sense of direction and purpose. I never thought I would be working with earthworms or turtles, but learning about these curious creatures made me look really cool at many graduate parties. On a serious note, I am grateful for having had the opportunity to interact with her. She provided me with interesting and challenging problems to work on and it made me sit back and appreciate the difference in perspectives among different disciplines.

Next, I would like to thank Prof. Carey Priebe and Dr. Randal Burns for all their time and mentorship. I've always enjoyed my open and honest discussions with Carey. He served as my guide and mentor for my masters degree in Applied Mathematics and Statistics, and he

took every opportunity he got to tease me about the misfortunes of the Indian cricket team - I enjoyed meeting him after India won the 2011 World Cup. Randal has a very unique style of teaching and conveying ideas. I took his “Transaction Processing System” course in my first semester here at Hopkins and I absolutely loved it - leaving no doubt in my mind that I had arrived at the right place.

Working with Razvan Musaloiu-E. and Doug Carlson on the Sundial and Phoenix projects were the most enjoyable periods during my Ph.D. Razvan’s enthusiasm and purity is truly refreshing. I was lucky to have worked with him. Not only was he a great partner to work with, he was also a great mentor and a really good friend. He would start all his emails with “Hi!” and now I have that habit! Doug, from whom I picked up “howdy”, has tremendous team spirit. In many ways, he taught me how to work in a team, and how much fun that can be. I admire his attention to detail and his ability to balance schedules (and find the time to bake bread!)

If you have gone through the Ph.D. process and broken through to the other side, you would know that this journey isn’t really possible without the help and support of many individuals. I’d like to acknowledge these special people and apologize for all the times I have made them play agony aunt.

- *My parents (Ajit and Sandhya Gupchup)*, for being a great support system
- All the members of *Life Under Your Feet*, *HiNRG* and *IDIES*
- *Abhishek Rege, Prashant Mali, Dheeraj Singaraju, Kiran Vanaja, Vivek Thampy, Rishab Shyam, Tushar Rane, Nupura Bhise, Amritha Ramakrishnan and Radha Mukherjee* for ensuring there is never a dull moment whilst I’m outside lab and for engaging me in many pointless discussions at One World cafe.
- *Farzeena Lakdawala, Ashwini Doshi, Huzefa Rangwala and Anshuman Shankar* for

being a friend whenever I needed one.

- *Cathy Thornton*, for always reminding me of the graduation administrative procedures and helping me negotiate them. Cathy, what would I have done without you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Understanding environmental processes requires collection of data at relevant spatial and temporal resolutions. To study the environment at a macroscopic level, scientists require access to data at high spatial as well as temporal resolutions to characterize and understand the variability present in their environment of interest. Traditionally, environmental scientists and ecologists have relied on the use of hand-held devices to collect such data. Measurements obtained from these devices are often recorded on paper using field books. This methodology of collecting data results in numerous problems and roadblocks. Technology referred to as wireless sensor networks (WSNs) has allowed scientists to collect data in real-time at scales that are relevant. However, numerous challenges related to the management of the data collected by these network need to be addressed. In this thesis, we introduce some of the key challenges and present an end-to-end solution to address them. We begin by understanding what WSNs bring to the table and why the study of this research topic is of interest to us.

Hand-held devices require scientists and field technicians to be physically present on-site. This results in increased labor costs and difficulties in collecting data during unfavorable weather conditions and increased logistical costs for remote sites. Not only is this methodology invasive, but the amount of data gathered from such devices remains limited and it is unable to provide a view that is necessary for detailed scientific studies. Data collected in this

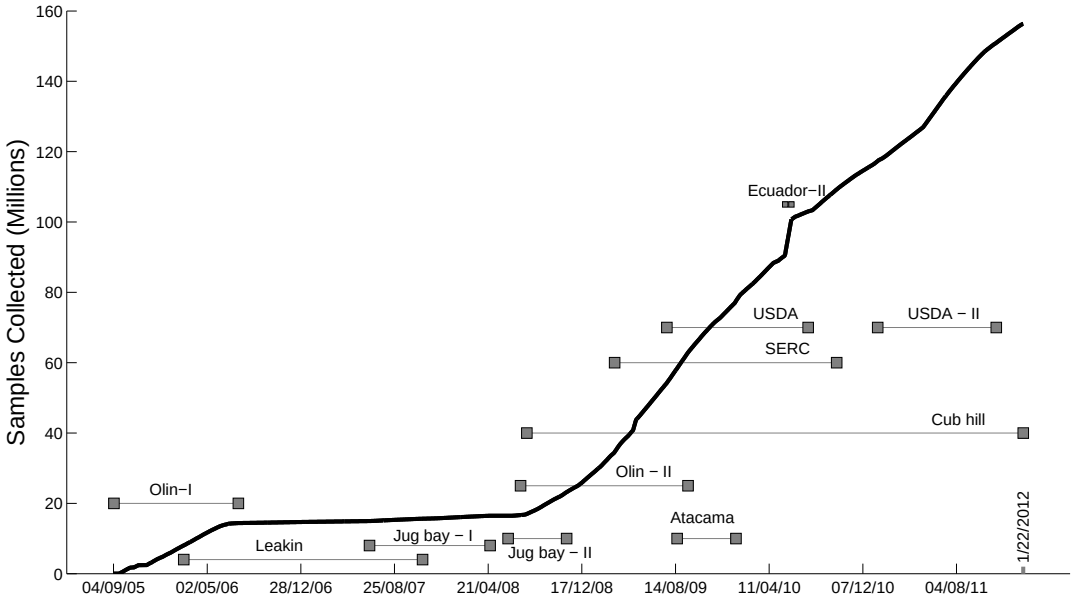


Figure 1.1: The amount of data collected by the Life Under Your Feet networks.

manner is typically stored in discrete but really separate spreadsheets that are distributed across the desktops of researchers. This methodology of storing data results in numerous versioning and provenance challenges.

To overcome the data collection obstacles, a number of environmental researchers began to employ a technology referred to as wireless sensor networks (WSNs) [38, 64, 69]. These early efforts introduced the community to some fundamental system-level challenges related to packaging, sensor failures, clock skews and network failures. In 2006, I had the opportunity of being part of Life Under Your Feet (LUYF) [67], a project aimed at studying long-term spatial and temporal heterogeneity related to the soil using WSNs. LUYF has been active since October 2005. The SensorScope project [5] and the GreenOrbs project [28] are two other prominent examples of long-term environmental monitoring deployments that are similar in scale to LUYF.

As the duration of the deployments and number of deployed sensors began to increase,

the amount of data being collected by WSNs started growing at a rate that was previously considered unimaginable within the environmental sciences community. For example, the amount of data gathered by the LUYF project since its inception is shown in Figure 1.1. This data avalanche brings with it numerous challenges with respect to the management, integrity and quality of the data. The purpose of this work is to address these challenges and deliver scientifically usable data to the LUYF ecologists. Specifically, this thesis addresses the following set of challenges:

I. Design of an end-to-end system to address the following challenges related to WSN data:

   a. Ingest data gathered by the sensor network in an incremental fashion;

   b. Track hardware and assets for provenance and sensor calibrations;

   c. Modularize processing steps to track and update data transformations;

   d. Interactive user access to the archived data;

   e. Ability to fuse with other data sets (E.g. weather stations);

II. Ensure integrity of timestamps assigned to the collected measurements.

III. Leverage spatiotemporal correlations to:

   a. Deal with sensor faults and missing observations.

   b. Improve the data collection subsystem by exploring the collection-accuracy tradeoff.

The remainder of this thesis is structured as follows. The high level goals of long term WSNs are outlined in Chapter 2. The end-to-end data management system and design motivated by the high level goals is described in Chapter 3. Challenges related to assigning timestamps to the collected measurements are introduced in Chapter 4. The methods proposed by us to tackle these challenges are presented in Chapters 4 and 5. In Chapter 6,

we describe how we can exploit the correlations exhibited in the collected measurements to improve components of our data pipeline. In particular, we use these correlations to detect sensor faults and study ways to collect data in a more informative fashion. Finally, in Chapter 7, we conclude.

Chapter 2

Environmental Monitoring

The overarching idea of using WSNs in the context of environmental monitoring applications is to present scientists with a tool to collect data. One of the fundamental design principles of WSN is to utilize limited power and monetary budgets. The goal of the LUYF project is to apply this technology to study various aspects of soil ecology at scientifically meaningful scales.

In this chapter, I will use soil ecology as an example to outline some of the challenges in typical environmental monitoring studies. This will serve as a platform for outlining the scientific requirements in typical data collection scenarios. Based on these requirements, we present an overview of the various components and the engineering challenges involved in order to meet these goals. I would like to point out here that although soil ecology is used as the application domain, the ideas and concepts remain fairly general and apply to other environmental monitoring applications too. To give a concrete example, most of the technology developed for the LUYF project was utilized to collect data for five weeks in 2009 at a Brazilian rainforest for a completely different purpose - to understand the temperature and moisture conditions at different layers of the forest canopy.

2.1 A Macroscope

Soil is the most complex layer of the terrestrial ecosystem and it has been referred to as the "final frontier" by the scientific community [42]. Soil is an active reservoir of both carbon dioxide (CO_2) and water and it plays a pivotal role in the global biogeochemical cycle. Moreover, the presence of active microorganisms and their activities in the soil impact various physical and chemical properties resulting in tremendous heterogeneity and variation. Processes related to the soil vary spatially, temporally and with factors such as soil and vegetation type. Thus, in order to get a complete picture, one needs to collect small scale as well as large scale data continuously with respect to time and space. This new paradigm of collecting environmental data has been termed as a "macroscope" in WSN literature [69].

The most common method of obtaining data in the past has been to use hand-held devices by physically going to the field and recording the measurements in a log-book. Data collected in this manner is extremely sparse and difficult to collect under harsh conditions without disturbing the environment. The use of manual labor makes this method particularly restrictive, costly and ineffective for achieving the necessary spatiotemporal coverage. Another commonly employed technique is the use of data-loggers. The main disadvantage with commercially available data-loggers is that they do not provide scientists with a "real-time" view of the environment being sensed. Typically, one needs to physically connect a data-logger to a reader in a laboratory environment to read the data stored on the logger.

Hopefully, by now I have convinced you that what soil ecologists really require is a way of sampling the environment at various levels of detail. Furthermore, if this data is available to them in real-time, it would enable them to coordinate and plan their scientific experiments in a more productive manner. For example, Lijun Xia, our resident earthworm researcher is interested in understanding the distribution of earthworms in an urban forest under high

moisture conditions. If Lijun has the ability to remotely monitor the moisture conditions at her site of interest, then she could use this information to plan her experiment appropriately.

2.2 System Requirements

Having introduced some of the basic concepts and methodologies related to environmental monitoring, let us understand the general requirements for setting up and maintaining a continuous monitoring network. These requirements are as follows:

1. **Data Collection:** Sampling locations, modalities (or sensors), sampling frequencies and deployment durations are selected by environmental scientists. These values are largely driven by scientific questions of interest. For example, a typical soil ecology experiment might need two depths of soil temperature and soil moisture that require to be sampled every 10-15 minutes for a period of a year. The size of the deployment depends on the required spatial granularity and the available resources. In general, sampling locations are selected so as to capture the heterogeneity of the underlying soil and vegetation type.

2. **Data Delivery:** The data collected from the sampling locations are expected to be delivered to scientists with a latency in the order of hours to days, but not weeks. The major engineering challenge lies in the fact that these quality guarantees need to be met in the presence of hard power constraints and limitations of radio connectivity. The data needs to be delivered via a multi-hop network using the motes' on-board radio that has a limited range ($\sim$ 30 meters). Since there is no access to line power in remote locations, these motes are powered by small batteries that are expected to provide power for a period of a year or more. This problem warrants an ultra low-power solution that not only meets the data-delivery requirements of scientists, but also maximizes the network

lifetime simultaneously.

3. **Data Management:** WSNs generate vast quantities of data that were previously considered unimaginable in the environmental sciences domain. For example, a 50 node deployment that collects data every minute from 10 different sensing modalities will produce over 26 million records over a period of a year. The data collected by these networks should be easily accessible to scientists at various levels of detail with low latency. Another major challenge with sensor data is provenance. A number of transformation steps are necessary to convert raw sensor values in to final usable data. For scientific reasons, intermediate results need to be stored allowing the origins of every piece of data to be traced. This motivates the need for a unified system that manages the data gathered by WSNs and also stores various intermediate results. Orthogonally, data managers also need to track the health and status of various hardware components and this forms an integral part of the data management system.

4. **Data Integrity:** Scientific understanding of processes is only as good as the data used to study them. Therefore, it is imperative to assess the quality of the data gathered, by monitoring networks before presenting them to the scientists. In WSNs, assigning accurate timestamps to the collected measurements is a well-known problem due to a variety of reasons. Environmental studies require measurements to be timestamped to an accuracy of seconds to milliseconds. Apart from ensuring temporal integrity, the data gathered by such network contains a lot of missing values and are inherently noisy. Thus, whenever applicable, it is crucial to apply methods that are robust and can deal with data of this nature to ensure that the scientists are provided with high quality data.

2.3 Contributions

The main goals of this thesis are to study, understand and present solutions to various challenges related to the management and integrity of the data collected by environmental WSNs. The thesis titled "Low-power Wireless Sensor Networks for Environmental Monitoring" by Răzvan Musăloiu-E. addresses the challenges associated with data delivery in long-term WSNs.

The overall architecture of the data organization and loading system is presented in Chapter 3. This system design was motivated by some of the lessons learned from three early deployments. Based on these experiences, a two-phase data pipeline was engineered that serves as the backbone for the visualization and the data access layers; the two layers scientists interact with. We present a framework that supports the data and scientific data processing that goes along with the data. This modularity makes such a design very attractive as one can easily upgrade, improve existing algorithms and produce new and higher quality versions of the data.

In Chapter 3, based on our experiences from deployments in 2005 and 2006, we provide some of the design philosophies behind loading and organizing sensor data. Using some on these lessons, we outline our current data loading schema and elaborate on how it supports data and metadata from multiple deployments. we describe how the data pipeline serves as the backbone for the visualization and data access layers.

Achieving temporal integrity is one of the main tasks of the data processing pipeline. Broadly speaking this refers to the process of assigning and validating timestamps for the collected measurements. This thesis proposes Sundial and Phoenix - two novel energy efficient algorithms that assign global timestamps to the measurements collected using the motes' local clocks. The details of Sundial and Phoenix can be found in Chapter 4 and Chap-

ter 5 respectively. This work is different from most of the other bodies of work in the literature as it is a post-facto approach rather than the commonly adopted online one. This offline nature of processing greatly simplifies the code that needs to run on the mote. It also leads to significant energy savings in comparison to online timestamping solutions.

Data gathered from WSN's exhibit strong spatiotemporal correlations. As such, they present us with opportunities to take a data-driven approach to improve and optimize aspects of the overall system. In Chapter 6, we look at two applications that leverage these correlations. First, a simple and effective methodology to detect sensor faults is applied to soil temperature data gathered from the Cub hill deployment. Second, we explore a data-driven system to collect data adaptively in order to strike a balance between reducing power consumption and maintaining data fidelity. This work builds on some recent results obtained in the area of finding informative locations using mutual information. Specifically, we will discuss a system that uses these results and explores strategies to reduce communication costs by selectively transmitting measurements whilst ensuring low prediction errors for measurements that are suppressed. This system is designed to work in conjunction with the existing LUYF architecture and infrastructure.

Chapter 3

System Design

When the first LUYF pilot project was underway, there were few or no guidelines for designing an end-to-end data processing system for environmental WSNs. The reality was that by the early part of the decade (2000-2010) a lot of the teething problems had not yet been completely dealt with. Therefore, most of the efforts were focussed on the core issues of hardware and packaging [31, 38], wireless characterization and packet losses [12, 77], routing protocols [49, 63], and time synchronization [22, 40]. Once solutions for these challenges began to emerge, the idea of using WSNs for monitoring remote environments started gaining considerable momentum, and by the middle part of the decade, a number of applications and pilot studies [69] were showcased. In the fall of 2005, the first LUYF pilot project began with a goal to test the feasibility of using sensor networks for year-long soil ecology studies. Not surprisingly, much of our initial effort also focussed on waterproofing, dealing with sensor failures, ensuring reliable data downloads, and mundane TinyOS [29] and nesC [26] programming tasks.

As our systems, processes and requirements matured, the data processing pipeline underwent several design changes. These modifications have been a result of varying deployment conditions accompanied by the changing needs of the domain scientists. The pipeline is heavily motivated by field deployments. We will discuss various deployments and the lessons

learned in Section 3.1. The two phase design is developed based on these experiences, and is discussed at a high level in Section 3.2. In Section 3.3, we drill down into the details of this design. Section 3.4 details the methodology used to monitor the health of the system. The scientists interact with this system by extracting and visualizing the data using Grazor, a web-based tool described in Section 3.5. An account of the challenges and shortcomings of this system is provided in Section 3.6, and in Section 3.7, we discuss directions that will help us to address these shortcomings.

3.1 Deployments

Field deployments bring together professionals from various disciplines and backgrounds. A complete list of the LUYF deployments is given in Table 3.1. These deployments have covered a wide spectrum of technical and scientific objectives. The initial deployments served as pilot studies in an effort to validate our hardware and software platform. After exhaustive testing of our platform and learning from the pilot studies, we changed our download and data loading architecture - the two phase architecture. A number of long-term deployments were supported by this architecture, each having a highly focussed scientific objective. As our system matured, scientists became interested in taking this technology to collect data at a high rate (frequency) in remote areas during their field trips (campaigns). These campaigns shed light on some new requirements and challenges that arise in the absence of a persistent basestation and connection to the database. These challenges are outlined in detail in Section 3.1.3.

The lessons learnt from each deployment were of great value and each deployment helped us to improve and prepare for the next one. The impact of these deployment experiences on the data processing pipeline will be the topic of discussion in the upcoming section.

3.1.1 Pilot Deployments

At the time of writing of this thesis, there is an abundance of "lessons learned" papers in the sensor network deployment literature [34, 64, 69, 73]. Barrenetxea et. al cover a wide spectrum of issues in their article titled "The Hitchhiker's Guide To Successful Wireless Sensor Deployments" [4]. This article, published in 2008, is a good starting point to understand the common pitfalls in deployments. I would strongly encourage researchers to read this article before they go ahead with their first deployment. Prior to 2005, research groups primarily focussed on short-term deployments lasting around a month [31, 38]. There were few or no accounts of deployments that were operated for multiple months so a lot of the lessons had to be learnt first-hand.

In September 2005, the first set of MicaZ motes [17] were deployed in an urban forest near Olin hall of the Johns Hopkins Campus. The second deployment, called Leakin park, was also deployed in an urban forest co-located with Baltimore Ecosystem Study's (BES, [62]) sensors. In both cases, we were interested in understanding the robustness of the MicaZ hardware and software platform, and the quality of the collected data. Soil temperature, soil moisture and light information was collected with an interval of one minute at Olin and 20 minutes at Leakin Park. These deployments lacked a persistent basestation and the motes were acting as glorified data loggers. Field trips were made (typically, once every 3 months) to download the data using an ad-hoc basestation. The data was collected during this time using a one-hop wireless link. This data was preprocessed and then loaded into a database for analysis. The next two deployments were located at a wetland sanctuary known as Jugbay [6]. The main difference between this study and the previous ones was the use of TelosB motes [50] instead of MicaZ.

The data collected by these deployments was loaded in a SQL database. The first design

of the schema and data pipeline was designed by Jim Gray and Alex Szalay. Their design was modelled around the skyserver project, and a lot of the stored procedures and utility functions were re-used [59]. The main contributions of the original schema were as follows:

- Established a framework for representing metadata - data related to the hardware components used in the field.
- Incorporated procedures to convert from raw sensor data (ADC) to physical values.
- Formulated a methodology to re-sample the data based on the scientific goals.

For more details of the data processing pipeline used for the pilot deployments, I would urge the readers to refer to [67].

Lesson Learned

The length of the pilot deployments were longer than most sensor deployments up until that point. Although a number of lessons were learnt fairly quickly into the deployments, a number of pitfalls became evident only after a few months had passed. It would be difficult to provide an exhaustive list of our observations so I will highlight the lessons that helped us in designing the data processing pipeline. A summary of the lessons learnt are below:

- Inability to monitor the network regularly resulted in loss of data ;
- Lack of a persistent basestation and mote reboots posed problems in assigning accurate global timestamps ;
- Storing meta information (flash address, reboot counter) for records written to the mote flash is helpful ;
- The metadata (moteid's, sensorid's etc) should be flexible enough to represent hardware and location reconfigurations ;

- The system needs to be able to redo all the computations from scratch resulting in a delete-nothing philosophy.

When the sensing hardware is exposed to harsh conditions, failures are to be expected. The most common modes have been battery failures and failures due to excessive moisture in the mote enclosures. Without a persistent basestation, monitoring the network becomes challenging and we are unable to detect failures remotely. For example, in the Olin deployment two motes stopped collecting data two months before the scheduled end of the deployment. Such failures result in invaluable loss of data and often compromise the scientific experiment.

Measurements are timestamped using the motes' local clocks. The hardware does not support an on-board, battery-backed real time clock. Thus, whenever the mote resets, it loses its clock state and the local clock value restarts from zero. These measurements are assigned a global timestamp outside the network by obtaining a linear mapping between the motes' local clock to the global time base. This mapping is obtained by collecting reference points, and each reference point contains two values - the motes local clock value and the basestation's global clock value. Linear regression is then used to fit the equation of a straight line to these points giving us the the mapping (skew and offset) of interest. This methodology works as long as the mote does not reboot. In reality, they reboot due to a variety of reasons (low battery levels, high moisture, software bugs etc). When a mote reboots a new mapping needs to be obtained and if they reboot at a rate higher than the basestation contacts, it becomes difficult to reconstruct the measurement timeline in terms of the global time. This problem was observed in the Leakin park deployment. Motes rebooted (often multiple times) in between the basestation contacts and reconstructing the timestamps became a major challenge. We overcame this problem using a data-driven methodology known as *Sundial* (Described in detail in Chapter 4). Although data-driven approaches provide us with a mechanism to

achieve temporal integrity, they are a reactive solution and not a proactive solution for accurate timestamping. Going forward, we emphasised on designing solutions that provide us accurate timestamping in the presence of network partitions and random mote reboots.

Environmental sensors are connected to motes. Data recorded by these sensors is stored on the motes local flash. At a given time instant, a mote may store data from one or more sensors in its flash. As mentioned previously, this data is time stamped using the motes' local clock. Typically, all sensors are activated at the same time and these measurements result in a tuple (or a record). These records are temporarily stored on the motes' flash and we refer to them as *FlashRecords*. In the absence of being able to reconstruct timestamps entirely, storing the reboot counter and address of the record on the flash helps us to establish an ordering of the records in terms of the logical clock. The key take away message is that even if records are unable to be assigned accurate timestamps, a sequential view of the data is valuable so it is important to store meta information about the records.

In typical academic settings (such as the one we are in), the budget for hardware is limited. Thus, when a mote fails or gets waterlogged, it may be brought back from the field to be refurbished. The same mote may be used to replace another malfunctioning mote. In this schema, data streams were identified by their mote ID. This schema would essentially represent data from two different locations in the same data stream. From a scientist's perspective, data from the same location and sensor type should represent one sensor stream regardless of the hardware being used. The main lesson learned here was that the metadata needs to be flexible enough to deal with hardware replacements and representation of data should be done in a hardware-agnostic manner.

For the vast majority of the times, scientists are interested in data in the final form - timestamped physical values. Every now and then, the data will demonstrate some interesting features and it is hard to tell if it is an artifact of the data processing or due to real

phenomenon. For instance, one of the soil temperature sensors at Jug bay demonstrated a sudden jump in its readings. When we first saw this, we were not sure if it was a real phenomenon or an artifact of the data processing. We traced it back to the ADC values and found that even the ADC values showed this jump. After doing some tests we realized that this was due to a sensor fault and not because of faulty data processing. However, we learned that there are times when we need to check every single step and trace the observation back to the acquisition step to ensure the validity of the data.

3.1.2 Two Phase Architecture Deployments

Once we began to look through the lessons learnt, it became clear that we needed to redesign our existing pipeline. We looked at various steps involved in data processing and realized that the final representation of the data is identical to data gathered by other environmental monitoring sources (other deployments, weather stations etc.). However, a sensor network acquires data in a manner that is very different from a weather station. Moreover, the data acquisition process of one network may vary from that of another. Therefore processing data from deployments can be thought of as two sets of tasks - one is deployment specific and the other is deployment independent

The deployment specific component performs tasks that are unique to the deployment. For example, two deployments may employ different timestamping solutions (possibly due to varying power budgets). The process of assigning global timestamps will vary for these two deployments. On the other hand, once the measurements are timestamped and converted to their physical values, data from both deployments are similar in that they both represent a collection of time series. The data generated from sensors could be attached to TelosB motes or to a weather station - their representation does not change (deployment independent).

Before we proceed, I would like to motivate the deployment specific tasks a little more. In scientific studies, the ability to trace the origins of the data (provenance) is critical. At the lowest level, this boils down to storing and representing how the data is acquired and what sort of preprocessing has been applied before generating the final time series. The process of acquiring and treating the data highly depends on the hardware configuration. In sensor networks, there is an entire ecosystem of motes, mote software, sensors, conversion functions, data acquisition /retrieval protocols. In order for researchers to trace the origins of every measurement, all the information related to acquisition needs to be preserved and made available. This approach calls for a deployment specific phase where all the acquisition and preprocessing steps are logged and can be queried easily.

The two phase data loading architecture is composed of deployment dependent and independent phases. At this point, I wanted to introduce you to the overall philosophy. The details and experiences in using this scheme will be presented in Sections 3.2, 3.3, 3.4, 4.2. The two phase scheme addresses the following needs of the scientists:

1. Ability to download and visualize timestamped sensor data;
2. Achieve a high data yield for all the sampling locations;
3. Preserve information that allows scientists to trace the origins of each measurement;
4. Reprocess data in the event that anomalies are found;
5. Register hardware configuration changes with the system;
6. Monitor the health of the deployments;

The two phase architecture was used in six deployments. Five of these deployments were located in Baltimore and one was located in the Atacama desert in Chile. For more details on

the deployments that employed the two phase architecture, refer to Table 3.1. From a hardware perspective, the main difference between these deployments and the pilot deployments was the presence of a persistent basestation. These sites were also reasonably accessible. In the event of hardware or system failures, researchers/technicians could get to the sensing sites within an hour's drive. These characteristics facilitated a number of architectural changes on the way data was loaded and stored (as we shall see later). The main contributions of the two phase pipeline were as follows:

- Developed a framework for storing and quering data gathered from environmental sensors;
- Presented a modular and layered view of the various stages in processing sensor data that allows developers to "plug-in" and "plug-out" appropriate processing methods;
- Enabled users to trace the origins of each measurement;
- Provided data access and visualizations to users in an interactive and adaptive way;
- Improved the overall yields and the ability to monitor network health

3.1.3 Campaign Deployments

The two-phase architecture was designed for collecting data from long term deployments. In this set up, researchers could visit the sites frequently (once a week) for maintenance work. This architecture assumed that the basestation would be present at the site at all times. Once the scientists started using data gathered by sensor networks, they became interested in carrying the hardware and deploying these networks themselves at their remote study sites. These sites were completely cut off from line power, and consequently, could not support

a persistent basestation and an internet connection. There is a growing trend and interest in the community for such deployments, and they are referred to as *campaign* deployments.

The interest is to collect data at high sampling frequencies for a period of a few days to a few weeks. Technicians (or graduate students) could potentially visit the site once a day and data would be harvested onto a laptop that acted as a basestation. This data would then be analysed and hardware reconfigurations would take place based on the quality of the collected data. The goal of these hardware configurations was to utilize the hardware optimally. For instance, a few locations in the Ecuador-II deployment were recording soil CO_2 in excess of 10K ppm. Thus, the previously installed 10K sensors had to replaced with 20K ones and the 20K sensors were replaced with 10K at locations where the concentration was lower than 10K. The field visits in such campaigns tend to be highly unpredictable and it is common to have a series of days without any visits resulting in unattended network operation.

The lack of a persistent basestation, and consequently, a global clock source, forced us to redesign our timestamping solution. Phoenix was designed for a worst-case scenario involving multiple node reboots, temporary network partitions and the absence of a global clock source for days to months (Chapter 5). This situation was realized in the Brazil campaign. The plan was to use two GPS motes that would serve as the global clock source. The GPS devices were held at the airport customs and arrived three weeks after the deployment had begun - one week before the deployment ended. To make matters worse, one of them did not work. During this three week period, the motes rebooted a number of times and left and joined the network on a number of occasions resulting in the worst-case scenario that we were hoping against. Phoenix was able to timestamp more than 99% of the measurements collected during the entire deployment.

At first glance, it appeared as though the rest of the pipeline could be reused for these

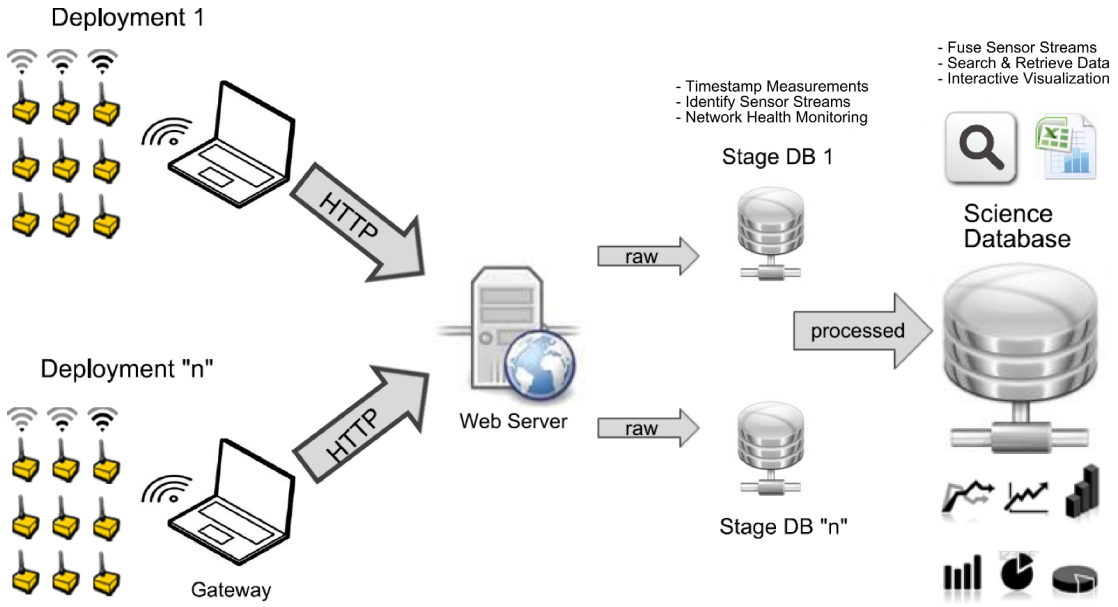


Figure 3.1: The overall architecture of the LUYF data processing pipeline.

deployments. As the deployments started taking shape, the shortcomings of the existing architecture started becoming clear to us. These campaigns have provided us with a number of lessons that has forced us to take notice and redesign for future deployments. The exact nature of these shortcomings will become clear in Section 4.2. Before we go any further, I would like to introduce the core design concepts of the two-phase data pipeline and discuss its various components.

3.2 Two Phase Loading Architecture

The stages involved in the system are shown in Figure 3.1. At one end of the pipeline data is stored on the motes, and at the other end, it is stored in a database accessible to the scientists. We describe this data movement at a high level in this section.

Recall that the sensor measurements are stored on the mote's flash and we refer to them as FlashRecords. In addition to FlashRecords, a mote stores status records (JournalRecords)

and time states of other motes (AnchorRecords). A basestation requests the mote to transmit all these outstanding records using an ultra low power mechanism known as Koala [44]. The basestation identifies where its cache ends and where the mote's current data ends, and requests the difference. The gateway stores this data locally, and in turn, pushes (using HTTP) this data to synchronize with a remote server. It is the responsibility of the basestation to push records in order. For example, if the basestation has records 1,2,5 and 6 available, it will only push 1 and 2 at this time. When it queries the database for its status next, the response will be 2. Until the basestation has records 3 and 4 available, it will not push 5 and 6. In short, the pushes happen in order. An application running on the web server parses these records and pushes it to a *stage database* created for that deployment. One stage database is created for every deployment. I would like to point out that the interaction between the basestation and the motes is completely autonomous from the interaction of the basestation and the remote data server.

Our current hardware assumes that many different sensors can be connected to a node. Sensors are referenced by their sensor ID – a combination of the sensor's hardware ID and deployment date. Nodes are referenced by a node ID – a combination of the box ID (or mote ID) and deployment date. Every unique location, referenced by the location ID, hosts a node to which a sensor is attached at one of four available channels. The TelosB platform has on-board sensors (temperature, humidity and total solar radiation, photo active radiation, battery voltage and stabilized voltage) and they occupy sensing channels five to ten. These internal sensors typically measure the conditions within the water-tight box. The location-node-sensor arrangement is recorded during the time of deployment by scientists (or field technicians) on log-books and subsequently entered in to the system using a web-based portal. This methodology allows us to represent a box servicing different locations at non overlapping time intervals. In practise, this happens a lot as boxes need to be replaced due

to hardware failures. We refer to this hardware configuration information as *metadata* and it is required for converting the sensor's raw measurements to physical values (in addition to provenance purposes). The stage database is unable to process information in the absence of this information. The tasks performed by the stage database are as follows:

- Order records received from the basestation;
- Assign a global time stamp to the sensor measurements by a process of post-mortem reconstruction of timestamps;
- Parse the records to extract the sensor streams
- Convert the raw sensor measurements to physical values.

The status records collected by the mote are used to generate periodic automated health reports. These reports are used to replace malfunctioning sensors and troubleshoot networking in order to ensure that our collaborating scientists obtain high data yields.

The stage database next pushes the converted values to a unified database known as the *science database*. The science database contains and hosts data from all the LUYF deployments and it performs the following tasks:

- Re-sample data based on the scientific goals ;
- Store data at various levels of detail (hourly, weekly, daily) ;
- Detect and flag measurements that appear to be faulty ;
- Expose data to the scientists in the form of visualizations and text files.

3.3 System Components

In this section, we will take a look at the four main sub-systems of the data processing pipeline – the upload application, stage database processing, science database processing

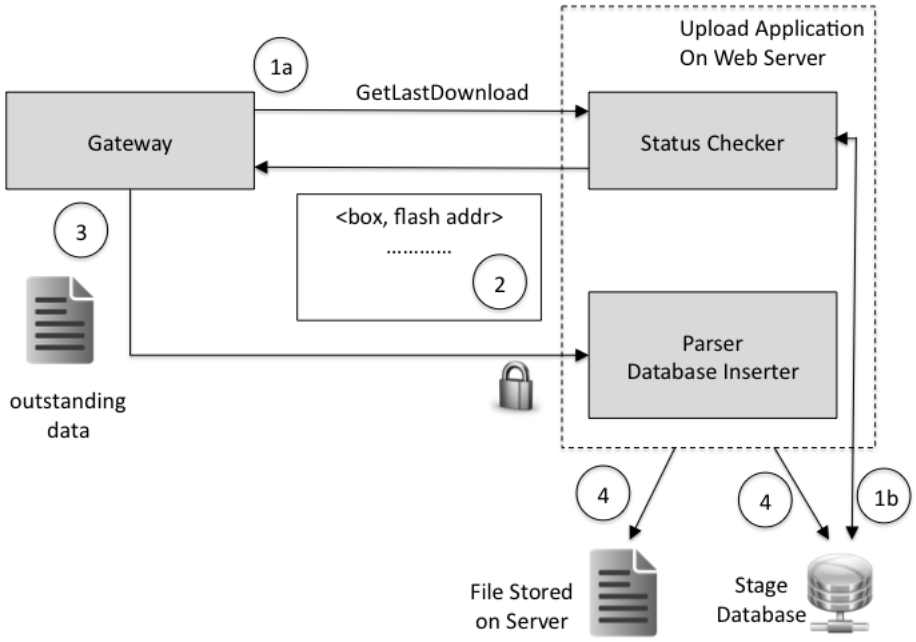


Figure 3.2: Various steps involved in transferring data from the basestation to the stage database via the web server.

and the metadata framework. The focus will be to provide a high level overview and the design philosophies of these subsystems and implementation details wherever it is necessary.

3.3.1 Upload Application

The basic mechanism for transferring the outstanding data from the gateway to the stage database is shown in Figure 3.2. This involves a process of handshaking that is described as follows. The gateway first establishes the amount of outstanding data by requesting the upload application (Step 1a in Figure 3.2) to query the stage database (Step 1b in Figure 3.2) to find out how much data is available at the basestation but not available at the database. The upload application responds by reporting the last received address on the mote's flash for each of the boxes that are known to be out in the field (Step 2 in Figure 3.2). The basestation

compares the flash address values in this table with what is available on it locally. This lets the gateway know how much data is outstanding for each mote from the database's perspective. The gateway then puts together one file containing all the outstanding data from all the boxes and sends it back to the web server (Step 3 in Figure 3.2). The uploaded data (D) is signed by hashing the contents with a private key (K) that is shared with the web server using a SHA-1 hashing function. The hash (F(D+K)) and the contents (D) are sent together. The web server recomputes the hash (F(D'+K)) over the transferred contents (D') and the private key(K). If the two hashes agree, the file is accepted (since D=D'). The upload application parses the contents to extract the records, stores the file on the remote server (can be the same as the web server), and pushes the records to the stage database (Step 4 in Figure 3.2).

The basestation uploads data as a collection of four types of records. The bulkiest of these are the actual sensor measurements or the FlashRecords. The second type of records contains information about the status of the mote. These types of records are called *JournalRecords* and they are recorded at the basestation whenever it comes in contact with a box (typically during download rounds). They help us diagnose the health of the motes. Each mote periodically broadcasts its current time state. Other motes listen for these broadcasts and store this information in the flash along with its own current time state. These time references are called AnchorRecords and they enable translation of one mote's time state to that of another. Each box also collects information (LQI, RSSI) about its links to other boxes and this information is harvested by the basestation. Additionally, the basestation also records the path selected to download data from the boxes. These types are records are called *NetworkRecords*.

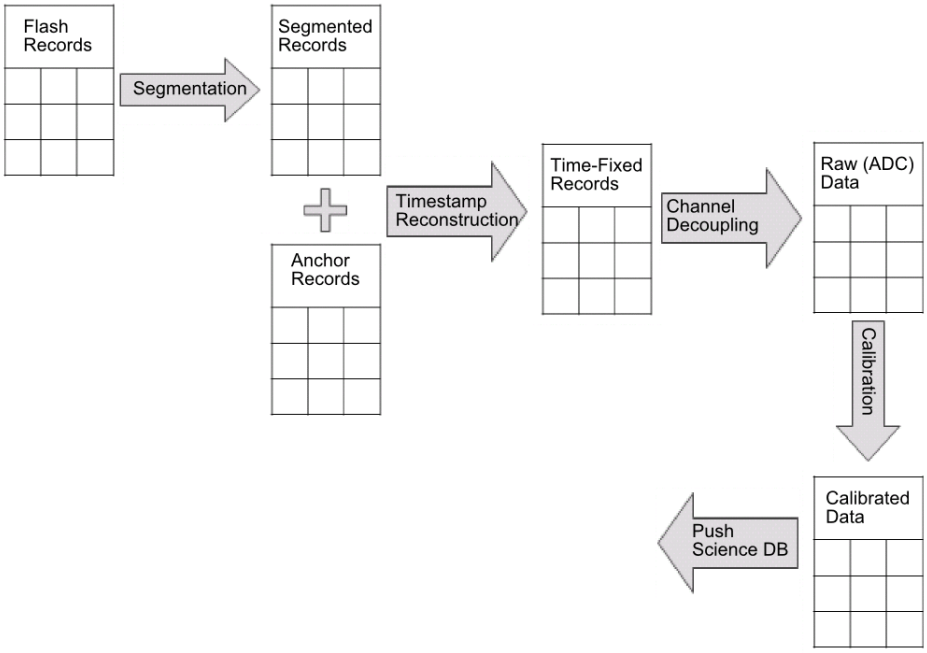


Figure 3.3: Various processing steps involved in the stage database

3.3.2 Stage Database

The high level roles of the stage database were introduced in Section 3.2. Here, I will go into the details of these roles. The processing steps that take place in the stage database are shown in Figure 3.3. The complete stage database schema is shown in Figure A.2

Segmentation

A segment is defined as a collection of records (of a given mote) where the local clock is monotonically increasing. We say that a new segment is started when a mote resets (thereby reseting its logical clock). Upon resetting, the mote increments its reboot counter. A mote's reboot counter is also reset when a new version of the code is installed and its flash is wiped clean. Each installation is associated with an installation time that is stored on the mote. Each segment is identified by its reboot counter and installation time. Each record is iden-

tified by its local timestamp and the segment number. This process of assigning each record a segment number is referred to as segmentation. Prior to December 2009, the boxes did not record the reboot counter and installation time in the flash so these two values had to be inferred from the data itself. Hence there are two implementations of the code that identifies and assigns segments to data records. The segmentation process is done to create a logical construct that facilitates translation from the mote's logical clock state to the global time base.

Timestamp Reconstruction

Motes do not have an on-board real-time clock and hence they timestamp the measurements using their local clock. These measurements need to be translated to the universal time and this process is what we refer to as postmortem timestamp reconstruction. In order to do this, the segmented data is combined with the time-state information stored in the Journal table. This information is essentially a pair of values – first being the mote's local time, and second, the basestation's clock. These points, referred to as *anchor points* are collected whenever the basestation comes in contact with the mote. Moreover, the basestation's clock is synchronized using Network Time Protocol [43]. Using these anchor points, we can map the mote's local time base to the global time scale since they are related linearly. In practice, we need to identify the segments that each anchor point belongs to and obtain one mapping (or fit) for each segment using linear regression. This mapping involves estimating the clock skew and the motes offset from the basestation clock. Using these fit parameters, the records are translated to their corresponding global timestamps. This original methodology is extended by Phoenix (Chapter 5) to deal with network partitions and the absence of a basestation for an extended period of time.

Channel Decoupling

Sensors can be connected to a mote on any of the available sensing channels. To be more specific, the LUYF motes have 4 sensing channels to which external sensors (soil temperature, soil moisture and CO_2) can be connected in any order. Given this design, it is important to have a map between the channel numbers and the sensor types connected to those channels. This map is recorded in the lab prior to the deployment.

The time corrected records contain mote number along with the ADC (raw) values associated with the different sensing channels. The channel-map information needs to be referenced to unpack the record and identify the sensor data associated with each sensing channel. The result of this process is stored in a table (RawData) where each record contains three columns – sensor ID, timestamp and raw (ADC) value. This decoupling process creates a dependency on the metadata information (See Section 3.6.1). The pipeline will be unable to decipher the time corrected ADC streams if the associated metadata information is not registered with the system.

Physical Value Conversion

Records stored in the RawData table need to be converted to their physical values. For instance, for a soil moisture sensor, a value of 1023 in ADC corresponds to 31% humidity. A conversion function is defined and stored in the database for each sensor type. The metadata table is used to extract the sensor type of each sensor and this is used to invoke the appropriate conversion function. The results of applying the conversion function are stored in a table known as Calibrated data. This data is then pushed to the unified science database.

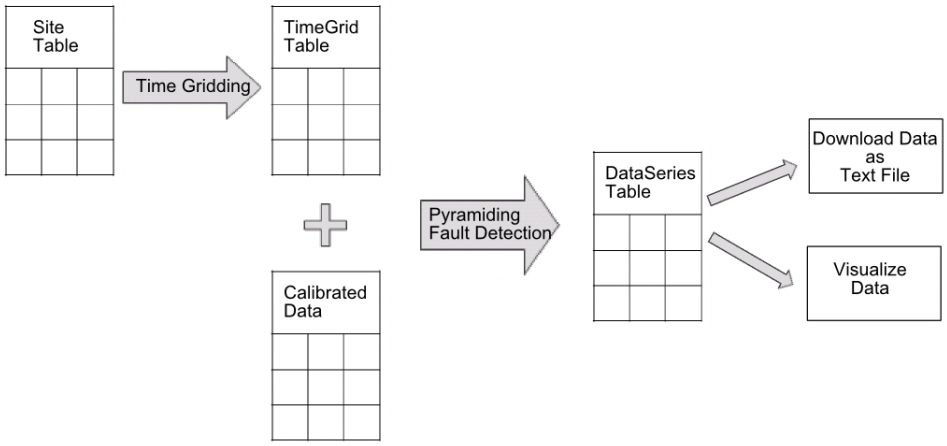


Figure 3.4: Various processing steps involved in the Science database

3.3.3 Science Database

While the stage database undertakes tasks that are tightly coupled to the architecture of a sensor network, the science database performs tasks that can be applied to environmental data streams in general. For example, data collected from weather stations can be stored in the science database and treated the same way as data collected by the LUYF sensor network. The various stages in the science database are shown in Figure 3.4.

Time Gridding

When we present data to scientists, it needs to be at scales that are relevant to the question at hand. Co-located sensor streams often collect data in an asynchronous manner. For example, a flux tower and a weather station might be co-located but they may be operated by two different institutions, and hence, their data collection rate and schedules might be completely different. Scientists, on the other hand, prefer to work with data that is aligned in time across the sensors. To elaborate on this, let us consider an example in sensor networks where time gridding comes into the picture. Motes collect data based on their independent schedules.

Let's assume the schedules of two motes collecting data every 10 minutes are as follows:

N1 : 5m,15m, 25m, ...

N2 : 3m, 13m, 23m, ...

These schedules represent minutes past the hour mark. Let us suppose that scientists want half hourly values for their purpose. To achieve this, a time grid is constructed that is aligned to the hour mark and the grid points are half an hour apart. This process of constructing such a grid is called Time Griding. In our pipeline, a SQL stored procedure takes the grid interval (step size in terms of time) from the Site table, computes these pre-defined intervals and then stores them in the TimeGrid table. Data from the CalibratedData table is interpolated around these pre-determined points. For example, let's say that we want to compute the interpolated value around time T and the step size is $2 * \delta$. We take the data points that fall in the time interval given by $[T - \delta, T + \delta]$ and we average these points. This average represents the interpolated value around time T. The results of these interpolation for all the sensors are stored in the DataSeries table.

Pyramiding

The amount of data gathered by long-term environmental monitoring projects can be considerably large. For example, LUYF has collected over a 100 million data points over the course of five years. Based on our experience in working with environmental scientists, it is important for data to be visualized in an interactive manner for them to get a good feel for the essential features in the data. The sensor data is typically visualized as a timeseries plot with time on the X-axis and the observations on the Y-axis. A period of a year corresponds to a large number of data points for data sampled every half hour, and visualizing all these points on the same timeseries plot would have a number of disadvantages. The plot becomes very busy, it consumes a lot of memory and creates a lot of work for the plotting library.

Any operation on this plot requires a lot of computation and reduces the interactivity of the visualization.

We implement a simple scheme to return data in a layered manner inspired by the way Google maps renders images. The overall goal of this scheme is to limit the number of data points being visualized without losing the prominent features of the data. We note that plotting more points than the number of available pixels makes the plot very busy and serves no real advantage. In this scheme, the time period of interest is provided as an input. The number of points (NUM) returned to the user is a variable and the default value is the screen width (in terms of pixels). This time period is then chopped up into NUM equally spaced periods to create a custom time grid. We then interpolate around these custom time points as explained previously. This scheme is particularly useful when the users are exploring the data. They can change the time period (zoom in/zoom out) and visualize information in an interactive manner .

In practice, the interpolated data is pre-computed and stored in the DataSeries at various zoom levels. This concept is inspired the image cutout application in SkyServer [59]. Once the time period is provided, it fixes the step size. Using this step value, we can retrieve data at the nearest zoom level and use that for interpolation. This minimizes the amount of data that needs to be interpolated on the fly and improves the interactivity (speed).

Fault Detection

Environmental sensors are subjected to harsh conditions which often results in short-term and long-term faults in the data. These faults can significantly affect the quality of data visualization and analysis. Sensors exhibit a strong degree of correlation in space and time and these correlations are exploited to determine the integrity of the sensor measurements. Currently the pipeline makes use of a median filtering method to identify and flag faults. The

method works as follows. At each time step corresponding to the time grid, the median of all the measurements for a given modality is computed. If the deviation of a measurement from the median is greater than a pre-determined threshold, the measurement is considered as a fault and it is flagged. The threshold is varies from one modality to another.

Data Access

Data is made available via a web-based portal called Grazor [32]. Grazor allows data to be visualized quickly using the pyramiding mechanism. Scientists typically browse around and are given the ability to bookmark and download data of interest. The format of the downloaded files is comma separated values.

3.3.4 Metadata Framework

Representation or nomenclature of sensor streams and site information is a key component for inter-deployment and intra-deployment data integrity. To the best of my knowledge, there is no commonly accepted data model for representing data collected from sensor networks. A number of research groups and consortium's are addressing the general problem of representing environmental observations. It is impossible to provide an exhaustive list of these bodies of work, but here are some of the prominent ones [7,48,57,58]. Although this literature addresses representation of environmental data, we did not find it suitable to represent metadata for our architecture. The main reason for this is their failure to capture and represent hardware reuse - the ability to track the usage of the hardware (motes, sensors) throughout the length of the project. We decided to use a very simple data model to represent metadata customized to our needs. There are three main aspects to this metadata framework:

- Representation of site and hardware information (tables);

- Associations and hierarchy among tables;
- Management of keys for the tables

Metadata Entities

The various metadata entities are:

1. Site: A region of study.
2. Patch: A region within a site where a group of motes can communicate with each other without requiring additional relays or networking support.
3. Location: A piece of real-estate that is of scientific interest. A location is denoted by the geographic coordinates where some hardware is placed.
4. Node: A mote class device which facilitates data collection and communication between other motes and/or the basestation. E.g. TelosB
5. Sensor: A device which collects raw scientific data. E.g. Vaisala GMP 220 [71]

Metadata tables interact among themselves and these relationships are captured in Figure 3.5.

Metadata Hierarchy

A *site* is a region of scientific interest. The geographic location and the period of study completes the definition of a site. The site is specified by latitude (lat), longitude (lon) and the time zone (in relation to the GMT). Each site is associated with a unique ID known as a site ID. A site contains one or more regions of interest depending on the nature of the study. For example, in the 2009-10 SERC study, there were two regions of interest that were located

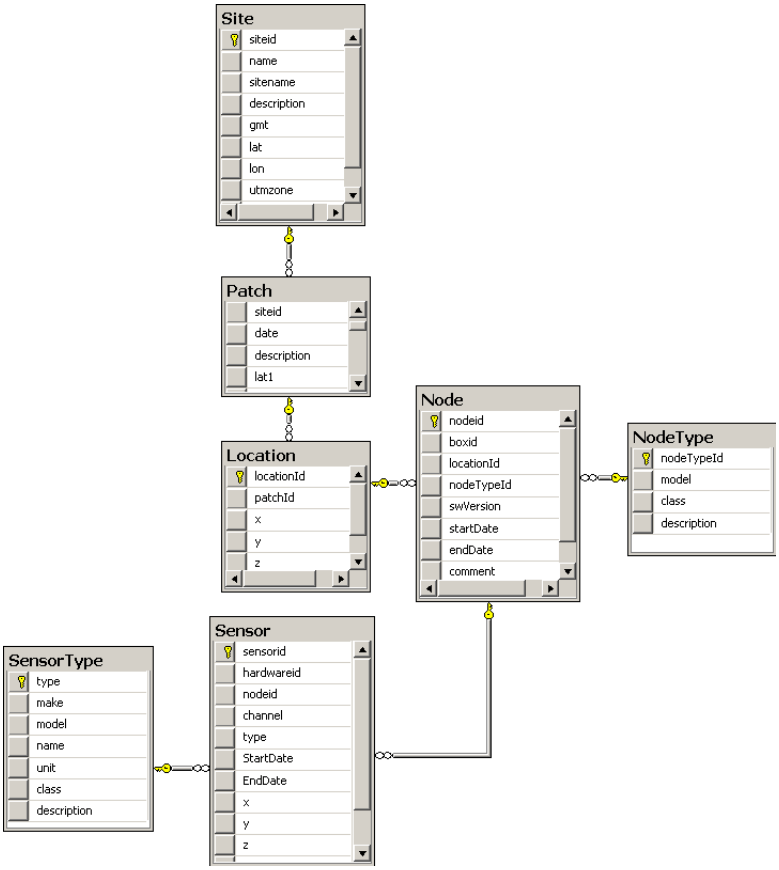


Figure 3.5: Relationship between the metadata entities.

roughly one kilometer apart. Each region contained a collection of motes that communicated with each other using their on-board radio. These regions are called *patches*. Each site may contain one or more patches, given by a unique identifier known as a patch ID The patches themselves may be connected using a more powerful radio link or a series of relay nodes.

Each patch is associated with a number of sensing locations. At each sensing location, a mote is placed. A *location* is characterized by a piece of real-estate that is being monitored. In the LUYF deployments, a location is represented by its latitude (lat), longitude (lon) and elevation (z). As with site ID and patch ID, each location has a unique location ID.

At every location, a mote (also referred to as a box) is placed during the period of study.

At any given time, only one mote is placed at one location. A mote might be replaced due to hardware failures or other reasons. Thus, during the entire period of study, one or more mote(s) may be placed at a given location at non-overlapping time periods.

Every mote is addressed using a unique identifier, referred to as a box ID. The active duration for each box is given as the time between when it was placed in the field (start date), and when it was recovered/replaced (end date). The combination of boxid, start date and end date forms a unique identifier and we refer to this as a *node* (referred by a unique nodeid). The notion of nodes allows us to effectively reuse or replace a previously used box at another location at a different time. At the time of deployment, a node entry is "opened" by recording the boxid and the time when the node was placed in the field. During replacement or removal, a node entry is "closed" by registering the time when the node was removed from the field.

Typically, each node is responsible for collecting data of one or more modalities using *sensors*. As explained previously (Section 3.2), it may make use of some on-board sensors and/or some external sensors. Each sensor is denoted by a unique identifier known as a hardwareid. The hardwareid could be something the manufacturer's unique serial number. For sensors made in the LUYF lab, each sensor was assigned a unique hardwareid. A sensor is attached by noting its start time and detached by marking the end time to only one box at one point of time. Hardwareid, start time and end time together forms a unique identifier referred to as sensor ID. The sensor replacement dynamics are similar to the box replacement dynamics. Specifically, a sensor can be attached to another node at another point of time by opening and closing the sensor entry - as long as the database reflects that it is detached from the previous box.

Motes and sensors can have different manufacturers and our schema is flexible enough to represent the mote and sensor manufacturer's information. This information is stored in the

node type and sensor type entities (as shown in Figure 3.5)

Key Management

A basic system requirement of the two phase architecture was to identify hardware and provide the ability to retrace the processing steps back to the data acquisition (ADC value stored on the mote). In meeting this requirement, it is important to be able to uniquely identify all the information that allows us to do this. From a data perspective this means that various elements need to be uniquely identifiable and consistent among themselves. In database terminology, each table presented in Figure 3.5 requires to have a primary key - a way to uniquely identify each record in the table. Furthermore, tables need to maintain referential integrity. Referential integrity implies that if a column, col, is a primary key for Table A and it exists in any other table B then all unique values of col in table B must be present in Table A. To give a very intuitive example, all unique values of locationid (foreign key) in the node table must be present in the location table where the primary key is locationid. If this fails to hold, the system violates the referential integrity property. In our system, these constraints are honored.

The system must deal with the challenge of managing keys in the stage database and the science database. Care must be taken to ensure that the metadata tables contain unique values for the primary key columns. One observes that the site table does not contain any foreign keys. The upshot of this is that the primary keys for the location and hardware tables can be derived from the site information in a systematic way. Our scheme generates a key space for each of the dependent tables (Patch, Location, Node, Sensor) as a function of the siteid. The functions used for determining the key spaces are given in Table 3.2. As an example, siteid=2 can have 256 locations from the range $256 : 511$ (inclusive). The first locationid in site 2 will be assigned a value of 256. The successive locationid's will increase

in steps of 1. We note that the number of locations was chosen based on our original system design and assumptions - we did not expect to have a deployment having more than 256 distinct locations. It is worth mentioning that there is nothing sacred about the number of keys allowed per site. In future deployments, the number of reserved patches/locations (or nodes/sensors for that matter) per site can be increased by changing the key mapping function.

At this point, you might be wondering how the keys and metadata are distributed between the stage and science databases. I would like to remind you about the roles of the two databases - the science database is a unified database that contains data streams from all deployments whereas the stage database contains deployment specific data. This implies that the science database requires a "global view" of the metadata across all deployments, whereas each stage database needs metadata information specific to its deployment only. Note that we opted for a unique flat key space (unique locationids across deployments) rather than a composite key space (site id, non unique locationid across deployments) across all the metadata tables in order to make it easier to join tables. Having one single column makes it easier to join tables and removes any ambiguity about the columns being joined.

From an implementation point of view, the metadata structure is shared between the two databases. The science database acts as the master database and it is the entry point for registering any changes in the location or hardware configurations. Each stage database synchronizes the metadata information with the science database by passing it the site ID (can be thought of as deployment ID). The science database acts as the master because it hosts data from all deployments and is therefore entrusted with the job of ensuring that there are no conflicts (unique keys) in the metadata information across deployments. Note that it would have been perfectly reasonable to have another database reserved for the metadata. We consciously avoided this because SQL Server does not support cross database referential

integrity. We would not have been able to establish integrity for the tables that store sensor stream data.

Entries can be populated in the metadata tables in a number of ways. An administrator can do it in a supervised manner by reading data from configuration files and populating the relevant tables. A web-based system can accept input directly from the user and insert records in the metadata tables. In our system, both methods have been employed extensively. Metadata updating needs to be done carefully and it is crucial that these transactions ensure that the ACID (atomicity, consistency, isolation, durability) properties are maintained. For example, if node x needs to be replaced by node y, we cannot add the entry for node y unless we put an end date on node x - both statements must go through before we can commit. Adding y without closing x leaves the system in an inconsistent state.

Metadata Dependencies

The importance of the metadata being registered with the system was discussed in Section . The lack of metadata makes the pipeline stall and over the years this has proven to be a major hurdle. We will see some of the pitfalls of recording metadata in Section 4.2. In Section 3.7 we also discuss the solution that will help streamline this process.

3.4 Health Monitoring

One of the big take aways from the pilot studies was that networks require constant monitoring and attention. The LUYF group decided that a strong emphasis needs to be put on maintenance in order to minimize data loss. It was interesting to experience that our relatively simple system produced a myriad of failure modes. Failures included the whole spectrum ranging from individual motes failing to the collapse of the entire network.

I wanted to motivate this topic a little more and share some aggregates related to mote

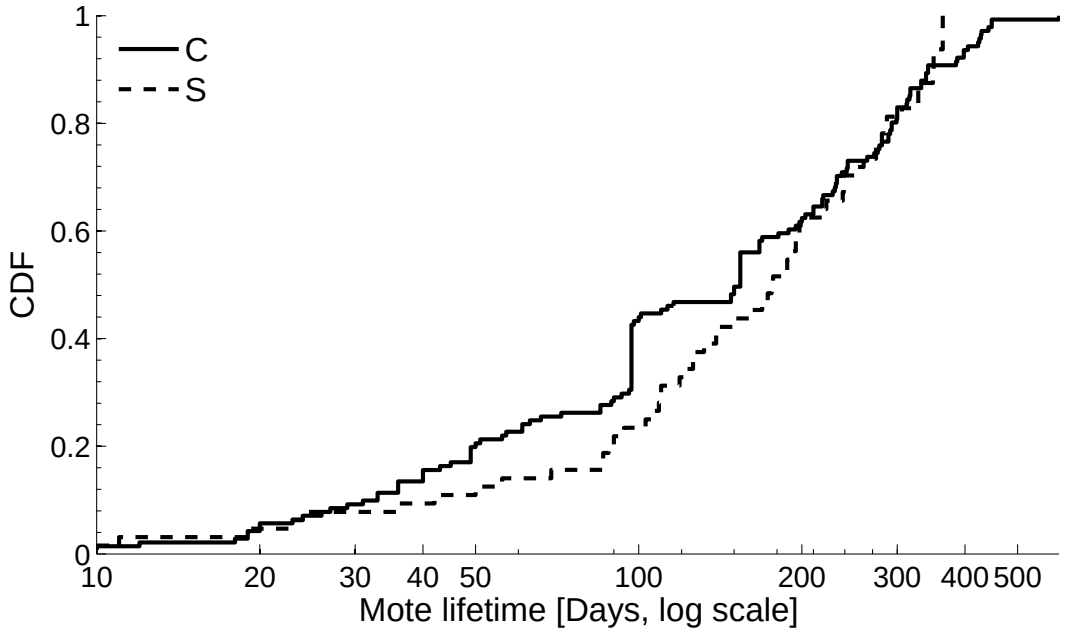


Figure 3.6: Distribution of node lifetimes for the Cub hill and SERC deployments.

failures and lifetimes. The distribution of the node lifetimes for the Cub hill and SERC deployments is shown in Figure 3.6. As the deployment gets older, the overall maintenance goes up as motes begin to fail with a higher frequency. The median node lifetime for the Cub hill deployment was found to be 154 days and that for SERC was found to be 176 days. One also observes that there is a lot of variation even though the motes are built exactly the same way. The causes for failures is shown in Figure 3.7. As expected, the most significant cause of failure is the consumption of the 19 Ah battery source. However, for a large fraction of the motes the failure cause remains a mystery. Although it is important to be able to diagnose the cause of failure, it is even more important to detect the failure and take corrective action to reduce data loss.

3.4.1 Monitoring goals

The high level goals of monitoring and maintaining the deployments were as follows

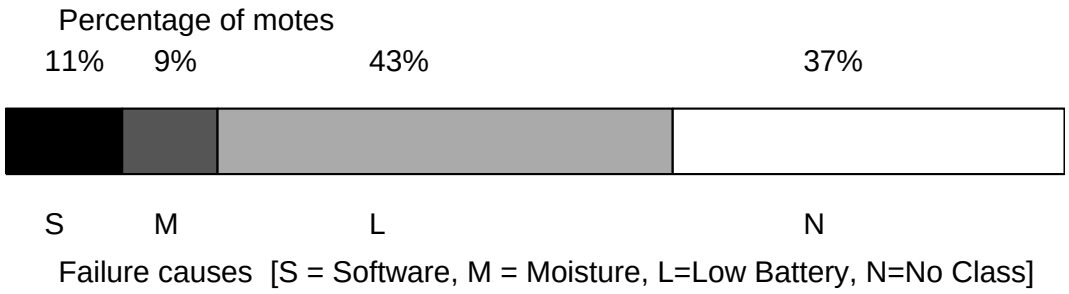


Figure 3.7: Distribution of mote failure

- Ensure early detection of unhealthy motes;
- Validate mote code based on diagnostic counters;
- Monitor the radio usage of the motes;
- Detect network partitions and communication failures;
- Correlate diagnostic information to understand system behavior

3.4.2 Overall Methodology

In this section, I will describe the mechanism by which the deployments were monitored.

In section 3.2 we introduced the idea of JournalRecords. These records contain information about the current status of the mote. Each mote maintains a set of in-memory counters that provide us with a summary of its operations. Some of the major counters are:

- Operations: current local time, reboot counter, address of last record written, number of samples taken
- I/O : number of flash reads/writes, flash read/write errors
- Radio : Radio usage, packet retransmissions, beacons sent/received.

- Sensors : Latest ADC values of all the channels
- Others : Battery voltage, moisture levels

Whenever the basestation is able to contact the mote (typically twice a day), the values of these counters get recorded at the basestation along with the current Unix timestamp. This record is what we refer to as JournalRecords - a journal of the motes actions since its last contact with the basestation. It is worth noting that these records are very small ($\sim$ 128 bytes) and therefore the radio needs to be used for a very short duration. These downloads are decoupled from the actual data records. This provides us with a mechanism for checking the vital signs of the network even if we do not download the actual data from the network. This proved to be useful in the Ecuador deployments where it was important to ensure that the network and sensors were functioning correctly - even if data downloads were not done regularly. These records are inserted into a table (Journal table) in the stage database. This table is queried and reports are created to detect and check for failure conditions. Examples of some of the prominent queries are listed below

- List the basestation contact times for each mote in the last week
- Determine the number of reboots in the last week for each mote
- Investigate the current battery and moisture levels of the boxes
- For each mote, report the radio-on fraction (duty cycle) for each day in the last week
- Obtain the timestamp of the last observation collected by each mote.

The output of these queries were consumed in two ways. The first method was to display the summary tables and graphs on a web page. We refer to this method as *KoalaReports* as this monitoring code was developed along with the Koala routing protocol. The second

Report : 1
Last Contact Table

Number	boxid	locationid	LastGatewayContact	GatewayTimeElapsed
9	63	289	8/19/2011 10:51:02 AM	16d 6h 1mi
10	34	265	8/24/2011 10:02:40 PM	10d 18h 50mi
11	442	298	8/27/2011 2:40:25 AM	8d 14h 12mi
12	40	284	8/27/2011 2:43:36 AM	8d 14h 9mi
13	61	324	8/30/2011 10:30:08 PM	4d 18h 22mi
14	219	299	9/2/2011 4:47:34 PM	2d 0h 5mi
15	9	287	9/3/2011 6:32:11 AM	1d 10h 20mi
16	453	329	9/3/2011 6:41:39 AM	1d 10h 11mi
17	499	314	9/3/2011 6:41:43 AM	1d 10h 11mi
18	502	297	9/3/2011 6:42:33 AM	1d 10h 10mi
19	7	319	9/3/2011 7:40:07 PM	0d 21h 12mi
20	490	327	9/3/2011 7:48:28 PM	0d 21h 4mi
21	81	300	9/3/2011 7:50:07 PM	0d 21h 2mi
22	24	307	9/4/2011 8:45:22 AM	0d 8h 7mi
23	28	303	9/4/2011 8:45:41 AM	0d 8h 7mi
24	37	257	9/4/2011 8:45:41 AM	0d 8h 7mi

Figure 3.8: Table shows the last known gateway contact for motes in the Cub Hill deployment. This is a screenshot (clipped) from the actual table output of KoalaReports

method involved disseminating information using the microblogging site Twitter [70]. The twitter method was meant for alarms (mote out of contact, battery levels low etc) whereas KoalaReports was used for a more detailed snapshot of the status of the motes. It took the deployment name as an argument and produces a set of tables and graphs on-the-fly that would allow the viewer to see the current state of all the motes. In the upcoming section, I will show a few examples of the tables and graphs generated by KoalaReports.

Report : 2
Reboots (Week's History)

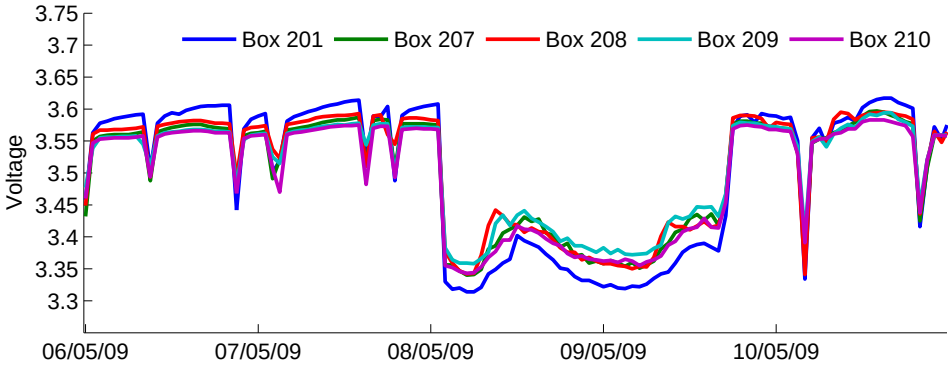
boxid	lastrbt	numReboot	revision
219	9/2/2011 4:47:34 PM	512	1898
403	8/31/2011 11:49:16 AM	3	1898
91	9/2/2011 4:41:46 PM	2	1898
8	9/3/2011 6:31:41 AM	2	1898
470	9/1/2011 2:13:53 PM	2	1898
490	9/1/2011 1:08:46 AM	2	1898
499	9/2/2011 3:27:17 AM	1	1898

Figure 3.9: Table shows the KoalaReports output that monitors the number of reboots for each mote. Note that this screenshot is clipped and it's purpose is mere illustration of the report.

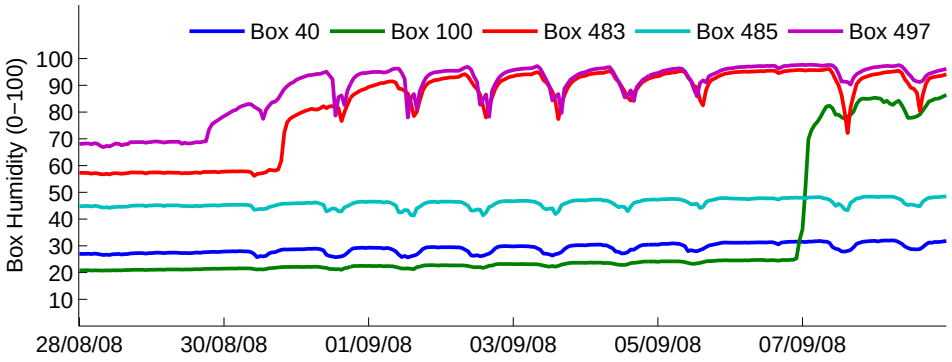
3.4.3 Examples of Monitoring Reports

An example of the gateway contact table is shown in Figure 3.8. This report is used to establish the last known time that the gateway was able to establish any contact with the node. In the example, a few of the nodes have not contacted the gateway for over 8 days. This could either imply a network partition or a mote failure. A quick look at the battery levels, moisture conditions and the number of reboots of the mote before the last known contact helps to disambiguate the two situations. For example, combining the information from Figure 3.8 and Figure 3.9, we are able to see that box 219 has been out of contact for over 2 days and the number of reboots in the last week has been over 512. On investigating further, we find that the maximum battery voltage over the week was found to be 2.69V - significantly lower than the level (¿ 3V) that is required for smooth operation of the mote.

Depletion of the battery voltage is the most dominant reason for node failures (as shown



(a) The impact on the battery levels when the radio fails to turn off. The sharp spikes correspond to the periodic downloads done by the basestation.



(b) Rain events causing sudden jumps in the box moisture. Note that some boxes are unaffected by the rain events while some show a jump in the moisture conditions within the box enclosure.

Figure 3.10: Examples of mote failures caused by battery levels and high moisture.

in Figure 3.7). The voltage levels of five boxes in the cub hill deployment for a period of six days is shown in Figure 3.10(a). We see some prominent features in this graph. Firstly, there are sharp downward spikes at regular intervals. These spikes correspond to times when the basestation is downloading data from these nodes. The radio needs to be kept on for a short amount of time and this results in a dip in the battery levels. Secondly, there is a significant drop in the voltage levels between 8/5/2009 and 10/5/2009. This signature is characteristic and indicative of a failure to turn off the radio. The motes were unable to communicate with the basestation due to a hardware problem and this resulted in a failure to turn off the radios

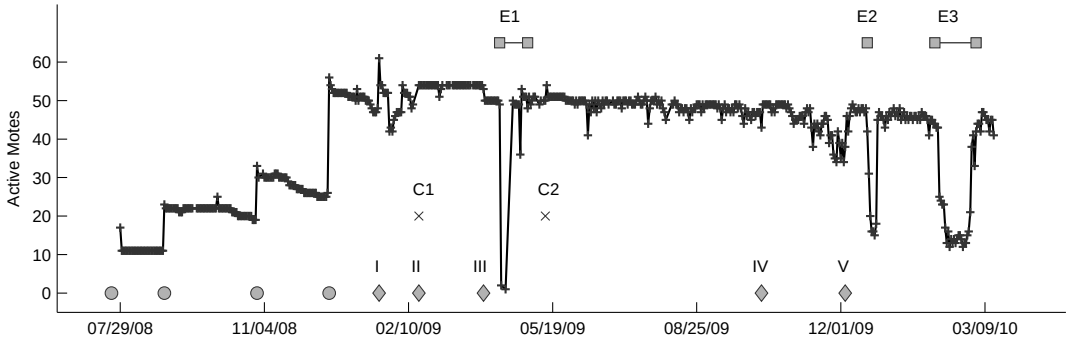


Figure 3.11: The number of motes that were in contact with the gateway during each day of the Cub Hill deployment. The circles at the bottom of the figure represent network expansions while diamonds represent mote replacements. Event C1 represents the watchdog fix and Event C2 corresponds to the transition from 6 hr to 12 hr downloads. Finally, event E1 corresponds to the basestation's failure and E2 and E3 are the two snow storms.

of the motes. We were able to detect this early enough to not cause significant damage to the network.

High levels of moisture in the box enclosure is another common cause of node failures. Figure 3.10(b) shows box humidity values for a period of a week for 5 boxes in the Cub hill deployment. The average moisture levels of boxes 100, 483 and 497 jump up after a big rain event. After 7/9/2008, these boxes are operating with a moisture level that is greater than 80%. Boxes 483 and 497 eventually failed due to excessive moisture.

3.4.4 Case Study - Cub Hill Mote Contacts

The Cub hill deployment has been maintained since July 2008. A number of events have played a role in how this deployment has shaped up. Some of the major events are shown in Figure 3.11. This graph is created using the journal records and it shows the number of motes that are active during each day for a period between July 2008 and March 2010. The variation in the number of active motes is mainly due to mote failures and due to network

disconnections. The purpose of this graph is to give a flavor for various factors that may have a bearing on the overall health of the network.

The network went through three major network expansions. One can observe that as more nodes joined the network, more nodes also started failing. The cause of these failures were loosely related to the mote software but we were unable to track down the exact cause of these failures. We found that the motes were freezing and were unable to come out of this stuck state. Prior to Feb 2009, motes had to replaced very frequently. At that time, we took a decision to prevent these nodes from getting stuck by using a watchdog timer. The watchdog timer is an independent circuit that works as follows. A timer is initialized to zero. The timer starts counting and when it counts up to the watchdog timeout period, it reboots the system. To avoid the system from rebooting, an interrupt is set up that fires with a period that is less than the watchdog timeout period. This interrupt will reset the watchdog timer so that is starts counting from zero again. In Figure 3.11, the watchdog fix is marked as C1.

After the watchdog timer fix, the number of active motes stayed relatively constant until we encountered the period marked E1. This period corresponds to the time when the basestation was experiencing some hardware failures. These failures resulted the nodes to be stuck in a 'radio-on' state as the basestation initiates the process of putting the motes to sleep (turing off their radios). At the point when the basestation problem was fixed, the network had been active for over 10 months. The overall battery levels were low. In order to reduce the power consumption, we minimized the download frequency from 6 hours to 12 hours (event marked as C2). Even though the duty cycle was lowered, the motes had very low battery levels and many motes started to fail and reboot. The period between E1 and E2 represents a time when many motes started to fail.

Events E2 and E3 represent two major snow storms. February 5 observed around 30 inches of snow. The snow and ice was deposited on the motes and it took days to melt. During

this period, a number of the motes could were unable to be reached by the basestation.

Visualizing these summaries allowed us to understand various subtle aspects of the system. The use of visual aids in monitoring the network proved to be very effective. At the same time, we started turning our attention to the challenge of effectively understanding the sensor observations being collected.

3.5 Grazor: Data Access

These LUYF networks were deployed with a goal of understanding the spatial and temporal heterogeneity related to soil. As such, the data consumers were interested in "mining" for information rather than using it for validating a highly focussed hypothesis. The typical usage of the data was to explore various features in the data by looking at it through various prisms - grouping by location/modalities/time etc. This usage pattern motivated us to develop Grazor - a tool that allows scientists to visually and intuitively explore the sensor data along various dimensions.

The idea of developing Grazor started taking shape in the summer of 2009. At that point, we had overcome the initial teething problems of collecting and storing data. Koala and two-phase system had become relatively stable. The challenge now was to engage the environmental scientists to use the data being collected. A number of meetings were held with the data consumers to understand their requirements. These requirements governed the functionality and graphical user interface design.

The main questions that came up during the discussions are listed below:

- Can I visualize data by selecting specific locations, modalities and time intervals?
- Can I look at data trends at various levels of detail - yearly, weekly, daily and hourly?
- Can I download data that has been narrowed down by visual exploration?

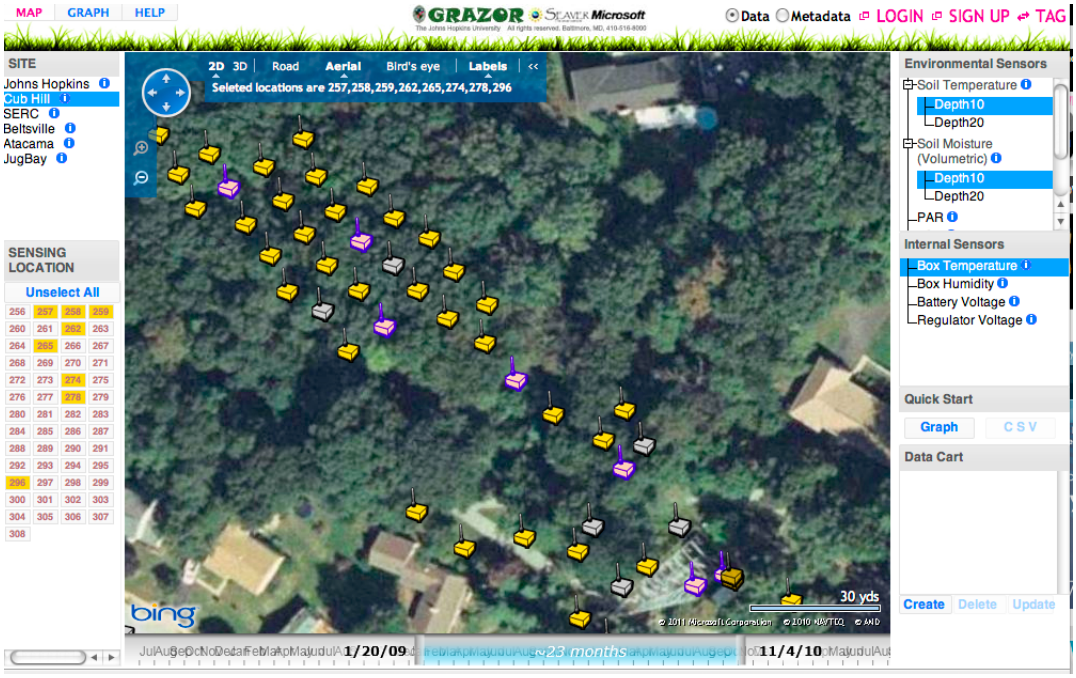


Figure 3.12: **Grazor User Interface**

- Can I save the visual charts and retrieve them a later point in time?

3.5.1 Overall Design

In this section, we will draw attention to the most significant components of the grazor tools. I will describe at a high level three main pieces - Data selection, Data display experience and exporting/saving the data.

The data selection and data display components are shown in Figure 3.12 and Figure 3.13 respectively. Lets begin by taking a look at the Data selection module.

Data Selection

A fundamental design consideration was to engage the scientists and make it intuitive for them to use. Selecting data required a fair amount of thought and tinkering with until we

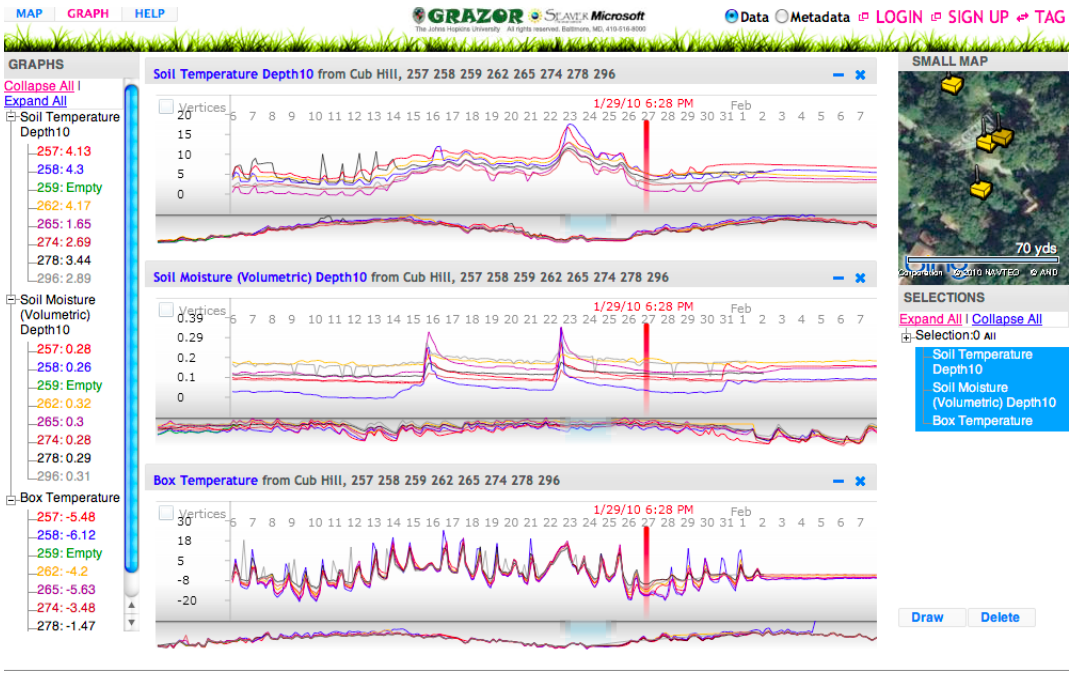


Figure 3.13: Output of Grazor after user makes selections

settled on a design that satisfied the scientists and was easy enough to show on a webpage. The selection interface (shown in Figure 3.12) comprises of three parts - selecting locations, time interval and sensing modalities. The content of these parts is obtained by querying the database.

The location panel can be further divided into deployments and the locations within the deployment shown in the left panel of the user interface. The time controller panel at the bottom of the interface lets users select the time period. The two vertical bars on the time controllers lets a user expand or contract the time window. Once the locations and the time period are fixed, the system queries the database to find the active sensors for these locations and time periods and displays them on top part of the right panel. The sensors are broken down into external sensors (soil temperature, moisture, CO_2 etc) and internal sensors (box temperature, humidity, light).

The user can visualize the data by clicking on the graph button on the quickstart menu shown on the right panel. The current selection state can be saved (and deleted later) and these entries will appear in the *data cart* - a staging area for saving and retrieving selections during an active session. The data cart appears on the south east corner of the selection screen (Figure 3.12). The output of these selections is shown in the *graph screen*. The selection screen is referred to the *map screen*. Users can toggle between the two screens by clicking on the map/graph screen on the north-west corner of the interface.

3.5.2 Data Display and Interactivity

Grazor needs to extract the data corresponding to these selections and the database is contacted for this purpose. A stored procedure, written in transact-SQL, accepts the input parameters and returns a handle to the output. This output is piped to a plotting library that was developed and customized by the LUYF developers. Figure 3.13 displays the result of the selections made in Figure 3.12.

One chart is created for each sensing modality. These set of three charts is collectively referred to as a *chart group*. Please note that each chart contains a time controller (blue panel with start and end time handles at the bottom of each chart) which lets users "zoom-in" and "zoom-out". This allows users to change the start and end time of the chart group and visualize data at arbitrary levels of detail. Furthermore, the time controllers of all the charts are synchronized, that is, a change in one will also change the resolution of the others. This feature lets users correlate data from various modalities at the same location. The red vertical line shown in Figure 3.13 tracks the user's mouse. It shows the actual physical values for all the locations and selected modalities on the left panel for the time instance denoted by the mouse position. Users can add, subtract or minimize charts using the controls on the

charts. The selection controller on the bottom right allows user to manage multiple groups of charts. These chart groups have a direct relation with the entries in the data cart.

Each time the user requests data at a different resolution (by zooming in or zooming out) , Grazor contacts the database and requests for the data using the pyramid scheme (Section 3.3.3).

Data Exporting and Saving

Grazor lets the users export the data as text files. Typically, users do not want to export all the data but a subset of it. This subset is usually determined after spending some amount of time visualizing and exploring. This subset can be downloaded as comma separated value (CSV) files. The user also has the option of saving (bookmarking) the current Grazor state. This state includes the currently selected locations, , time period and modalities . This mechanism allows the user to return at a later point of time (perhaps from another computer), retrieve this state and continue working. Each user's bookmarks are stored on the server in a database. When a user log's in his/her credentials are verified and all the bookmarks saved by the user are retrieved. If the user selects a bookmark, Grazor will use the selection attributes (locations, time period and modalities) to restore the state that existed prior to saving the bookmark.

In a nutshell, Grazor allows users to visualize, export and save data of interest. It acts as a web-based visualization layer to the LUYF database.

3.6 System Shortcomings

The two phase data pipeline has been used in seven deployments over the course of four years. A number of important lessons have been learnt during this time and we are able to identify shortcomings and opportunities to improve the overall efficacy of the system. These

shortcomings are listed below:

- Lack of flexibility in registering metadata
- High maintenance efforts for the upload application
- Failure to monitor the health of the deployment effectively
- Inability to deal with sensor faults for different modalities

We will now look at the details of these shortcomings and discuss the experiences we have had over the course of these four years.

3.6.1 Metadata Inflexibility

The importance of metadata has been highlighted in Sections 2 and 3. The methodology for recording the metadata information is to map the site-patch-location-node-sensor associations during the deployment. This information is then entered into the system using the web portal, Grazor. The existing metadata collection system has posed problems on a number of different levels. They are listed below:

1. Metadata (node, sensor and location association) are entered manually
2. Inability to update metadata on-site
3. Challenges in collecting geo-location information
4. Data processed with inaccurate metadata is not gracefully offloaded

A typical outdoor deployment is very chaotic and human resource intensive. During this process, researchers tend to focus most of their effort on the immediate logistical challenges of setting up the deployment. Although collection of metadata is crucial, its value is only

realized once the data needs to be looked at. Thus, metadata recording is not prioritized during the deployment resulting in inconsistencies in the final output.

Our deployment experiences in Ecuador have shown us that researchers may need to change the channel assignment during the middle of the deployment. Due to the absence of communication in remote areas, these hardware configuration changes may not be relayed to the database administrators. This results in the application of an incorrect conversion function to the raw stream for the channels that have changed. For example, let's consider that mote X is connected with a soil moisture sensor on channel 1. Ten days into the deployment a researcher decides that it is more valuable to have a soil CO_2 sensor at that location. Let's further suppose that the researcher forgets to update the system regarding this change. As a result, the system assumes that data on channel 1 of mote X is coming from a soil moisture sensor even though it represents a CO_2 stream. On a number of occasions, the data recorded on log books or loose papers have been misplaced amidst deployment pressures and this information never got entered into the system.

The system assumes that the location information is collected and entered in the system so that data can be accessed and visualized using Grazor. This information is difficult to collect unless there is access to an accurate GPS or surveying equipment (which is expensive and labor-intensive). The lack of this information makes it difficult to show data on Grazor. A related issue is that the system assumes that the external sensors connected to a box are co-located with the box (sensors inherit location information from the box). These sensors may differ in terms of their depth. At USDA, we encountered situations where the same box had sensors that were collecting data at the same depth but at two different locations. The schema has provisions for adding offsets from the box so the issue can be dealt with. However, logically speaking they are completely different locations so it may be more effective if each sensor had an associated location.

Once changes in hardware configuration are made available to the system, the system should delete data resulting from the incorrect application of the conversion function (data processed prior to the hardware change notification) and reprocess all the data with the correct conversion function. At present, the system does not do this. This shortcoming is easy to fix and can be implemented easily by making sure that upon detecting a hardware change, the system offloads the previously processed data and reprocesses it with the correct conversion function.

3.6.2 Upload Application Maintenance

The details of the upload application were described in Section 3.3.1. A major shortcoming of this design is that two separate code bases need to be maintained – one for the basestation and another one for the upload application. Whenever the record formats change, code needs to be updated at both places. This design proved costly during the Brazil deployment where we decided to compress records to save space on the flash. Code was changed on the mote, the basestation and the upload application. The former two code changes are inevitable but the last one could be easily avoided if the basestation was set-up to communicate with the database directly.

The scientific and networking needs for each deployment vary. Thus, the record organization and the way information is stored on the flash may change from one deployment to the next. At present, the upload application is tightly coupled with the way records are stored on the flash and this structure is shared by all the deployments. Furthermore, there is a one to one correspondence between the database tables and the types of records. This makes the system highly inflexible if changes need to be made to accommodate new record types. For example, some deployments may be used to test out some networking code which might

require changes to the existing record types. In the current design, the upload application expects a fixed format for it to insert records in to the database. This tight coupling makes it difficult to maintain and causes duplication of effort as the basestation code and the upload application code need to be updated to accommodate the necessary changes.

One way to alleviate some of these challenges and reduce the maintenance efforts is to keep only one code - at the basestation. This code should be able to talk with the database server and upload the outstanding data directly by inserting data according to the database schema.

3.6.3 Dealing with Sensor Faults

Over the years, we have shared data with a number of environmental scientists that have collaborated with the LUYF group. Their feedback has consistently focussed on the number of faults in the data. This is not surprising, considering that a high degree of faults in outdoor deployments have been experienced by a number of research groups. These groups have reported that data gathered from sensor networks (and particularly soil monitoring sensor networks) tend to be inherently faulty and the failure modes are highly unpredictable [47,52, 61]. As mentioned previously, we employ a simple median filter method to to detect faults. This method is used because of its wide applicability - considering that the data collected by our network comprises of a number of modalities. On the one hand, the scientists are in the best position to understand and pick out the faulty data points due to their domain expertise, but on the other hand, many environmental scientists are not comfortable working with large amounts of data and automate this process effectively. Furthermore, it has been challenging to develop a general method for such heterogeneous datasets. The modalities tend to respond differently to the environment, and therefore, picking a set of features that is effective for all

modalities has been proven to be rather problematic.

3.7 Discussion

After four years of experience with the two-phase architecture and storing sensor data, I would like to spend some time discussing about the design of future data pipelines.

3.7.1 Streamlining the Two-phase Architecture

The following tasks need to be addressed to improve the overall functioning of the pipeline

1. Eliminate the dependence on field researchers to collect metadata;
2. Establish well defined processes for monitoring system health;
3. Involve domain scientists and gather feedback at the earliest

The primary cause for the inconsistencies in the metadata collection lies with the premise that this information can be reliably collected during a deployment. This assumption has proven to be inaccurate time after time. Ideally, a design that auto detects changes in the hardware as sensors are plugged in and out would cut the human out of the loop making things more streamlined. These metadata change detections can be stored on the mote as a new record type. When the basestation downloads such a record, it can interpret the information appropriately and directly update the metadata tables that are present in the database. Although this design addresses challenges (see Section 3.6.1) related to (1) and (2), it does not address (3). It is not entirely clear on the best way to approach this problem. One possible solution is to use logical names (e.g. plot 3 at USDA) to represent location information abstractly. The down side is that it is not feasible to display such locations on a map in a consistent way.

The definition of a sensor fault is somewhat arbitrary. What one individual calls a fault may be the signal for another individual. Furthermore, it is difficult to get a clear consensus on the spectrum of sensor faults. Some fault types are well-documented in the literature. A stuck-at-value fault is one such example [61]. The system must be able to deal with such faults. However, a vast majority of faults are highly subjective and they can be identified by domain scientists after visual inspection but it is hard to automate this process. This problem is compounded by the fact that a typical sensor network collects data from multiple modalities that have varying response characteristics and exhibit huge variability in their failure modes.

One possible approach to deal with this is to build an engine that begins by asking each user to identify a few faults per modality. This information could be leveraged to identify other faults using a choice of machine learning algorithms. This methodology of mark, extract and confirm would be iterative and would allow scientists to "tune" the results of the fault detection based on their feedback. Faults identified based on one user's individual fault profile would differ from those identified for other since the tagged (faulty vs non-faulty) data will vary per individual. If a user does not want to spend his time identifying and confirming faults, all data points tagged as faults by other users will be assumed to be faults and will not be provided to him during data visualizations or downloads. This philosophy is similar to one used by GalaxyZoo [51].

Table 3.1: LUYF deployments. P stands for pilot deployments, T denotes deployments conforming to a two-phase architecture and C refers to campaign deployments

Deployment	Start	End	Nodes	Purpose
Olin (P)	9/5/05	7/21/06	10	Test MicaZ [17] hardware, software and experiment with sampling rates
Leakin Park (P)	3/3/06	11/5/07	6	Test MicaZ hardware and software platform
Jugbay-I (P)	6/22/07	4/26/08	13	Test TelosB platform and study nesting of box turtles [50]
Jugbay-II (P)	6/11/08	11/8/08	8	Study overwintering of box turtles
Olin-II (T)	7/13/08	9/15/09	18	Test Koala [44] download protocol and two-phase data loading
CubHill-I (T)	7/29/08	5/12/11	53	Study spatial and temporal heterogeneity of soil in urban forests
SERC (T)	3/11/09	7/20/10	37	Study impact of leaf litter and forest types on soil respiration
USDA (T)	7/23/09	7/20/10	22	Study impact of soil conditions on crop yields
Atacama (T)	8/18/09	1/16/10	3	Monitor conditions around a telescope at an high altitude
Brazil (C)	11/17/09	12/17/09	50	Collect atmospheric data to improve micro-font development models
Ecuador-I (C)	1/12/10	1/31/10	20	Contrast soil respiration of old and young forests in summer
Ecuador-II (C)	5/22/10	6/7/10	20	Study soil respiration of old and young forests in winter
CubHill-II (T)	5/12/11	–	77	Study and understand relationship between soil CO_2 and ambient CO_2

Table 3.2: **Function mappings for generating key spaces for the Patch, Location, Node and Sensor tables.**

Table	key range	Example (for siteid=2)
Patch	$2^4 * (siteid - 1) : 2^4 * (siteid) - 1$	**Key Range :** $16 : 31$
Location	$2^8 * (siteid - 1) : 2^8 * (siteid) - 1$	**Key Range :** $256 : 511$
Node	$2^{10} * (siteid - 1) : 2^{10} * (siteid) - 1$	**Key Range :** $1024 : 2047$
Sensor	$2^{16} * (siteid - 1) : 2^{16} * (siteid) - 1$	**Key Range :** $65536 : 131071$

Chapter 4

Time Reconstruction I - Sundial

Time reconstruction was briefly introduced in the previous chapter. In this Chapter, I will introduce its fundamentals and describe some of its challenges. Up until the Olin-II deployment (Table 3.1), the LUYF deployments did not have a persistent basestation. In the absence of a time synchronization protocol and a persistent basestation, assigning timestamps to measurements turned out to be a challenge. The problem is fundamentally caused by random mote reboots. This chapter describes the discovery of this problem and our data-driven solution, referred to as *Sundial*.

4.1 Introduction

A number of environmental monitoring applications have demonstrated the ability to capture environmental data at scientifically-relevant spatial and temporal scales [67, 69]. These applications do not need online clock synchronization and in the interest of simplicity and efficiency often do not employ one. Indeed, motes do not keep any global time information, but instead, use their local clocks to generate local timestamps for their measurements. Then, a postmortem timestamp reconstruction algorithm retroactively uses (local, global) timestamp pairs, recorded for each mote throughout the deployment, to reconstruct global timestamps for all the recorded local timestamps. This scheme relies on the assumptions that a mote's

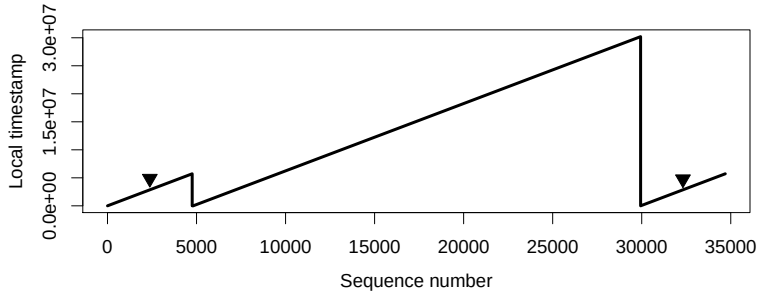


Figure 4.1: An illustration of mote reboots, indicated by clock resets. Arrows indicate the segments for which anchor points are collected.

local clock increases monotonically and the global clock source (e.g., the base-station's clock) is completely reliable. However, we have encountered multiple cases in which these assumptions are violated. Motes often reboot due to electrical shorts caused by harsh environments and their clocks restart. Furthermore, basestations' clocks can be desynchronized due to human and other errors. Finally the basestation might fail while the network continues to collect data.

We present *Sundial*, a robust offline time reconstruction mechanism that operates in the absence of any global clock source and tolerates random mote clock restarts. Sundial's main contribution is a novel approach to reconstruct the global timestamps using only the repeated occurrences of day, night and noon. We expect Sundial to work alongside existing postmortem timestamp reconstruction algorithms, in situations where the basestations' clock becomes inaccurate, motes disconnect from the network, or the basestation fails entirely. While these situations are infrequent, we have observed them in practice and therefore warrant a solution. We evaluate Sundial using data from two long-term environmental monitoring deployments. Our results show that Sundial reconstructs timestamps with an accuracy of one minute for deployments that are well over a year.

4.2 Problem Description

The problem of reconstructing global timestamps from local timestamps applies to a wide range of sensor network applications that correlate data from different motes and external data sources. This problem is related to mote clock synchronization, in which motes' clocks are persistently synchronized to a global clock source. However, in this work, we focus on environmental monitoring applications that do not use online time synchronization, but rather employ postmortem timestamp reconstruction to recover global timestamps.

4.2.1 Recovering Global Timestamps

As mentioned before, each mote records measurements using its local clock which is not synchronized to a global time source. During the lifetime of a mote, a basestation equipped with a global clock collects multiple pairs of (local, global) timestamps. We refer to these pairs as *anchor points*[1]. Furthermore, we refer to the series of local timestamps as LTS and the series of global timestamps as GTS. The basestation maintains a list of anchor points for each mote and is responsible for reconstructing the global timestamps using the anchor points and the local timestamps.

The mapping between local clock and global clock can be described by the linear relation $GTS = \alpha \cdot LTS + \beta$, where α represents the slope and β represents the intercept (start time). The basestation computes the correct α and β for each mote using the anchor points. Note that these α and β values hold, if and only if the mote does not reboot. In the subsections that follow, we describe the challenges encountered in real deployments where the estimation of α and β becomes non-trivial.

[1]We ignore the transmission and propagation delays associated with the anchor point sampling process.

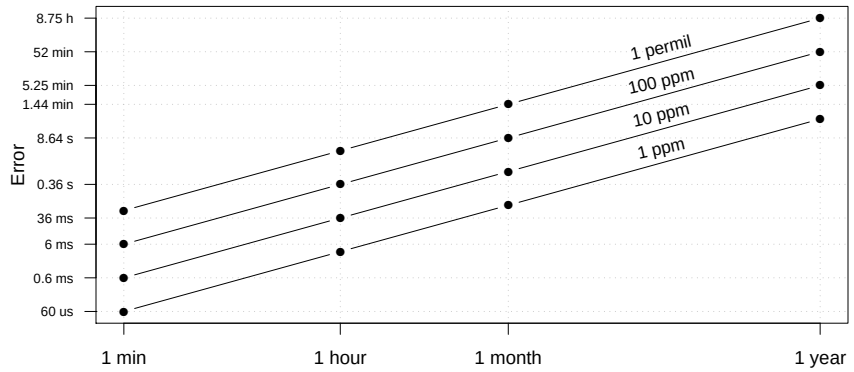


Figure 4.2: Time reconstruction error due to α estimation errors as a function of the deployment lifetime.

4.2.2 Problems in Timestamp Reconstruction

The methodology sketched in Section 4.2.1 reconstructs the timestamps for blocks of measurements where the local clock increase monotonically. We refer to such blocks as *segments*. Under ideal conditions, a single segment includes all the mote's measurements. However, software faults and electrical shorts (caused by moisture in the mote enclosures) are two common causes for unattended mote reboots. The mote's local clock resets after a reboot and when this happens we say that the mote has started a new segment.

When a new segment starts, α and β must be recomputed. This implies that the reconstruction mechanism described above must obtain at least two anchor points for each segment. However, as node reboots can happen at arbitrary times, collecting two anchor points per segment is not always possible. Figure 4.1 shows an example where no anchor points are taken for the biggest segment, making the reconstruction of timestamps for that segment problematic. In some cases we found that nodes rebooted repeatedly and did not come back up immediately. Having a reboot counter helps recover the segment chronology but does not provide the precise start time of the new segment.

Furthermore, the basestation is responsible for providing the global timestamps used in

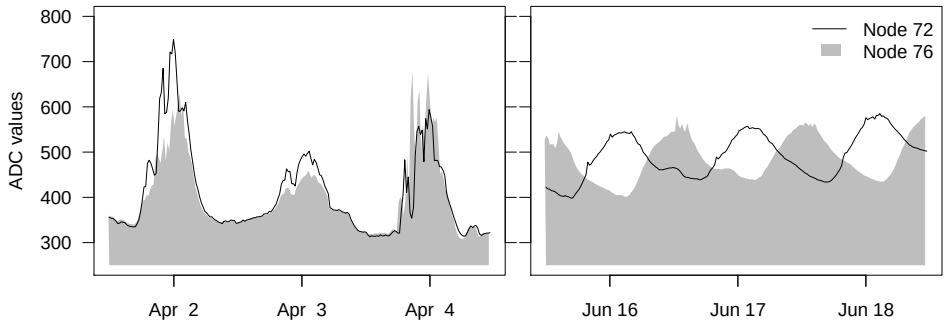


Figure 4.3: Ambient temperature data from two motes from the *L* deployment. The correlation of temperature readings in the left panel indicates consistent timestamps at the segment's start. After two months, the mote's reading become inconsistent due to inaccurate α estimates.

the anchor points. Our experience shows that assuming the veracity of the basestation clock can be precarious. Inaccurate basestation clocks can corrupt anchor points and lead to bad estimates of α and β introducing errors in timestamp reconstruction. Long deployment exacerbate these problems, as Figure 4.2 illustrates: an α error of 100 parts per million (ppm) can lead to a reconstruction error of 52 minutes over the course of a year.

4.2.3 A Test Case

Our *Leakin Park* deployment (referred to as *"L"*) provides an interesting case study of the problems described above. The *L* deployment comprised six motes deployed in an urban forest to study the spatial and temporal heterogeneity in a typical urban soil ecosystem. The deployment spanned over a year and a half, providing us with half a million measurements from five sensing modalities. We downloaded data from the sensor nodes very infrequently using a laptop PC and collected anchor points only during these downloads. One of the soil scientists in our group discovered that the ambient temperature values did not correlate among the different motes. Furthermore, correlating the ambient temperature with an independent weather station, we found that the reconstruction of timestamps had a major error

Algorithm 1 Robust Global Timestamp Reconstruction (RGTR)

```
constants
Q                                          ▷ Constant used to identify anchor points for the segment
δ_HIGH, δ_LOW, δ_DEC                       ▷ Constants used in iterative fit

procedure ClockFit(ap)
    (r, i) ← (0, 0)
    q ← HoughQuantize(ap)
    for each γ in Keys(q) do
        s ← Size(q{γ})
        if s > r then
            (r, i) ← (s, γ)
    return ComputeAlphaBeta(q{i})

procedure HoughQuantize(ap)
    q ← {}                                 ▷ Map of empty sets
    for each (lts_i, gts_i) in ap do
        for each (lts_j, gts_j) in ap and (lts_j, gts_j) ≠ (lts_i, gts_i) do
            α ← (gts_j − gts_i)/(lts_j − lts_i)
            if 0.9 ≤ α ≤ 1.1 then          ▷ Check if part of the same segment
                β ← gts_j − α · lts_j
                γ ← ROUND(β/Q)
                Insert(q{γ}, (lts_i, gts_i))
                Insert(q{γ}, (lts_j, gts_j))
    return q

procedure ComputeAlphaBeta(ap)
    δ ← δ_HIGH
    bad ← {}
    while δ > δ_LOW do
        (α, β) ← LLSE(ap)
        for each (lts, gts) ∈ ap and (lts, gts) ∉ bad do
            residual ← (α · lts + β) − gts
            if residual ≥ |δ| then
                Insert(bad, (lts, gts))
        δ ← δ − δ_DEC
    return (α, β)
```

in it.

Figure 4.3 shows data from two ambient temperature sensors that were part of the L deployment. Node 72 and 76 show coherence for the period in April, but data from June are completely out-of-sync. We traced the problem back to the laptop acting as the global clock source. We made the mistake of not synchronizing its clock using NTP before going to the field to download the data. As a result the laptop's clock was off by 10 hours, giving rise to large errors in our α and β estimates and thereby introducing large errors in the reconstructed timestamps. To complicate matters further, we discovered that some of the motes had rebooted a few times between two consecutive downloads and we did not have any anchor points for those segments of data.

4.3 Solution

The test case above served as the motivation for a novel methodology that robustly reconstructs global timestamps. The Robust Global Timestamp Reconstruction (RGTR) algorithm, presented in Section 4.3.1, outlines a procedure to obtain robust estimates of α and β using anchor points that are potentially unreliable. We address situations in which the basestation fails to collect any anchor points for a segment through a novel method that uses solar information alone to generate anchor points. We refer to this mechanism as Sundial.

4.3.1 Robust Global Timestamp Reconstruction (RGTR)

Having a large number of anchor points ensures immunity from inaccurate ones, provided they are detected. Algorithm 1 describes the Robust Global Timestamp Reconstruction (RGTR) algorithm that achieves this goal. RGTR takes as input a set of anchor points (ap) for a given segment and identifies the anchor points that belong to that segment, while censoring the bad ones. Finally, the algorithm returns the (α, β) values for the segment. RGTR assumes the availability of two procedures: INSERT and LLSE. The INSERT(x, y) procedure adds a new element, y, to the set x. The Linear Least Square Estimation [20], LLSE procedure takes as input a set of anchor points belonging to the same segment and outputs the parameters (α, β) that minimize the sum of square errors.

RGTR begins by identifying the anchor points for the segment. The procedure HOUGHQUANTIZE implements a well known feature extraction method, known as the Hough Transform [19]. The central idea of this method is that anchor points that belong to the same segment should fall on a straight line having a slope of ~ 1.0. Also, if we consider pairs of anchors (two at a time) and quantize the intercepts, anchors belonging to the same segment should all collapse to the same quantized value (bin). HOUGHQUANTIZE returns a map, q, which stores the an-

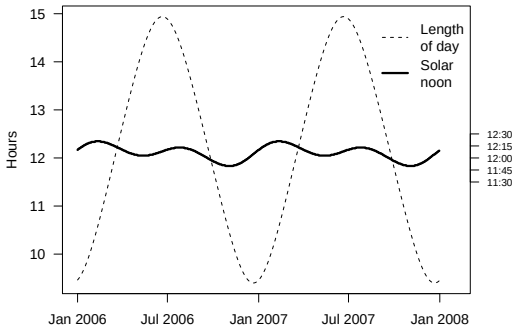


Figure 4.4: The solar (model) length of day (LOD) and noon pattern for a period of two years for the latitude of our deployments.

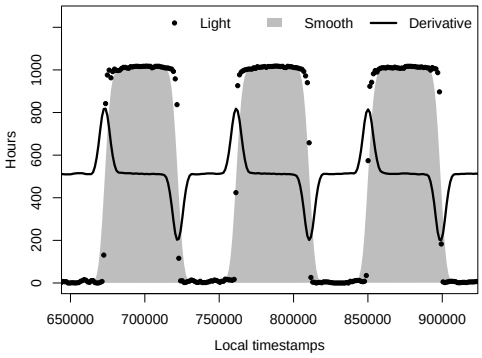


Figure 4.5: The light time series (raw and smoothed) and its first derivative. The inflection points represent sunrise and sunset.

chor points that collapse to the same quantized value. The key (stored in i) that contains the maximum number of elements contains the anchor points for the segment.

Next, we invoke the procedure COMPUTEALPHABETA to compute robust estimates of α and β for a given segment. We begin by creating an empty set, bad. The set bad maintains a list of all anchor points that are detected as being outliers and do not participate in the parameter estimation. This procedure is iterative and begins by estimating the fit (α, β) using all the anchor points. Next, we look at the residual of all anchor points with the fit. Anchor points whose residuals exceed the current threshold, δ, are added to the bad set and are excluded in the next iteration fit. Initially, δ is set conservatively to δ_{HIGH}. At the end of every iteration, the δ threshold is lowered and the process repeats until no new entries are added to the bad set, or δ reaches δ_{LOW}.

4.3.2 Sundial

The parameters of the solar cycle (sunrise, sunset, noon) follow a well defined pattern for locations on Earth with a given latitude. This pattern is evident in Figure 4.4 that presents

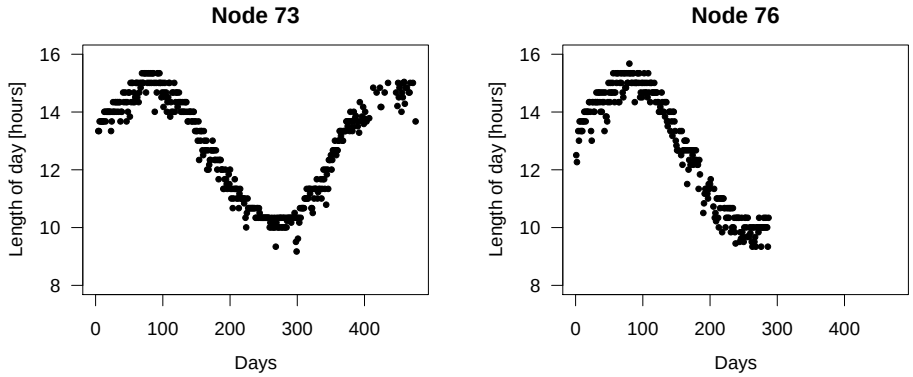


Figure 4.6: The length of day pattern for two long segments belonging to different nodes. Day 0 represents the start-time for each of the segments.

the length of day (LOD) and solar noon for the period between January 2006 and June 2008 for the latitude of the *L* deployment. Note that the LOD signal is periodic and sinusoidal. Furthermore, the frequency of the solar noon signal is twice the frequency of the LOD signal. We refer the reader to [23] for more details on how the length of day can be computed for a given location and day of the year.

The paragraphs that follow explain how information extracted from our light sensors can be correlated with known solar information to reconstruct the measurement timestamps.

Extracting light patterns:

We begin by looking at the time series L_i of light sensor readings for node i. L_i is defined for a single segment in terms of the local clock. First, we create a smooth version of this series, to remove noise and sharp transients. Then, we compute the first derivative for the smoothed L_i series, generating the D_i time-series. Figure 4.5 provides an illustration of a typical D_i series overlaid on the light sensor series (L_i). One can notice the pattern of inflection points representing sunrise and sunset. The regions where the derivative is high represent mornings, while the regions where the derivative is low represent evenings. For

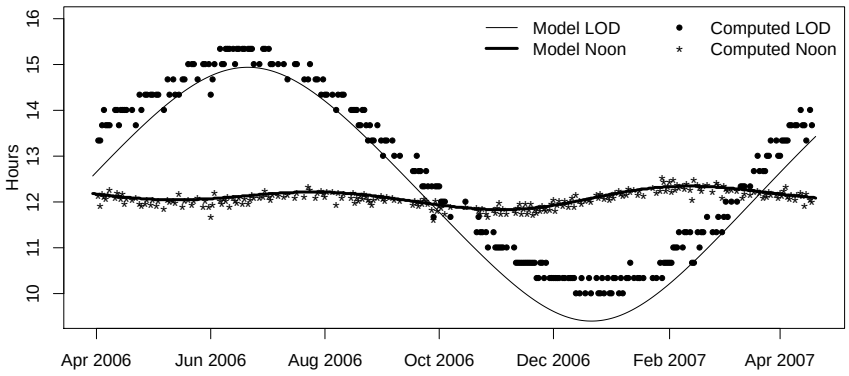


Figure 4.7: An illustration of the computed LOD and noon values for the lag with maximum correlation with the solar model.

this method, we select sunrise to be the point at which the derivative is maximum and sunset the point at which the derivative is minimum. Then, LOD is given as the difference between sunrise and sunset, while noon is set to the midpoint between sunrise and sunset.

The method described above accurately detects noon time. However, the method introduces a constant offset in LOD detection and it underestimates LOD due to a late sunrise detection and an early sunset detection. The noon time is unaffected due to these equal but opposite biases. In practice, we found that a simple thresholding scheme works best for finding the sunrise and sunset times. The light sensors' sensitivity to changes simplifies the process of selecting the appropriate threshold. In the end, we used a hybrid approach whereby we obtain noon times from the method that uses derivatives and LOD times from the thresholding method. The net result of this procedure is a set of noon times and LOD for each day from the segment's start in terms of the local clock. Figure 4.6 shows the LOD values obtained for two different node segments after extracting the light patterns.

Solar reconstruction of clocks:

The solar model provides the LOD and noon values in terms of the global clock (LOD_{GT}), while the procedure described in the previous paragraph extracts the LOD and noon values from light sensor measurements in terms of the motes' local clocks (LOD_{LT}). In order to find the best possible day alignment, we look at the correlation between the two LOD signals (LOD_{GT}, LOD_{LT}) as a function of the lag (shift in days). The lag that gives us the maximum correlation (ρ_{max}) is an estimate of the day alignment. Mathematically, the day alignment estimate (lag) is obtained as

$$\arg\max_{lag} \mathrm{Cor}(LOD_{GT}, LOD_{LT}, lag)$$

where $\mathrm{Cor}(X, Y, s)$ is the correlation between time series X and Y shifted by s time units. Figure 4.7 presents an example of the match between model and computed LOD and noon times achieved by the lag with the highest correlation. The computed LOD time series tracks the one given by the solar model. One also observes a constant shift between the two LOD patterns, which can be attributed to the horizon effect. For some days, canopy cover and weather patterns cause the extracted LOD to be underestimated. However, as the day alignment is obtained by performing a cross-correlation with the model LOD pattern, the result is robust to constant shifts. Furthermore, Figure 4.7 shows that the equal and opposite effect of sunrise and sunset detection ensures that the noon estimation in unaffected in the average case.

After obtaining the day alignment, we use the noon information to generate anchor points. Specifically, for each day of the segment we have available to us the noon time in local clock (from the light sensors) and noon time in global clock (using the model). RGTR can then be used to obtain robust values of α and β. This fit is used to reconstruct the global timestamps. As Figure 4.4 suggests, the noon times change slowly over consecutive days as they oscillate

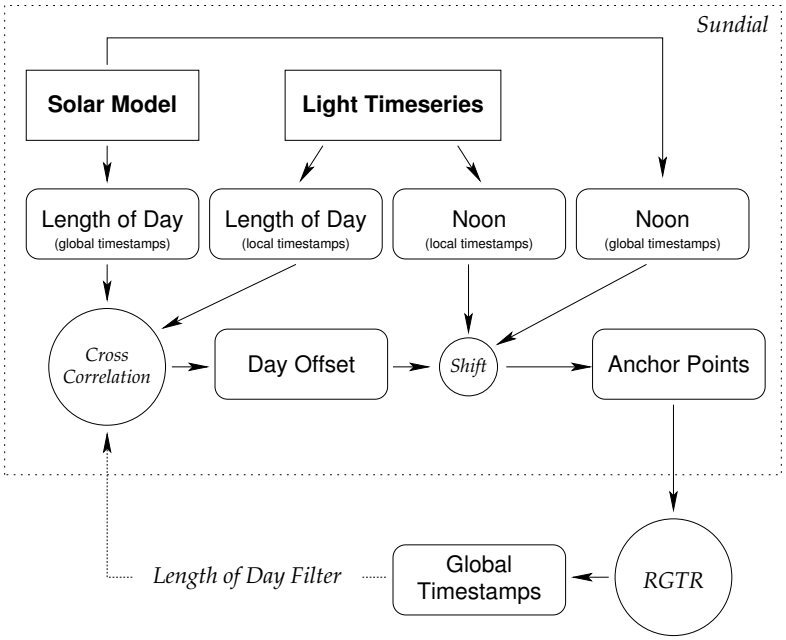


Figure 4.8: The steps involved in reconstructing global timestamps using Sundial.

around 12:00. Thus, even if the day estimate is inaccurate, due to the small difference in noon times, the α estimate remains largely unaffected. This implies that even if the day alignment is not optimal, the time reconstruction within the day will be accurate, provided that the noon times are accurately aligned. The result of an inaccurate lag estimate is that β is off by a value equal to the difference between the actual day and our estimate. In other words, β is off by an integral and constant number of days (without any skew) over the course of the whole deployment period.

We find that this methodology is well suited in finding the correct α. To improve the β estimate, we perform an iterative procedure which works as follows. For each iteration, we obtain the best estimate fit (α, β). We convert the motes' local timestamps into global timestamps using this fit. We then look at the difference between the actual LOD (given by the model) and the current estimate for that day. If the difference between the expected LOD and the estimate LOD exceeds a threshold, we label that day as an outlier. We remove these

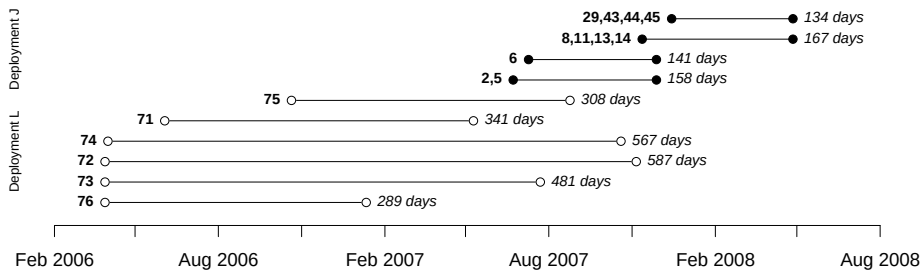


Figure 4.9: Node identifiers, segments and length of each segment (in days) for the two deployments used in the evaluation.

outliers and perform the LOD cross-correlation to obtain the day shift (lag) again. If the new lag differs from the lag in the previous iteration, a new fit is obtained by shifting the noon times by an amount proportional to the new lag. We iterate until the lag does not change from the previous iteration. Figure 4.8 shows a schematic of the steps involved in reconstructing global timestamps for a segment.

4.4 Evaluation

We evaluate the proposed methodology using data from two deployments. Deployment *J* was done at the Jug Bay wetlands sanctuary along the Patuxent river in Anne Arundel County, Maryland. The data it collected is used to study the nesting conditions of the Eastern Box turtle (*Terrapene carolina*) [65]. Each of the motes was deployed next to a turtle nest, whereas some of them have a clear view of the sky while others are under multiple layers of tree canopy. Deployment *L*, from Leakin Park, is described in Section 4.2.3.

Figure 4.9 summarizes the node identifiers, segments, and segment lengths in days for each of the two deployments. Recall that a segment is defined as a block of data for which the mote's clock increases monotonically. Data obtained from the *L* dataset contained some segments lasting well over 500 days. The *L* deployment uses MicaZ motes [17], while the *J*

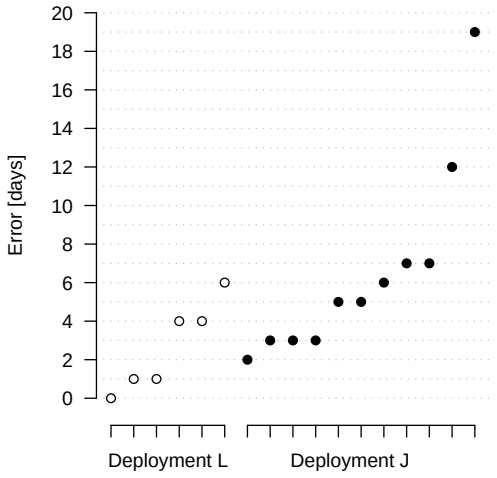


Figure 4.10: **Error in days for different motes from the L and J deployments.**

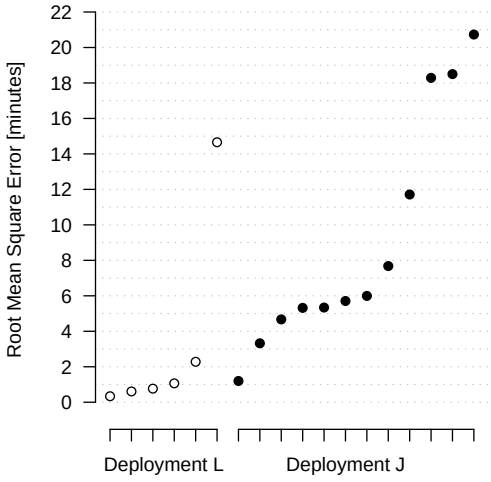


Figure 4.11: **Root mean square error in minutes ($RMSE_{min}$).**

deployment uses TelosB motes [50]. Motes 2, 5, and 6 from Deployment J collected samples every 10 minutes. All other motes for both deployments had a sampling interval of 20 minutes. In addition to its on-board light, temperature, and humidity sensors, each mote was connected to two soil moisture and two soil temperature sensors.

In order to evaluate Sundial's accuracy, we must compare the reconstructed global timestamps it produces, with timestamps that are known to be accurate and precise. Thus we begin our evaluation by establishing the ground truth.

4.4.1 Ground Truth

For each of the segments shown in Figure 4.9, a set of good anchor points (sampled using the basestation) were used to obtain a fit that maps the local timestamps to the global timestamps. We refer to this fit as the *Ground truth fit*. This fit was validated in two ways. First, we correlated the ambient temperature readings among different sensors. We also correlated the motes' measurements with the air temperature measurements recorded by nearby

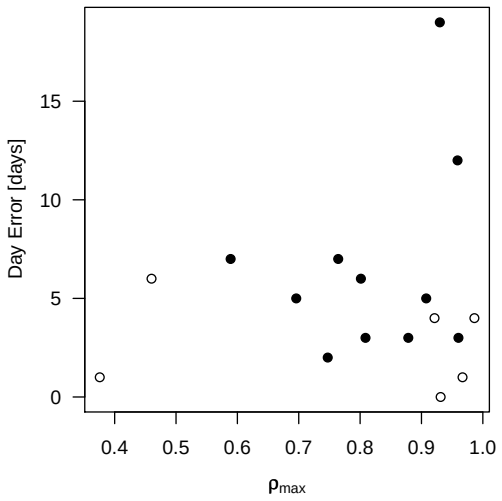


Figure 4.12: Relation between ρ_{max} and error in days.

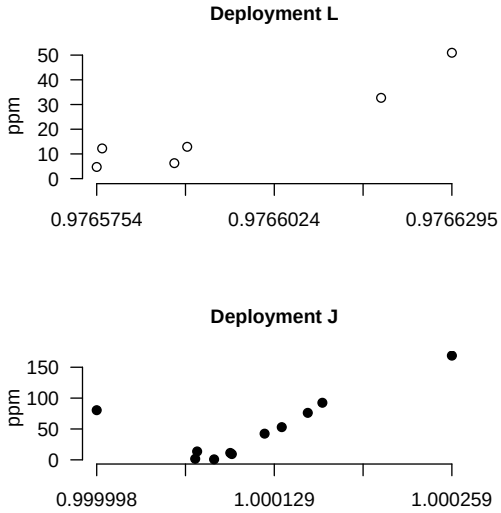


Figure 4.13: α estimates from Sundial and estimation errors in ppm.

weather stations. The weather station for the *L* deployment was located approximately 17 km away from the deployment site [1], while the one for the *J* deployment was located less than one km away [45]. Considering the proximity of the two weather stations we expect that their readings are strongly correlated to the motes' measurements.

Note that even if the absolute temperature measurements differ, the diurnal temperature patterns should exhibit the same behavior thus leading to high correlation values. Visual inspection of the temperature data confirmed this intuition. Finally, we note that due to the large length of the segments we consider, any inconsistencies in the ground truth fit would become apparent for reasons similar to the ones provided in Section 4.2.2.

4.4.2 Reconstructing Global Timestamps using Sundial

We evaluate Sundial using data from the segments shown in Figure 4.9. Specifically, we evaluate the accuracy of the timestamps reconstructed by Sundial as though the start time of these segment is unknown (similar to the case of a mote reboot) and no anchor points

are available. Since we make no assumptions of the segment start-time, a very large model (solar) signal needs to be considered to find the correct shift (lag) for the day alignment.

Evaluation Metrics:

We divide the timestamp reconstruction error to: (a) error in days; and (b) error in minutes within the day. The error in minutes is computed as the root mean square error ($RMSE_{min}$) over all the measurements. We divide the reconstruction error into these two components, because this decoupling naturally reflects the accuracy of estimating the α and β parameters. Specifically, if the α estimate were inaccurate, then, as Figure 4.2 suggests, the reconstruction error would grow as a function of time. In turn, this would result in a large root mean squared error in minutes within the day over all the measurements. On the other hand, a low $RMSE_{min}$ corresponds to an accurate estimate for α. Likewise, inaccuracies in the estimation of β would result in large error in days.

Results:

Figures 4.10 and 4.11 summarize Sundial's accuracy results. Overall, we find that longer segments show a lower day error. Segments belonging to the *L* deployment span well over a year and the minimum day error is 0 while the maximum day error is 6. In contrast, most of the segments for deployment *J* are less than 6 months long and the error in days for all but two of those segments is less than one week. Figure 4.12 presents the relationship between the maximum correlation (ρ_{max}) and the day error. As ρ_{max} measures how well we are able to match the LOD pattern for a node with the solar LOD pattern, it is not surprising that high correlation is generally associated with low reconstruction error. The $RMSE_{min}$ obtained for each of the segments in deployment *L* is very low (see Figure 4.11) . Remarkably, we are able

to achieve an accuracy ($RMSE_{min}$) of under a minute for the majority of the nodes of the *L* deployment even though we are limited by our sampling frequency of 20 minutes. Moreover, $RMSE_{min}$ error is always within one sample period for all but one segment.

Interestingly, we found that the α values for the two deployments were significantly different. This disparity can be attributed to differences in node types and thus clock logic. Nonetheless, Sundial accurately determined α in both cases. Figure 4.13 presents the α values for the two deployments. We also show the error between the α obtained using Sundial and the α value obtained by fitting the good anchor points sampled by the gateway (i.e., ground truth fit). The ppm error for both the deployments is remarkably low and close to the operating error of the quartz crystal.

4.4.3 Impact of Segment Length

Sundial relies on matching solar patterns to the ones observed by the light sensors. The natural question to ask is: what effect does the length of segment have on the reconstruction error. We address this question by experimenting with the length of segments and observing the reconstruction error in days and $RMSE_{min}$. We selected data from three long segments from deployment *L*. To eliminate bias, the start of each shortened segment was chosen from a uniform random distribution. Figure 4.15 shows that the $RMSE_{min}$ tends to be remarkably stable even for short segments. One concludes that even for short segment lengths, Sundial estimates the clock drift (α) accurately. Figure 4.14 shows the effect of segment size on day error. In general, the day error decreases as the segment size increases. Moreover, for segments less than 150 days long, the error tends to vary considerably.

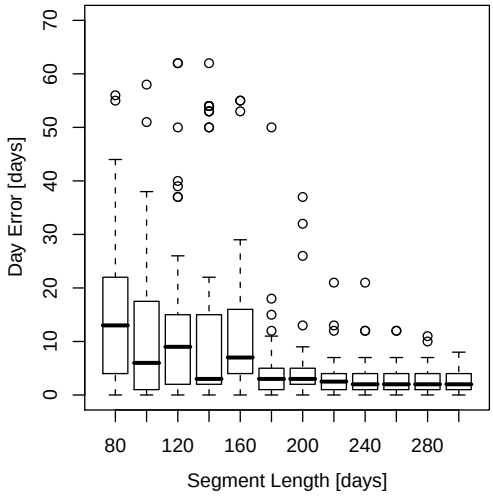


Figure 4.14: **Error in days as a function of segment size.**

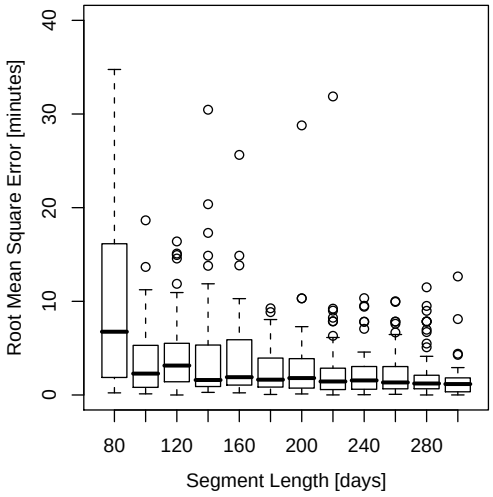


Figure 4.15: **Error in minutes ($RMSE_{min}$) as a function of segment size.**

4.4.4 Day Correction

The results so far show that 88% (15 out of 17) of the motes have a day offset of less than a week. Next, we demonstrate how global events can be used to correct for the day offset. We looked at soil moisture data from eight motes of the *J* deployment after obtaining the best possible timestamp reconstruction. Specifically, we correlated the motes' soil moisture data with rainfall data to correct for the day offset. We used rainfall data from a period of 133 days, starting from December 4, 2007, during which 21 major rain events occurred. To calculate the correlation, we created weighted daily vectors for soil moisture measurements (SM) whose value was greater than a certain threshold and similarly rainfall vectors having a daily precipitation (PPT) value of greater than 4.0 cm. Next, we extracted the lag at which the cosine angle between the two vectors (cosine similarity, θ_{SM-PPT}) is maximum. This method is inspired by the well-known document clustering model used in the information retrieval community [56]. Note that we computed θ_{SM-PPT} for a two-week window ($\pm$ seven days) of lags and found that seven out of the eight motes could be aligned perfectly. Figure 4.16

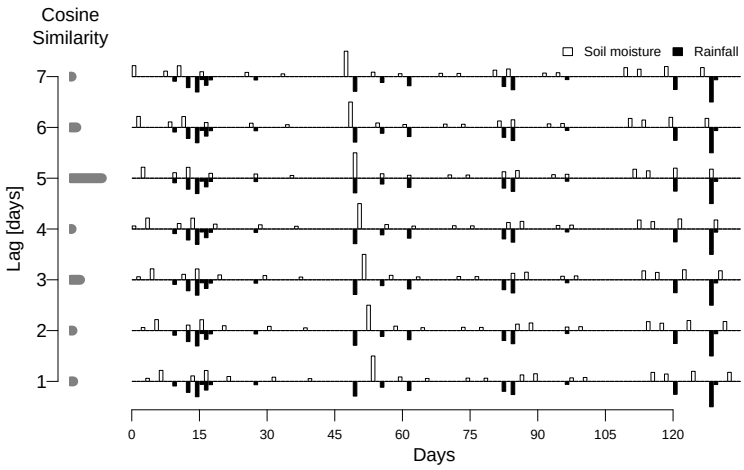


Figure 4.16: An illustration of the cosine similarity (θ_{SM-PPT}) values for seven different day lags between moisture and rainfall vectors. θ_{SM-PPT} peaks at the correct lag of five days, providing the correct day adjustment.

illustrates the soil moisture vectors, rainfall vectors and the associated θ_{SM-PPT} for seven lags for one of the segments. Note that θ_{SM-PPT} peaks at the correct lag of five, leading to the precise day correction. While we use soil moisture to illustrate how global events can be used to achieve macro-level clock adjustments, other modalities can also be used based on the application's parameters.

4.5 Related Work

This study proposes a solution to the problem of postmortem timestamp reconstruction for sensor measurements. To our knowledge, there is little previous work that addresses this problem for deployments that span a year or longer. Deployment length can be an issue because the reconstruction error monotonically increases as a function of time (cf. Sec.4.2.2). The timestamp reconstruction problem was first introduced by Werner-Allen et al. who provided a detailed account of the challenges they faced in synchronizing mote clocks during a 19-day deployment at an active volcano [73]. Specifically, while the system employed the

FTSP protocol to synchronize the network's motes, unexpected faults forced the authors to rely on an offline *time rectification* algorithm to reconstruct global timestamps.

While experiences such as the one reported in [73] provide motivation for an independent time reconstruction mechanism such as the one proposed in this paper, the problem addressed by Werner-Allen et al. is different from the one we aim to solve. Specifically, the volcano deployment had access to precise global timestamps (through a GPS receiver deployed at the site) and used linear regression to translate local timestamps to global time, once timestamp outliers were removed. While RGTR can also be used for outlier detection and timestamp reconstruction, Sundial aims to recover timestamps in situations where a reliable global clock source is not available. A system similar in spirit to Sundial is presented by Lukac et al. [36]. This system achieves temporal integrity by using the data and a model for microseism propagation to time-correct the data collected by their seismic sensors.

Finally, Chang et. al. [13] describe their experiences with motes rebooting and resetting of logical clocks, but do not furnish any details of how they reconstructed the global timestamps when this happens.

4.6 Conclusion

In this paper we present Sundial, a method that uses light sensors to reconstruct global timestamps. Specifically, Sundial uses light intensity measurements, collected by the motes' on-board sensors, to reconstruct the length of day (LOD) and noon time throughout the deployment period. It then calculates the slope and the offset by maximizing the correlation between the measurement-derived LOD series and the one provided by astronomy. Sundial operates in the absence of global clocks and allows for random node reboots. These features make Sundial very attractive for environmental monitoring networks deployed in harsh environments, where they operate disconnected over long periods of time. Furthermore, Sundial

can be used as an independent verification technique along with any other time reconstruction algorithm.

Using data collected by two network deployments spanning a total of 2.5 years we show that Sundial can achieve accuracy in the order of a few minutes. Furthermore, we show that one can use other global events such as rain events to correct any day offsets that might exist. As expected, Sundial's accuracy is closely related to the segment size. In this study, we perform only a preliminary investigation on how the length of the segment affects accuracy. An interesting research direction we would like to pursue is to study the applicability of Sundial to different deployments. Specifically, we are interested in understanding how sampling frequency, segment length, latitude and season (time of year) collectively affect reconstruction accuracy.

Sundial exploits the correlation between the well-understood solar model and the measurements obtained from inexpensive light sensors. In principle, any modality having a well-understood model can be used as a replacement for Sundial. In the absence of a model, one can exploit correlation from a trusted data source to achieve reconstruction, e.g., correlating the ambient temperature measurement between the motes with data obtained from a nearby weather station. However, we note that many modalities (such as ambient temperature) can be highly susceptible to micro-climate effects and exhibit a high degree a spatial and temporal variation. Thus, the micro-climate invariant solar model makes light a robust modality to reconstruct timestamps in the absence of any sampled anchor points.

Finally, we would like to emphasize the observation that most environmental modalities are affected by the diurnal and annual solar cycles and not by the human-created universal time. In this regard, the time base that Sundial establishes offers a more natural reference basis for environmental measurements.

Chapter 5

Time Reconstruction II - Phoenix

One of the big take aways from Sundial was to collect enough number of (local,global) reference points so that all the segments can be assigned a global timestamps. Employing a persistent basestation and performing regular downloads seemed like a reasonable way to achieve this. In the Cub Hill deployment, we put this to practise and it worked out well for the vast majority of segments. One of the things we overlooked was that the basestation is susceptible to failures and performing regular downloads may not always be possible due to power constraints.

In this chapter, I will present the reasons behind why we needed to design a new time reconstruction methodology that does not require a persistent global clock source and can tolerate random mote resets. I'll present a summary of Phoenix at this point and details of the method in the rest of the chapter.

Motes in Phoenix exchange their time-related state with their neighbors, establishing a chain of transitive temporal relationships to one or more motes with references to the global time. These relationships allow Phoenix to reconstruct the measurement timeline for each mote. Results from simulations and a deployment indicate that Phoenix can achieve timing accuracy up to 6 ppm for 99% of the collected measurements. Phoenix is able to maintain this performance for periods that last for months without a persistent global time source.

To achieve this level of performance for the targeted environmental monitoring application, Phoenix requires an additional space overhead of 4% and an additional duty cycle of 0.2%.

5.1 Introduction

Wireless sensor networks have been used recently to understand spatiotemporal phenomena in environmental studies [38, 69]. The data these networks collect are scientifically useful only if the collected measurements have corresponding, accurate global timestamps. The desired level of accuracy in this context is in the order of milliseconds to seconds. In order to reduce complexity of the code running on the mote, it is more efficient to record sensor measurements using the mote's local time frame and perform a postmortem reconstruction to translate them to global time.

Each mote's clock (referred to as local clock henceforth) monotonically increases and resets to zero upon reboot. A naive postmortem time reconstruction scheme collects $\langle local, global \rangle$ pairs during a mote's lifetime, using a global clock source (typically, an NTP-synchronized PC). These pairs (also referred to as "anchor points") are then used to translate the collected measurements to the global time frame by estimating the motes' clock skew and offset. We note that this methodology is unnecessary for architectures such as Fleck, which host a battery-backed on-board real-time clock (RTC) [15]. However, many commonly-used platforms such as Telos, Mica2, MicaZ, and IRIS (among others) lack an on-board RTC.

In the absence of reboots, naive time reconstruction strategies perform well. However, in practice, motes reboot due to low battery power, high moisture, and software defects. Even worse, when motes experience these problems, they may remain completely inactive for non-deterministic periods of time. Measurements collected during periods which lack $\langle local, global \rangle$ anchors (due to rapid reboots and/or basestation absence) are difficult or impossible to accurately reconstruct. Such situations are not uncommon based on our deployment

experiences and those reported by others [73].

In this work, we devise a novel time reconstruction strategy, *Phoenix*, that is robust to random mote reboots and intermittent connection to the global clock source. Each mote periodically listens for its neighbors to broadcast their local clock values. These $\langle local, neighbor \rangle$ anchors are stored on the mote's flash. The system assumes that one or more motes can periodically obtain global time references, and they store these $\langle local, global \rangle$ anchors in their flash. When the basestation collects the data from these motes, an offline procedure converts the measurements timestamped using the motes' local clocks to the global time by using the transitive relationships between the local clocks and global time.

The offline nature of Phoenix has two advantages: (a) it reduces the complexity of the software running on the mote, and (b) it avoids the overhead associated with executing a continuous synchronization protocol. We demonstrate that Phoenix can reconstruct global timestamps accurately (within seconds) and achieve low ($< 1\%$) data losses in the presence of random mote reboots even when months pass without access to a global clock source.

5.2 Motivation

We claim that the problem of rebooting motes is a practical aspect of real deployments that has a high impact on environmental monitoring applications. We also quantify the frequency and impact of reboots in a long-term deployment. We begin by understanding why mote reboots complicate postmortem time reconstruction.

5.2.1 Postmortem Timestamp Reconstruction

The relationship between a mote's local clock, LTS, and the global clock, GTS, can be modeled with a simple linear relation: $GTS = \alpha \times LTS + \beta$, where α represents the mote's skew and β

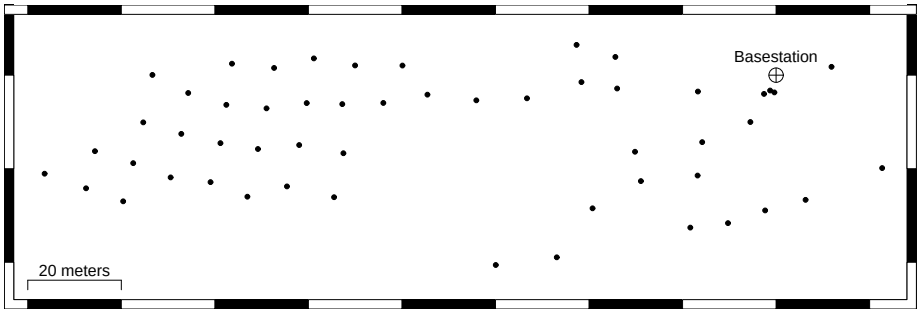


Figure 5.1: The 53-mote "Cub Hill" topology, located in an urban forest northeast of Baltimore, Maryland.

represents the intercept (global time when the mote reset its clock) [55]. This conversion from the local clock to global clock holds as long as the mote's local clock monotonically increases at a constant rate. We refer to this monotonically increasing period as a *segment*. When a mote reboots and starts a new segment, one needs to re-estimate the fit parameters. If a mote reboots multiple times while it is out of contact with the global clock source, estimating β for these segments is difficult. While data-driven treatments have proven useful for recovering temporal integrity, they cannot replace accurate timestamping solutions [27, 36]. Instead, time reconstruction techniques need to be robust to mote reboots and not require a persistent global time source.

5.2.2 Case Studies

We present two cases which illustrate the deployment problems that Phoenix intends to address. The first is an account of lessons learned from a year-long deployment of 53 motes. The second is a result of recent advances in solar-powered sensor networks.

Software Reboots. We present "Cub Hill", an urban forest deployment of 53 motes that has been active since July 2008 (Figure 5.1). Sensing motes collect measurements every 10

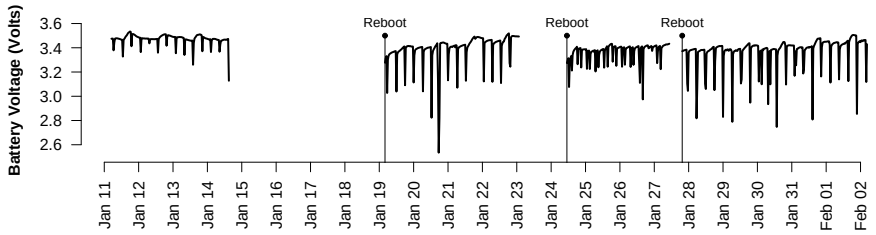


Figure 5.2: An example of a mote rebooting due to low battery voltage (no watchdog timer in use). The sharp downward spikes correspond to gateway downloads (every six hours). Gaps in the series are periods where the mote was completely inoperative.

minutes to study the impact of land use on soil conditions. The basestation uses the Koala protocol to collect data from these motes every six hours [44]. We use TelosB motes driven by 19 Ah, 3.6 V batteries.

We noticed that motes with low battery levels and/or high internal moisture levels suffered from periodic reboots. As an example, Figure 5.2 shows the battery voltage of a mote that rebooted thrice in one month. Despite their instability, many of these motes were able to continue collecting measurements for extended periods of time.

Following a major network expansion, a software fault appeared which caused nodes to "freeze". Unable to reproduce this behavior in a controlled environment, we employed the MSP430's Watchdog Timer to reboot motes that enter this state [68]. While this prevented motes from completely failing, it also shortened the median length of the period between reboots from more than 50 days to only four days, as Figure 5.3 shows.

Solar Powered Sensor Networks. A number of research groups have demonstrated the use of solar energy as a means of powering environmental monitoring sensor networks [37, 66]. In such architectures, a mote can run out of power during cloudy days or at night. Motes naturally reboot in such architectures, and data losses are unavoidable due to the lack of energy. It is unclear how one can achieve temporal reliability without a persistent

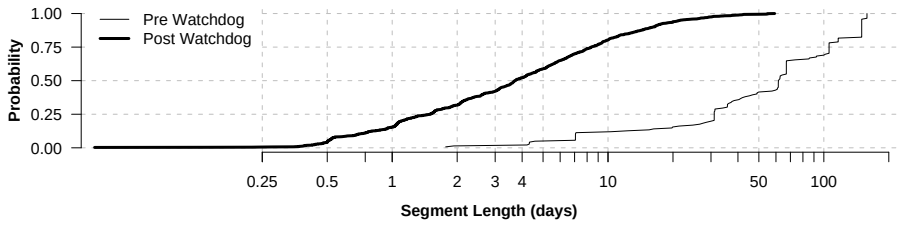


Figure 5.3: The distribution of the segment lengths before and after adding the watchdog timer to the mote software.

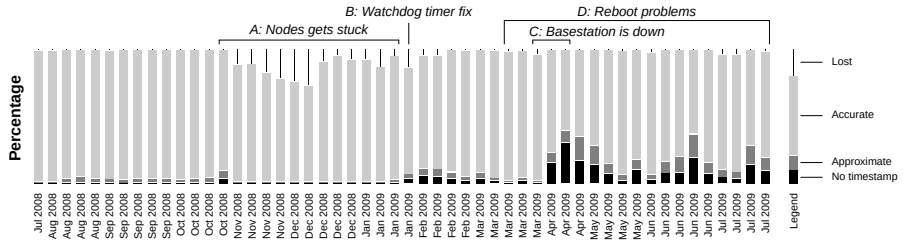
basestation or an on-board RTC. To the best of our knowledge, no one has addressed the issue of temporal integrity in solar-powered sensor networks. Yang et al. employ a model in which data collection happens without a persistent basestation [75]. The data upload takes place infrequently and opportunistically. Hard-to-predict reboot behavior is common to these systems. Furthermore, we note that even though there is very little information about the rate of reboots in such architectures, it is clear that such systems are susceptible to inaccurate timestamp assignments.

5.2.3 Impact

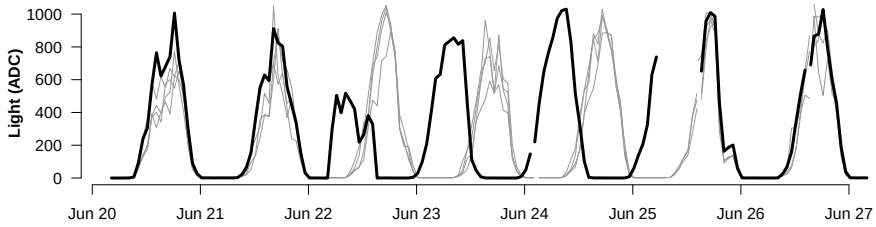
We evaluate the impact of mote reboots on the Cub Hill deployment using our existing time reconstruction methodology.

The basestation records an anchor point each time it downloads data from a mote. Motes that are poorly connected to the basestation may remain out of contact for several download rounds before connectivity improves and they can transfer their data. When motes reboot at a rate faster than the frequency with which the basestation contacts them, there exist periods which lack enough information to accurately reconstruct their measurement timestamps.

Upon acquiring the anchor points, the measurements are converted from their local clock to the global clock at the basestation. We employ our previously proposed algorithm, Robust



(a) The fraction of measurements that were assigned timestamps.



(b) An example of the impact of estimating β incorrectly when using approximate methods. Data from one of the motes (represented with the dark line) that rebooted multiple times between Jun. 22 and Jun. 25. During this period, the mote was out of sync with the rest (shown in gray) due to inaccurate β estimates

Figure 5.4: Impact of time reconstruction methodology using the RGTR algorithm.

Global Timestamp Reconstruction algorithm (referred to as RGTR), for this purpose [27]. We note that in order to estimate the fit parameters (α, β) for the segments, RGTR requires at least two anchor points. Depending on the accuracy requirements, one can assume that the skew (α) is stable per mote for small segments. Using this assumption, at least one anchor point is needed to estimate the β for any given segment, provided that α has been estimated accurately for the mote.

Figure 5.4(a) demonstrates the impact of mote reboots on time reconstruction for the Cub Hill deployment. During period A, motes were prone to freezing (and thus stopped sampling), leading to a decrease in the total data collected. At point B, the addition of the watchdog timer caused the total data collected to return to its previous level. However, due to the increased frequency of reboots, a larger portion of the samples could not be assigned a global timestamp (exacerbated by the absence of the base station during period C).

For segments where no anchor points were collected, we assumed that node reboots are

instantaneous. However, this assumption does not always hold (see Figure 5.2) and leads to a small fraction of misaligned measurements. Figure 5.4(b) presents an example of this misalignment. One node (shown in bold) rebooted multiple times and could not reach the basestation during its active periods. The assumption of instantaneous reboots led to inaccurate β estimates.

5.3 Solution

Phoenix is a postmortem time reconstruction algorithm for motes operating without in-network time synchronization. It consists of two stages.

5.3.1 In-Network Anchor Collection

Each mote operates solely with respect to its own local clock. A new segment (uniquely identified by $\langle moteid, reboot\ count \rangle$) begins whenever a mote reboots: each segment starts at a different time and may run at a different rate. Our architecture assumes that there is at least one mote in the network that can periodically obtain references from an accurate global time source. This is done to establish the global reference points needed by Phoenix. This source may be absent for long periods of time (see Section 5.4). The global time source can be any reliable source (a mote equipped with a GPS receiver, NTP-synced basestation, etc). Without loss of generality, we assume that the network contains a mote connected to GPS device and a basestation that collects data infrequently[1].

All motes (including the GPS-connected mote) broadcast their local clock and reboot-count values every T_{beacon} seconds. Each receiving mote stores this information (along with its own local clock and reboot counter) in flash to form anchor records. The format of these records is $\langle moteid_r, rc_r, lc_r, moteid_s, rc_s, lc_s \rangle$; where $rc, lc, r,$ and s refer to the reboot counter, local

[1]Note that the basestation collects data *only* and it does not provide a time source, unless specified otherwise.

clock, receiver and sender respectively. Periodically, motes turn on their radios and listen for broadcasts in order to anchor their time frame to those of their neighbors. Each mote tries to collect this information from its neighbors after every reboot and after every T_{wakeup} seconds ($\gg T_{beacon}$). The intuition behind selecting this strategy is as follows. The reboot time determines the β parameter. The earliest opportunity to extract this information is immediately after a reboot. To get a good estimate of the skew, one would like to collect multiple anchors that are well distributed in time. Thus, T_{wakeup} is a parameter that governs how far to spread out anchor collections. In the case of a GPS mote, the $moteid_r, rc_r$ and $moteid_s, rc_s$ are identical, and lc_r, lc_s represent the local and global time respectively.

The basestation periodically downloads these anchors along with the measurements. This information is then used to assign global timestamps to the collected measurements using Algorithm 2. If the rate of reboots is known, the anchor collection frequency can be fixed conservatively to collect enough anchors between reboots. One could also employ an adaptive strategy by collecting more anchors when the segment is small and reverting to a larger T_{wakeup} when an adequate number of anchors have been collected. It is advantageous for a mote to attempt to collect anchors from a small set of neighbors (to minimize storage), but this requires a mote to have some way of identifying the most useful segments for anchoring (see Section 5.4).

5.3.2 Offline Timestamp Reconstruction

The Phoenix algorithm is intuitively simple. We will outline it in text and draw attention to a few important details. For a more complete treatment, please refer to the pseudocode in Algorithm 2. Phoenix accepts as input the collection of all anchor points AP (both $\langle local, neighbor \rangle$ and $\langle local, global \rangle$). It then employs a least-square linear regression to extract the relation-

Algorithm 2 Phoenix

Ensure:
a, b : alpha and beta for local-local fits;
P : parent segment; Π : Ancestor segments

procedure PHOENIX(AP)
  for each (i, j) in KEYS(AP) **do** ▷ All unique segment pairs in AP
    $LF_{a,b,\chi,df}(i, j) \leftarrow$ LLSE($AP(i, j)$) ▷ Compute the local-local fits
  for each $s \in S$ **do** ▷ Set of all unique segments
    $GF_{\alpha,\beta,P,\Pi,\chi,df}(s) \leftarrow (\emptyset, \emptyset, \emptyset, s, \chi_{MAX}, \emptyset)$ ▷ Initialize global fits
  for each $g \in G$ **do** ▷ All segments anchored to GTS
    INITGTSNODES(g, LF, GF)
    ENQUEUE(Q, g) ▷ Add all the GTS nodes to the queue
  while NOTEMPTY(Q) **do**
    $q \leftarrow$ DEQUEUE(Q)
    $C \leftarrow$ NEIGHBORANCHORS(q)
    for each $c \in C$ **do**
      $T_{\alpha,\beta,P,\Pi,\chi,df}(c) \leftarrow$GLOBALFIT($c, q, GF, LF$)
      if (UPDATEFIT(c, T, GF)) **then** ▷ Check for a better fit
        ENQUEUE(C)
  return GF

procedure INITGTSNODES(g, LF, GF)
  $GF(g) \leftarrow (LF_a(g, g'), LF_b(g, g'), \emptyset, g, LF_\chi(g, g'), LF_{df}(g, g'))$ ▷ g' is GTS, g is LTS

procedure GLOBALFIT(c, q, GF, LF)
  if $q > c$ **then** ▷ Smaller segment is the independent variable
    $\alpha_{new} \leftarrow GF_\alpha(q) * LF_a(q, c)$
    $\beta_{new} \leftarrow GF_\alpha(q) * LF_b(q, c) + GF_\beta(q)$
  else
    $\alpha_{new} \leftarrow GF_\alpha(q)/LF_a(q, c)$
    $\beta_{new} \leftarrow GF_\alpha(q) - \alpha_{new} * LF_b(q, c)$
  $\chi \leftarrow \frac{GF_{df}(q)*GF_\chi(q)+LF_{df}(q,c)*LF_\chi(q,c)}{GF_{df}(q)+LF_{df}(q,c)}$ ▷ Compute the weighted GOF metric.
  $df \leftarrow GF_{df}(q) + LF_{df}(q, c)$
  return $(\alpha_{new}, \beta_{new}, q, \{c \cup GF_\Pi(q)\}, \chi, df)$ ▷ Update parent/ancestors

procedure UPDATEFIT(c, T, GF)
  if $c \in T_\Pi(c)$ **then** ▷ Check for cycles
    return false
  if $T_\chi(c) < GF_\chi(c)$ **then**
    $GF_{\alpha,\beta,P,\Pi,\chi,df}(c) \leftarrow T_{\alpha,\beta,P,\Pi,\chi,df}(c)$
    return true
  else
    return false

ships between the local clocks of the segments that have anchored to each other (LF, for Local Fit). In addition to $LF_a(i, j)$ (slope), $LF_b(i, j)$ (intercept), Phoenix also obtains a goodness-of-fit (GOF) metric, $LF_\chi(i, j)$ (unbiased estimate of the variance of the residuals) and LF_{df} (degrees of freedom). For segments which have global references, Phoenix stores this as GF (for Global Fit).

The algorithm then initializes a queue with all of the segments which have direct anchors to the global clock. It dequeues the first element q and examines each segment c that has anchored to it. Phoenix uses the transitive relationship between $GF(q)$ and $LF(q, c)$ to produce

a global fit $T(c)$ which associates segment c to the global clock through segment q. If $T_\chi(c)$ is lower than the previous value for $GF_\chi(c)$ (and using q would not create a cycle in the path used to reach the global clock), the algorithm replaces $GF(c)$ with $T(c)$, and places c in the queue. When the queue is empty, no segments have "routes" to the global clock which have a better goodness-of-fit than the ones which have been previously established. At this point, the algorithm terminates.

The selection of paths from an arbitrary segment to a segment with global time references can be thought of as a shortest-path problem (each segment represents a vertex and the fit between the two segments is an edge). The GOF metric represents the edge weight. The running time complexity of the implementation of Phoenix was validated experimentally by varying the deployment lifetime (thereby varying number of segments). The runtime was found to increase slower than the square of the number of segments.

5.4 Evaluation

We evaluate the effect of varying several key parameters in Phoenix using both simulated and real datasets. We begin by describing our simulator.

5.4.1 Simulator

Our goal is to minimize the data loss in long-term deployments. Hence, we fix the simulation period to be one year. We also assume that the basestation is not persistently present and does not provide a time source to the network. The network contains one global clock source (a GPS mote) that is susceptible to failures. The main components of the simulator are described below. The default values for the simulator are based on empirical data obtained from the one year long Cub Hill deployment.

Clock Skew: The clock skew for each segment is drawn from a uniformly distributed random variable between 40 ppm and 70 ppm. Burri et al. report this value to be between 30 and 50 ppm at room temperature[2] [10].

Segment Model: We use the non-parametric segment-length model based on the Cub Hill deployment after the watchdog timer fix (Figure 5.3). Additionally, after a reboot, we allowed the mote to stay inactive for a period that is randomly drawn between zero and four hours with a probability given by $p_{down} = 0.2$. The GPS mote's behavior follows the same model.

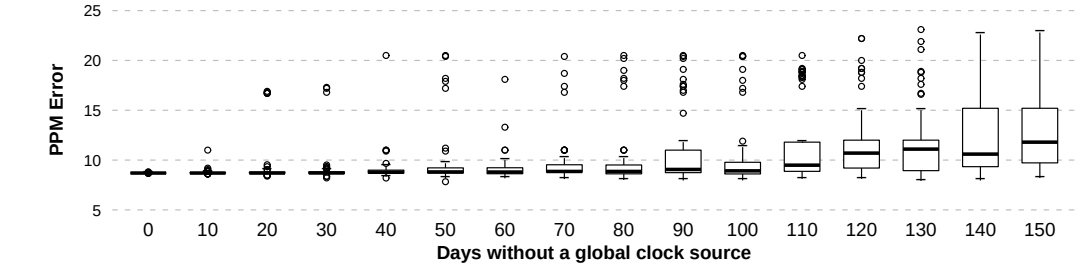
Communication Model: The total end-to-end communication delay for receiving anchor packets is drawn uniformly between 5 and 15 milliseconds. This time includes the interrupt handling, transmission, reception and propagation delays. To model the packet reception rate (PRR), we use the log-distance path loss model as described in [53, 76] with parameters: $(P_r(d_0), \eta, \sigma, d_0) = (-59.28, 2.04, 6.28, 2.0m)$.

Topology: The Cub Hill topology was used as the basis for all simulations.

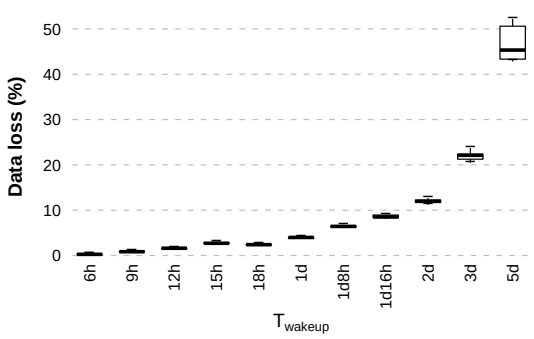
Event Frequencies: Motes recorded a 26-byte sample every 10 minutes. They beacon their local clock values with an interval of T_{beacon}. They stay up after every reboot and periodically after an interval of T_{wakeup} to collect these broadcasts. While up, they keep their radios on for a maximum of T_{listen}. The GPS mote collects $\langle local, global \rangle$ anchors with a rate of T_{sync}. By default, T_{beacon}, T_{wakeup}, T_{listen} and T_{sync} were set to 30 s, 6 h, 30 s and 6 h respectively.

Maximum Anchorable Segments: To minimize the space overhead in storing anchors, we limit the number of segments that can be used for anchoring purposes. At any given time, a mote can only store anchors for up to $NUMSEG$ segments. The default $NUMSEG$ value is set to four. Motes stop listening early once they collect $NUMSEG$ anchors in a single interval.

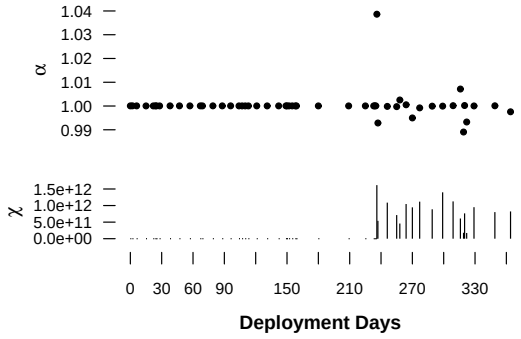
[2]We ignore the well-studied temperature effects on the quartz crystal. For a more complete treatment on the temperature dependence, refer to [41, 46].



(a) The effect of a missing global clock source on accuracy.



(b) The impact of T_{wakeup} on data loss.



(c) Robustness to bad anchors.

Figure 5.5: Evaluation of Phoenix in simulation. In (c), faults were injected to GPS anchors after day 237. Figure shows the α and χ values for the GPS mote for the entire period.

Eviction Policy: Since segments end and links between motes change over time, obsolete or rarely-heard segments need to be evicted from the set of $NUMSEG$ segments for which a mote listens. The timeout for evicting stale entries is set to $3 \times T_{wakeup}$. We evaluated three different strategies for selecting replacements for evicted segments. First-come, first-served (FCFS) accepts the first segment that is heard when a vacancy exists. RAND keeps track of the previous segments that were heard and selects a new segment to anchor with at random. Longest local clock (LLC) keeps track of the local clock values of the segments that are heard and selects the segment that has the highest local clock. FCFS was chosen as the default.

5.4.2 Evaluation metrics

Data loss (DL): The fraction of data that cannot be assigned any timestamps, expressed as a percentage.

PPM Error: The average error (in parts per million) for the assigned timestamps. PPM error is $\frac{|t'-t|}{t_\delta} \times 10^6$, where t is the true timestamp of the measurement, t' is the assigned timestamp, and t_δ denotes the elapsed time since the start of the segment in terms of the real clock.

Space overhead: The fraction of space that is used for storing anchors relative to the total space used, expressed as a percentage.

Duty cycle: The fraction of time the radio was kept on for anchor collection and beaconing, expressed as a percentage.

5.4.3 Simulation Experiments

Dependence on Global Clock Source: We studied the effect of the global clock's absence on data loss. We assume that the network contains one GPS mote that serves as the global clock source and it is inoperative for a specified amount of time. In order to avoid bias, we randomly selected the starting point of this period and varied the GPS down time from 0 to 150 days in steps of 10. Figure 5.5(a) shows the effect on the reconstruction using 60 independent runs. The accuracy decreases as the number of days without GPS increases, but we note that this decrease is tolerable for our target applications. The data loss stayed relatively stable at 0.21%, even when the global clock source is absent for as long as 5 months. We note that in a densely connected network, the number of paths between any two segments is

combinatorial, and hence, the probability of finding a usable path is very high[3]. The variance of the error increased with the length of the gateway's absence.

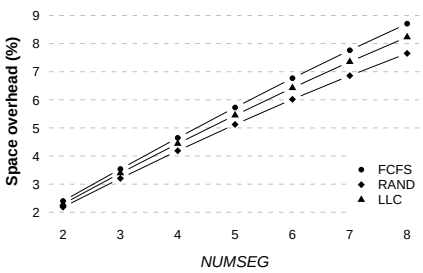
Dependence on Wake-up Interval: Figures 5.5(b) show the effect of varying wake-up rate on data loss. As expected, data loss increases as the rate of anchor collection decreases. This curve is strongly related to the segment model: if collections are less frequent than reboots, many segments will fail to collect enough anchors to be reconstructed.

Robustness: We studied the effect of faulty global clock references on time reconstruction. Noise from a normal distribution ($\mu = 60$ min., $\sigma =$10 min.) was added to the global references for a period of 128 days. Figure 5.5(c) shows the alpha and χ values for the GPS mote during the entire simulation period. One can also notice the correlation between high χ values and α values that deviate from 1.0 in Figure 5.5(c). These faults did not change the data loss rate. The faults increased the PPM error from 4.03 to 16.5. Although these faults decreased accuracy, this decrease is extremely small in comparison to the magnitude of the injected errors and within the targeted accuracy requirements. Phoenix extracted paths which were least affected by these faults by using the χ metric.

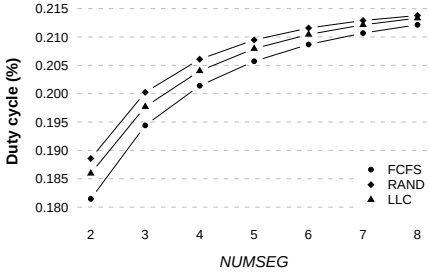
Effect of eviction and $NUMSEG$: We studied the effect of $NUMSEG$ on space, duty cycle, and data loss. The space overhead increases linearly with $NUMSEG$ (Figure 5.6(a)). The impact on duty cycle[4] was quite low (Figure 5.6(b)). A constant duty cycle penalty of 0.075% is incurred due to the beaconing messages sent every 30 s [44]. At low values of $NUMSEG$, motes are able to switch off their radios early (once they have heard announcements from segments they have anchored with), while at higher values, they need to stay on for the

[3]One can estimate the probability for finding a usable path using Warshall's algorithm [16]. The input to this algorithm would be a connectivity matrix where the entries represent the anchoring probabilities of the neighbor segments.

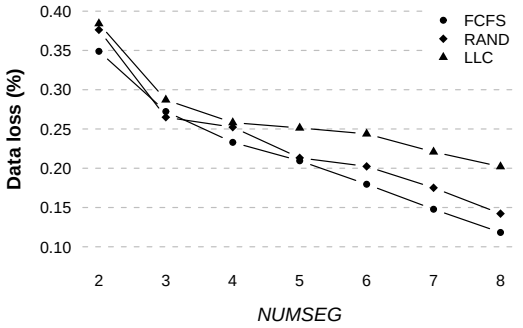
[4]Note that the duty cycle that we are referring to does not consider the communication costs during data downloads. Reducing the storage requirements would reduce the communication costs when the basestation collects data.



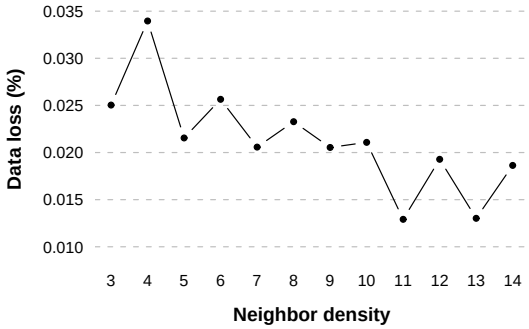
(a) Space overhead in storing anchors as a function of $NUMSEG$.



(b) Duty cycle as a function of $NUMSEG$.



(c) Data loss as a function of $NUMSEG$.



(d) Effect of varying node density on data loss with no eviction policy.

Figure 5.6: Effect of $NUMSEG$ on different eviction policies.

entire T_{listen} period. Increasing $NUMSEG$ decreases data loss, because motes have a better chance of collecting good segments to anchor with. We found that the FCFS eviction policy outperforms LLC and RAND. We found no significant differences in the PPM error results as we vary $NUMSEG$, and hence, we do not report those results here.

Neighbor Density: In this experiment, we removed links from the Cub Hill topology until we obtained the desired neighbor density. At every step, we ensured that the network was fully connected. We did not find any significant impact on performance as the average number of neighbors was decreased. In this experiment, the radios were kept on for the entire T_{listen} period, and no eviction policy was employed. This was done to compare the performance at each density level at the same duty cycle. Figure 5.6(d) presents our findings.

5.4.4 Deployment - I

We deployed a network (referred to as the "Olin" network) of 19 motes arranged in a grid topology in an urban forest near the Johns Hopkins University campus in Baltimore, MD. Anchors were collected for the entire period of 21 days using the methodology described in Section 5.3.1. The basestation collected data from these motes once every four hours and the NTP-corrected clock of the basestation was used as a reliable global clock source. The motes rebooted every 5.7 days on average, resulting in a total of 62 segments. The maximum segment length was 19 days and the minimum was two hours.

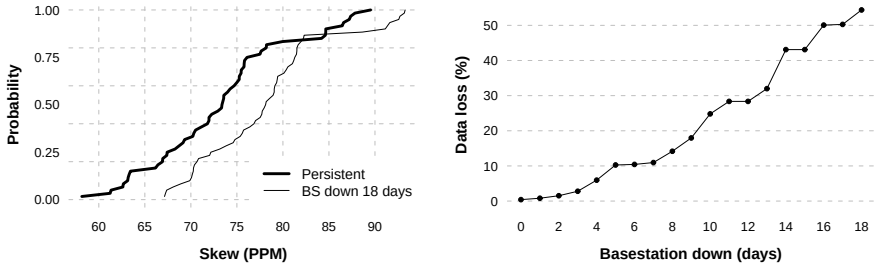
Perceived Ground Truth: It is very difficult to establish absolute ground truth in field experiments. Instead, we establish a synthetic ground truth by reconstructing timestamps using all the global anchors obtained from the basestation[5]. We record the α and β values for each segment and use these values as ground truth. Because we downloaded data every four hours we obtained enough global anchors from the motes to be confident with the derived ground truth estimates.

Emulating GPS node and Basestation Failure: In order to emulate a GPS mote, we selected a single mote (referred to as G-mote) that was one hop away from the basestation. We used the G-mote's global anchors obtained from the basestation as though they were taken using a GPS device. We ignored all other global anchors obtained from other motes. Furthermore, to emulate the absence of the basestation for N days, we discarded all the anchors taken by the G-mote during that N-day long period. We tested for values of N from one to eighteen.

[5]Note that every time a mote contacts the basestation, we obtain a global anchor for that mote.

Table 5.1: Phoenix accuracy using the Olin dataset as a function of the number of days that the basestation was unavailable.

Error\Days	2.4	6.8	10.12	14.16	18.				
α_{med} (ppm)	1.73	1.73	1.85	1.70	1.96	2.20	4.36	5.47	5.93
α_{std} (ppm)	3.41	3.40	3.40	3.39	3.30	3.26	3.17	3.00	3.00
β_{med} (s)	0.88	0.88	0.91	0.94	1.16	1.55	4.52	6.02	6.44
β_{std} (s)	0.58	0.57	0.58	0.57	0.65	0.91	2.43	3.11	3.45



(a) The CDF of α estimates on the Olin deployment

(b) Data loss using RGTR. Data loss from Phoenix was $< 0.06\%$.

Figure 5.7: The stability of the α estimates using Phoenix and the data loss using RGTR in comparison to Phoenix.

Phoenix Accuracy: After simulating the basestation failure, we reconstruct the timestamps by applying Phoenix using only the $\langle local, neighbor \rangle$ anchors, and global anchors available from the G-mote. This provides us with another set of α and β estimates for each of the segments. We compare these estimates with the ground truth estimates (pair-wise comparison). In order to provide a deeper insight, we decompose the average PPM error metric into its constituent components - α and β errors. Furthermore, we report the median and standard deviation of these α and β errors. Table 5.1 reports the results of these experiments. We found that the median α error stayed as low as 5.9 ppm, while the median β error stayed as low as 6.4 s for N =18. In general, α_{med}, β_{med} and β_{std} increased as N increased and α_{std} stayed relatively consistent for different values of N. The stability of the α estimates using Phoenix with $N = 0$ and $N = 18$ is shown in Figure 5.7(a). The CDF shows that median skew

was found to be around 75 ppm and the two curves track each other closely.

Data Loss: The data loss using Phoenix was found to be as low as 0.055% when N was 18 days. In comparison, we found that there was significant data loss when the timestamps were reconstructed using RGTR. Figure 5.7(b) shows the data losses for different values of N. The figure does not report the Phoenix data loss as we found it to be 0.055% irrespective of N. This demonstrates that Phoenix is able to reconstruct more than 99% of the data even when motes reboot frequently and the basestation is unavailable for days. We note that in comparison to Phoenix, RGTR does not incur any additional storage and duty cycle overheads as anchors are recorded at the basestation directly as part of the data downloads.

5.4.5 Deployment - II

The second deployment (termed Brazil) was at the Nucleo Santa Virginia research station in the Atlantic coastal rain forest near Sao Paolo, Brazil [11]. The goal of this deployment was to collect data to improve atmospheric micro-front models. 52 nodes were deployed for a total of 35 days and $5,418,074$ data points were produced during this campaign. The site could not host a persistent basestation. Instead, researchers would download data every alternate data using a laptop that served as a temporary mobile basestation. The basestation was running a linux VM over windows 7 - our download protocol required a linux installation.

Deployment Setup: Two GPS receivers were built on two motes and these were to serve as the global clock source. These motes would advertise their local clock values for others to anchor with and would also periodically store the (local,GPS) timestamps on their flash - this is in addition to storing time state announcements from other motes. However, due to the lithium battery shipping problems, these GPS motes were unavailable until 22 days into the

Figure 5.8: Data loss due to timestamping for the motes in the Brazil deployment

deployment. Due to these problems, we had to use the laptop's VM clock as the global clock source for the first 22 days. After the batteries arrived, we found out that one of the GPS receivers did not work.

Experiences: When we looked at the temperature time series plots of the reconstructed data for the first few days, we found a few motes "shifted" and "out-of-sync" from one another. The peaks and troughs in the temperature seemed lagged at a few sensors. The motes initially started in-sync and then gradually went out of sync. This indicated to us that some of the motes had poor estimates of α. On further investigation, we realized that the VM clock was highly unstable and this lead to poor reconstruction.

Our only hope was to then rely on using the GPS anchors, available from day 22 to day 35 collected by the one working GPS mote. Even though we did not intend things to go this way, this situation was exactly what phoenix was designed for - tolerance to a missing global clock source for extended periods of time.

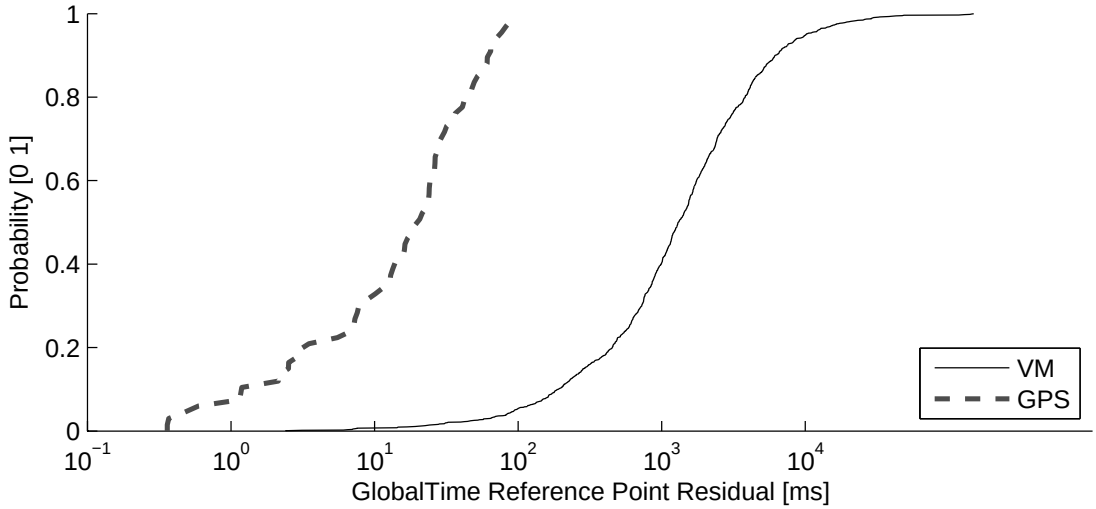


Figure 5.9: Residuals of fits with global time references for the VM clock and the GPS.

Results: Using the GPS anchors, Phoenix was able to timestamp 99.7% of all the data that was collected. The data loss due to timestamping for all the motes in the Brazil deployment is shown in Figure 5.8. Other than mote 41 and 46, more than 70% of the motes have less than 0.1% of timestamping data loss.

The accuracy of these timestamps is difficult to report since ground truth was not available to us. Nonetheless, we can compare the relative quality of the two global clock sources. The CDF of the residuals for the fits obtained using the VM clock and the GPS clock is shown in Figure 5.9. Note that low residuals indicate a good linear fit between mote clocks and the reference clock. By looking at the distribution in Figure 5.9, the median residual for GPS is almost two orders of magnitude lower than the VM clock. One can also notice the long tail (high errors) in the distribution of the VM residuals. The effect of temperature on the mote clock and non-deterministic delays in the GPS interrupt handling account for variation in the GPS residuals. An obvious, but often overlooked, take away from this experience is to ensure that the global clock source is trustworthy and accurate - just having one is not good enough.

5.5 Related Work

Assignment of timestamps in sensor networks falls under two broad categories. Strict clock synchronization aims at ensuring that all the mote clocks are synchronized to the same clock source. Flooding Time Synchronization Protocol (FTSP, [40]), Reference Broadcast Synchronization (RBS, [22]), and the Timing-sync Protocol for Sensor Networks [25] are examples of this approach. These systems are typically used in applications such as target tracking and alarm detection which require strong real-time guarantees of reporting events. The second category is known as postmortem time reconstruction and it is mostly used due to its simplicity. While strict synchronization is appropriate for applications where there are specific events of interest that need to be reported, postmortem reconstruction is well-suited for applications where there is a continuous data stream and every measurement requires an accurate timestamp.

Phoenix falls under the second class of methods. The idea of using linear regression to translate local timestamps to global timestamps was first introduced by Werner-Allen et al. in a deployment that was aimed at studying active volcanoes [73]. This work, however, does not consider the impact caused by rebooting motes and basestation failures from a time reconstruction perspective. More recently, researchers have proposed data-driven methods for recovering temporal integrity [27, 36]. Lukac et al. use a model for microseism propagation to time-correct the data collected by their seismic sensors. Although data-driven methods have proved useful for recovering temporal integrity, they are not a solution for accurate timestamping.

Routing integrated time synchronization protocol (RITS, [55]) spans these categories. Each mote along the path (to the basestation) transforms the time of the reported event from the preceding mote's time frame, ending with an accurate global timestamp at the

basestation. RITS does not consider the problem of mote reboots, and is designed for target tracking applications. The problem of mote reboots have been reported by a number of research groups. Chang et al. report that nodes rebooted every other day due to an unstable power source [13], whereas Dutta et al. employed the watchdog timer to reboot nodes due to software faults [21]. Allen et al. report an average node uptime of 69% [73]. More recently, Chen et al. advocate *Neutron*, a solution that detects system violations and recovers from them without having to reboot the mote [14]. They advocate the notion of preserving "precious" states such as the time synchronization state. Nevertheless, Neutron cannot prevent all mote reboots and therefore Phoenix is still necessary.

5.6 Conclusions

In this paper we investigate the challenges facing existing postmortem time reconstruction methodologies due to basestation failures, frequent random mote reboots, and the absence of on-board RTC sources. We present our time reconstruction experiences based on a year-long deployment and motivate the need for robust time reconstruction architectures that minimize data losses due to the challenges we experienced.

Phoenix is an offline time reconstruction algorithm that assigns timestamps to measurements collected using each mote's local clock. One or more motes have references to a global time source. All motes broadcast their time-related state and periodically record the broadcasts of their neighbors. If a few mote segments are able to map their local measurements to the global time frame, this information can then be used to assign global timestamps to the measurements collected by their neighbors and so on. This epidemic-like spread of global information makes Phoenix robust to random mote reboots and basestation failures. We found that in practice there are more than enough possible ways to obtain good fits for the vast majority of data segments.

Results obtained from simulated datasets showed that Phoenix is able to timestamp more than 99% of measurements with an accuracy up to 6 ppm in the presence of frequent random mote reboots. It is able to maintain this performance even when there is no global clock information available for months. The duty-cycle and space overheads were found to be as low as 0.2% and 4% respectively. We validated these results using a 21 day-long real deployment and were able to reconstruct timestamps in the order of seconds.

In the future, we will investigate using other metrics for determining edge weights and their impact on the quality of the time reconstruction. Moreover, we will explore adaptive techniques for determining the anchor collection frequency. Finally, we will derive theoretical guarantees on the accuracy of Phoenix, which can be used to allow for fine-grained tradeoffs between reconstruction quality and overhead.

Chapter 6

Exploiting Spatiotemporal Correlations

The theme of this chapter is mining the information within the sensor data to improve processes in the data pipeline. We touched upon the amount of data generated by the LUYF system in the Introduction (Figure 1.1). The overall goal is to understand some features in the data and fold in this information to improve system components. Using correlations between sensor measurements to detect faulty readings is one such application. Understanding the sources of heterogeneity in the data to optimize the data collection subsystem is another application.

The rest of the chapter reads as follows. Section 6.1 provides an introduction to some prominent features in the environmental datasets gathered by the LUYF project. We take a closer look at these features in Section 6.2 and understand the subtleties associated with soil temperature data obtained from the Cub Hill deployment. This case study provides a platform for understanding how we robustly deal with data outliers (outlined in Section 6.3). Section 6.4 explores the tradeoff between reducing fidelity and increasing network lifetime, and finally, In Section 6.5, I present results of applying this adaptive data collection method to the LUYF system.

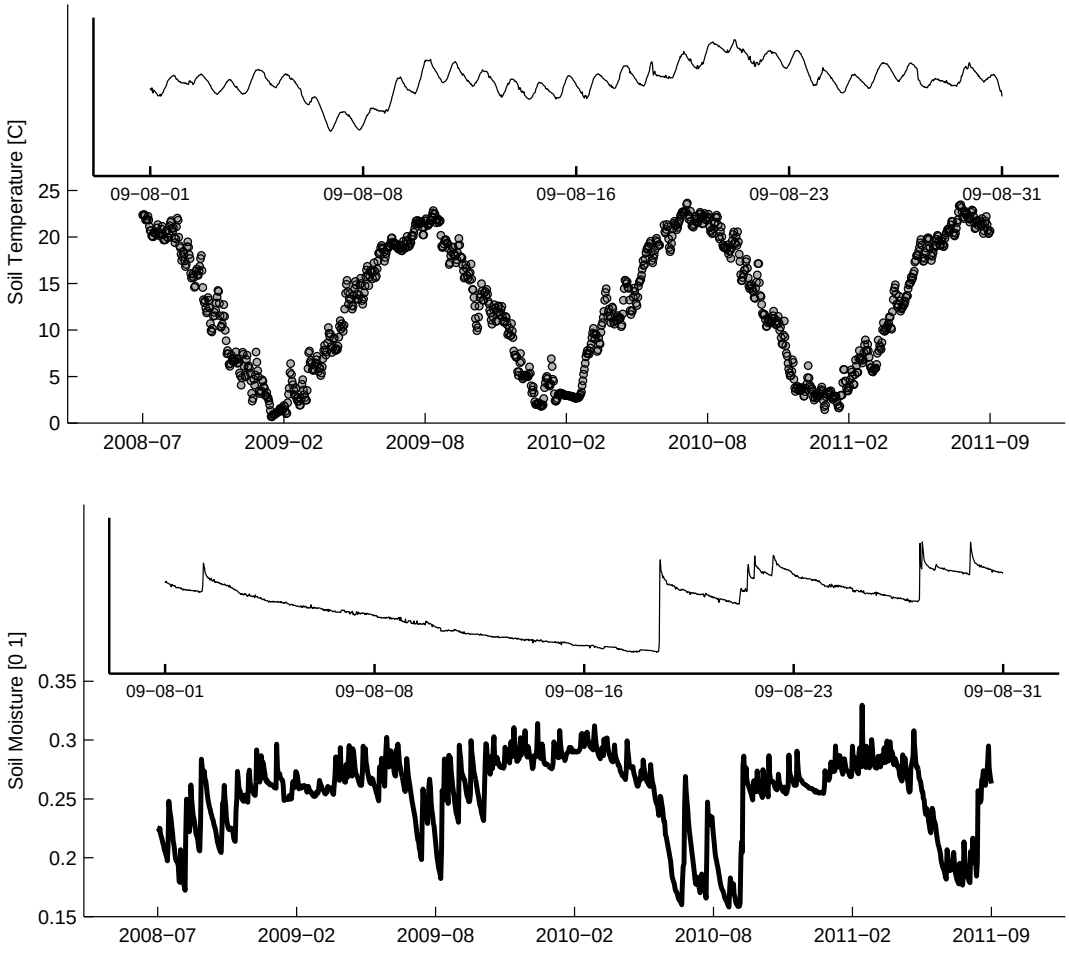


Figure 6.1: The average soil temperature and soil moisture at the Cub Hill deployment for the period between July 2008 to September 2011. The figure also shows the detail for the month of August in 2009.

6.1 Features of LUYF Data

In order to understand various applications of spatiotemporal correlations, I wanted to begin by giving a flavor of some properties and features that are observed in our datasets. Environmental data can be sliced and diced along various dimensions. To name a few, data can be characterized along time (temporal correlation), space (spatial correlation), between modalities (cross correlation) etc. We begin by looking at some high-level features in the data and

Figure 6.2: The sampling locations for the Cub Hill deployment. TelosB nodes are placed at each location and data related to soil conditions (temperature, humidity) are collected every 10 minutes from each location. Note that this is an aerial view of the deployment site captured during the winter (leaf cover is absent).

then drill down to some specifics.

Long-term Data Features : Soil Temperature (ST) and soil moisture (SM) data collected over three years from Cub Hill is shown in Figure 6.1. Both modalities are strongly correlated in time. Stated differently, measurements obtained at time t are strongly related to measurements recorded at $t-1, t-2, ...$ etc. ST data demonstrates strong annual and diurnal components whereas SM demonstrates sharp spikes that are responses to rain events. One also notes that ST and SM appear to be weakly correlated in a negative way. The presence of the annual and diurnal patterns in the ST data are due to the changes in the solar cycle.

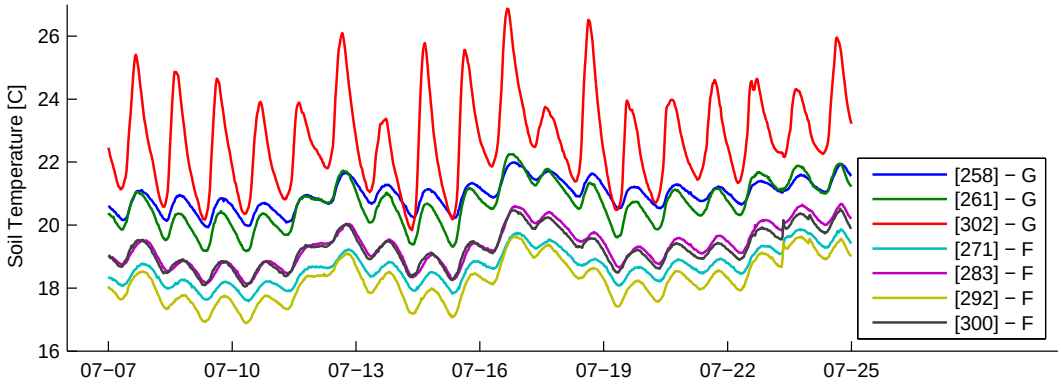


Figure 6.3: Compare and contrast soil temperature data collected from sensors located in the forest (F) and in grass (G). Note: This data is from 2009. The year label has been omitted for brevity.

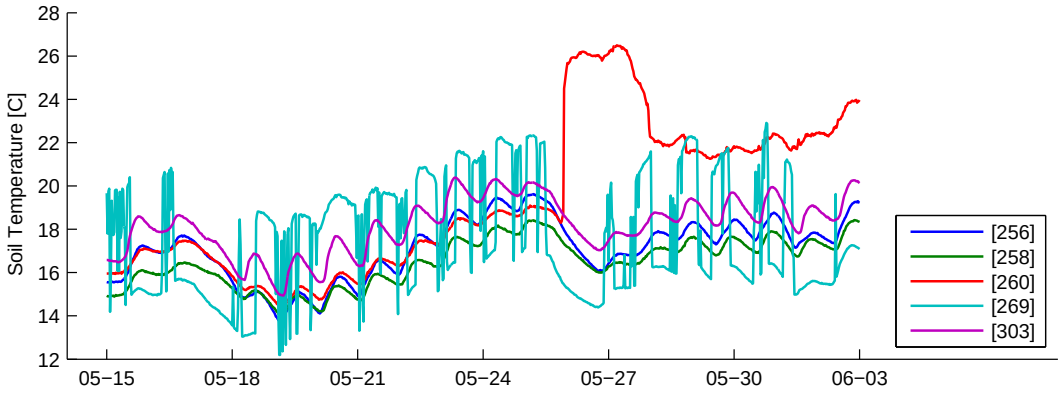


Figure 6.4: An illustration of faults in the soil temperature data. Data from location 269 is faulty and sensor at location 260 measures faulty readings after a big rain event on 05-26.

Variability Between Locations : Environmental WSNs are deployed to capture the underlying heterogeneity over time and space. The Cub Hill deployment, for example, is deployed with a goal to understand the impact of land usage and cover on soil properties. Figure 6.2 shows the locations where soil probes are placed. Majority of the sensors are placed in the forest. A small number of sensors are placed on grass patches and on the edge of the forest (the forest-grass boundary). As one might expect, the data obtained from these sensors

is strongly correlated. That said, a number of subtle differences can be observed that are of interest to the scientists.

Two week's worth of soil temperature data collected from a few forest and grass locations is shown in Figure 6.3. The most striking feature of this data is that all the sensors demonstrate the presence of the dominant diurnal cycle. One notes that the means of the sensor values in the grass are higher than those located in the forest. One of the main reasons for this is the lack of direct sunlight exposure in the summer months due to the presence of a thick leaf cover on the trees. The sensors located in the forest exhibit a characteristic lag with respect to ones located in the grass. The precise reason for this is beyond the scope of this discussion - it is related to the inertia and the buffering effect of the forest soil.

Data Quality : There are two main challenges in dealing with the data collected using environmental WSNs. The presence of missing values is the first one and the presence of noise/faults in the sensor measurements is the second major challenge.

Missing values are caused by hardware and/or software failures. When nodes run out of power or fail due to environmental factors, they stop logging data. These failures result in missing values until the faulty nodes are replaced. The inability to assign timestamps to the measurements is also a cause of missing values in the final processed dataset. An example of faults in the data is shown in Figure 6.4. The soil temperature sensor at location 269 is faulty for the most part (barring small periods of time when it appears to be working correctly). Data gathered from the sensor located at location 260 is working correctly until May 26^{th}. A big rain event damages the sensor and causes it to fail. The sensor records a temperature of $27^{\circ}C$ immediately after the rain event and consistently records values that are significantly higher than expected. Furthermore, the diurnal pattern is not exhibited by the sensor at location 260 after May 26^{th}. Such types of failures are frequently observed in our datasets.

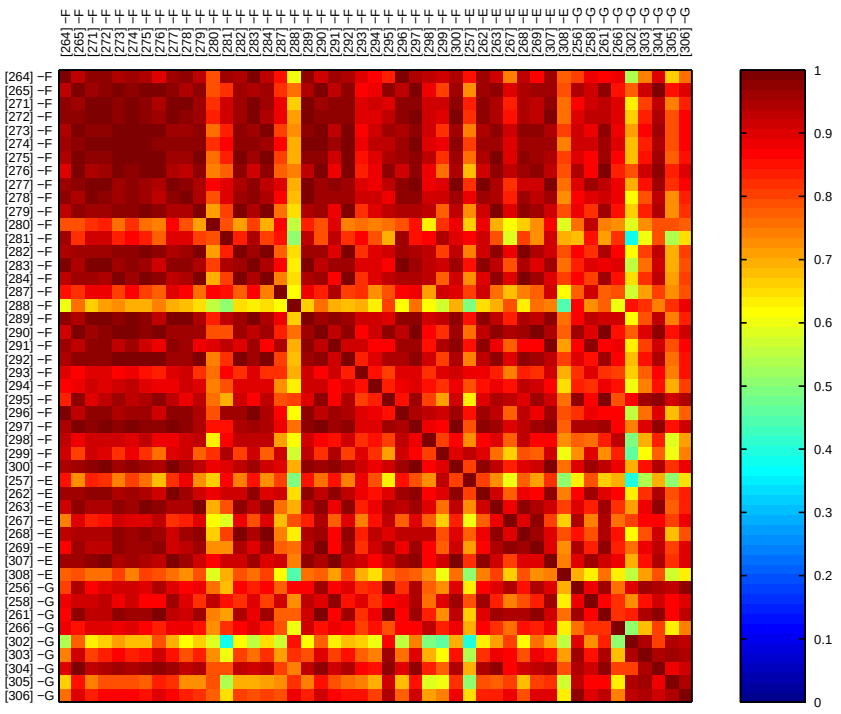


Figure 6.5: The correlation matrix for the soil temperature data from Cub Hill for a one week (2009-5-28 and 2009-06-05) period in the summer. The labels on the axis represent the locations. Forest locations are represented as "F", grass as "G" and locations on the forest-grass boundary as "E".

6.1.1 Applications of Spatiotemporal Correlations

The strong correlations we have been talking about so far are demonstrated in Figure 6.5. Data from one week is used to demonstrate the strong associations. These correlations can leveraged to improve our system in the following ways:

- **Data Preprocessing**: Identify measurements that deviate from expected behavior.
- **Data Collection**: Engineer strategies to minimize the amount of data retrieved (communicated) by the network without considerable loss in fidelity (accuracy).

In the sections to come, we will take a closer look at each of these applications.

6.2 Cub Hill Data Case Study

In the previous section, I presented some high-level features observed in environmental datasets. We also touched upon the spectrum of applications that can benefit from exploiting these features and correlations. In this section, I want to go one level deeper and present data obtained from our Cub Hill dataset. To be specific, I want to achieve two things. First, to provide a higher resolution of the characteristics shown by typical LUYF datasets, and second, to describe the methodology used to pre-process (clean) our datasets.

The Cub hill deployment consists of a network of 50 locations as shown in Figure 6.2. At each sampling location, soil temperature and soil moisture probes are placed at depths of 10 cm and 20 cm. Data from these sensors is acquired every 10 minutes. Soil temperature data gathered from Cub Hill for over ten months is shown in Figure 6.6. I would like to draw your attention to the following aspects of this figure:

1. Prevalence of sensor faults and missing observations. The faults are represented by sudden discrete transitions in the color map and the prominent faults are annotated.

2. Strong correlations among sensing locations represented by the similarity in colors across various time snapshots.

The annotated faults shown in Figure 6.6 are caused due to low-quality packaging and insulation from rain water. Researchers from other groups have also reported the prevalence of faults in their soil monitoring networks [52, 72].

6.2.1 Data Preprocessing Challenges

Before we start to look at the preprocessing details, let us understand some of the challenges in cleaning up the data at a high level. The presence of missing observations means that the

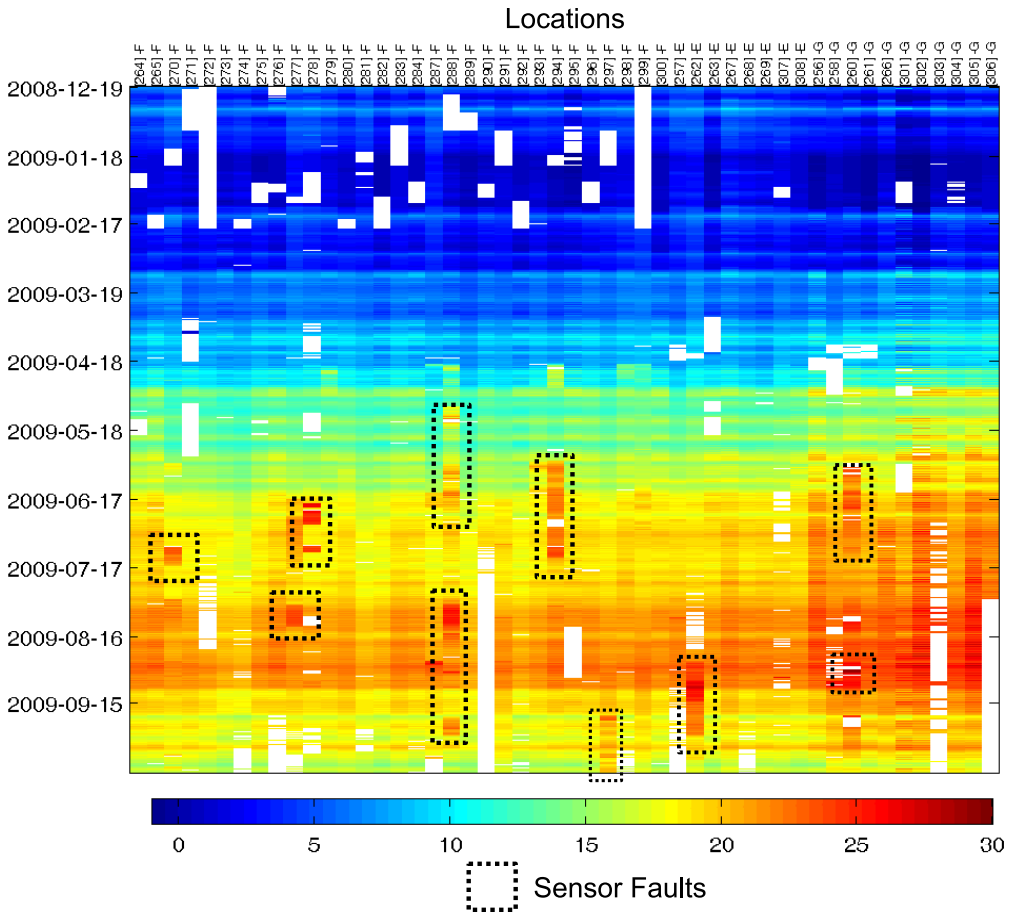


Figure 6.6: Soil temperature dataset collected from Cub Hill. Figure illustrates the prevalence of sensor faults and missing observations in the data. Time is represented by the Y-axis. Each strip along the Y-axis represents data collected from a given location (labels on top). The colored pixels represent the temperature at a given time for a given location. The locations and land usage types are labelled on top. F is forest, E is edge of the forest and G is grass

fault detection method should be able to deal with gaps gracefully. From Figure 6.6, it is clear that the sensor measurements are a function of time (data is non-stationary). The method and its parameters should be designed keeping this aspect in mind. Finally, I would like to highlight a very subtle point. Observe the faulty periods for some of the forest locations - one can see that these measurements are comparable in magnitude to the measurements collected in some of the grass locations. Thus, in order to identify whether a measurement is faulty or not, the method also needs to consider its site characteristics.

Previous work in Fault Detection

Sharma et al. provide a nomenclature for sensor faults and explore methods for detecting these faults [60]. They look at heuristic methods, least square regression and hidden Markov models. The Cub Hill dataset contains faults that are best identified using spatial correlations with site characteristics that are similar. In that regard, least square regression would be suitable but the other methods would not as they are based on temporal correlation. Ni et al. present a detailed study of the various types of faults observed in environmental data collected using sensor networks [47].

Ramanathan et al. present, Suelo [52], a system that involves humans to validate, calibrate, and detect sensor faults. The system is initialized by letting humans annotate certain faults. These annotations are used to learn the human's actions and responses. Suelo extracts features from the data collected from chemical sensors and uses an exponential weighted moving average based Gaussian estimator to detect faults. While these features are appropriate for their dataset, these features are not observed in the Cub hill soil temperature dataset.

Yao et al. study the problem of identifying anomalies in time series data by breaking up the signal into piecewise linear models. This broken up time series is compared against a reference time series using a distance metric. Periods that differ significantly from the reference time series are considered to be anomalous.

Designing a fault detection mechanism depends strongly on the data. In general, it is hard to provide one general recipe for all datasets. Our approach, presented in the next section, is closest in spirit to work done by Yao et al. It also relies on the notion of constructing a reference signal. We refer to this method as SMADS - a supervised median-based anomaly detection method for spatiotemporal data.

6.3 SMADS

SMADS is a novel approach we have developed based on the features observed in the Cub Hill dataset. The main steps in SMADS are outlined below:

1. **Cluster Identification**: Determine the number of clusters in the region being sensed. Assign cluster membership to each sensing locations.

2. **Label Data Per Cluster**: Identify data points that are considered to be non-faulty (clean). Tag adequate number of data points for each cluster.

3. **Training Phase**: Develop a robust model for each sensing location. Use the tagged data to obtain a threshold for declaring a measurement faulty.

4. **Prediction and Identification**: Use the model to obtain a prediction and compare it with the observed measurement. Use the threshold to decide if the measurement is faulty.

In Section 6.3.4, we evaluate SMADS qualitatively. We also discuss its strengths, weaknesses and applicability in general.

6.3.1 Cluster Identification

It is clear from Figure 6.6 that data from similar sites (in terms of land usage) exhibit strong similarities. Note that in Figure 6.6 data from forest locations are grouped and they are placed on the left whereas data from grass locations are shown on the right .

This clustering effect is found to be based primarily on the location. We used domain knowledge to assign cluster membership to the sensing locations. Each location was assigned from one of three groups - forest (F), grass (G) or edge (E). We validated this assignment by

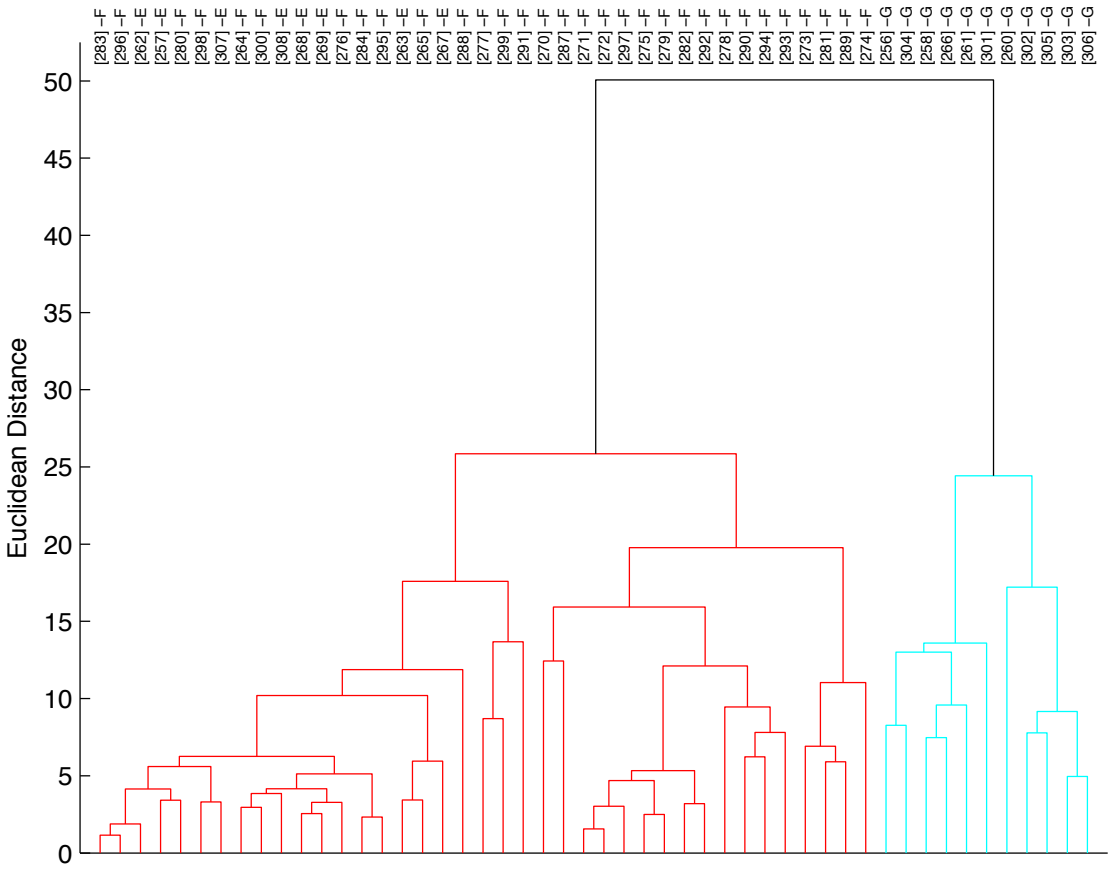


Figure 6.7: The dendrogram obtained using the Cub hill dataset. This plot was generated in MatLab using the weighted distance option and the Euclidean distance metric. Cub hill data obtained between March 2009 and May 2009 is used.

using half hourly data between 2009-3-15 and 2009-5-15 and running MatLab's hierarchical clustering (dendrogram) routine. This is a bottom up approach to clustering - each location starts out as its own cluster. Subsequently, pairs of clusters are merged based on the similarity of their data. The dendrogram (cluster merge tree) using the weighted average Euclidean distance metric is shown in Figure 6.7.

6.3.2 Label Data Per Cluster

Being a supervised method, SMADS requires us to tag a small amount of the data for training purposes. Tagging data requires a high amount of effort because an expert has to go through the dataset and classify data points as faulty or non-faulty. SMADS tries to minimize the amount of effort and time involved in the tagging process and this methodology is described below.

For each cluster we present the data as a visualization to the expert. The expert then has to annotate only the time periods (starts and ends) corresponding to data that he considers as reasonable and not faulty. For instance, going back to the Cub hill dataset, the entire period (from December 2008 to October 2009) for locations $272, 280$ from the forest cluster are found to be good. These locations are tagged as non-faulty. Similarly, data from locations $256, 266$ from the grass cluster are tagged as non-faulty. Note that these tags are not the same as the annotations shown in Figure 6.6. The annotations shown in the figure are just to illustrate the prevalence of sensor faults. These tags of non-faulty data are used to bootstrap the fault detection process and classify measurements from other locations. The intuition is that we build a model for data that is considered to be non-faulty using these tags. Data points that are "outlying" with respect to this model are classified as faulty.

6.3.3 Training Phase

The training phase consists of the following steps.

1. Compute a cluster specific median signal.

2. Develop a robust location specific model using data from each location and the median signal.

3. Use the model and tagged data to obtain a cluster specific error distribution and determine a suitable threshold.

An untagged data point is classified using the model and threshold. I will describe the details of the model building process next.

Representative Cluster Specific Signal

The cluster specific median signal for a given time t, and cluster c, is given by the following equation

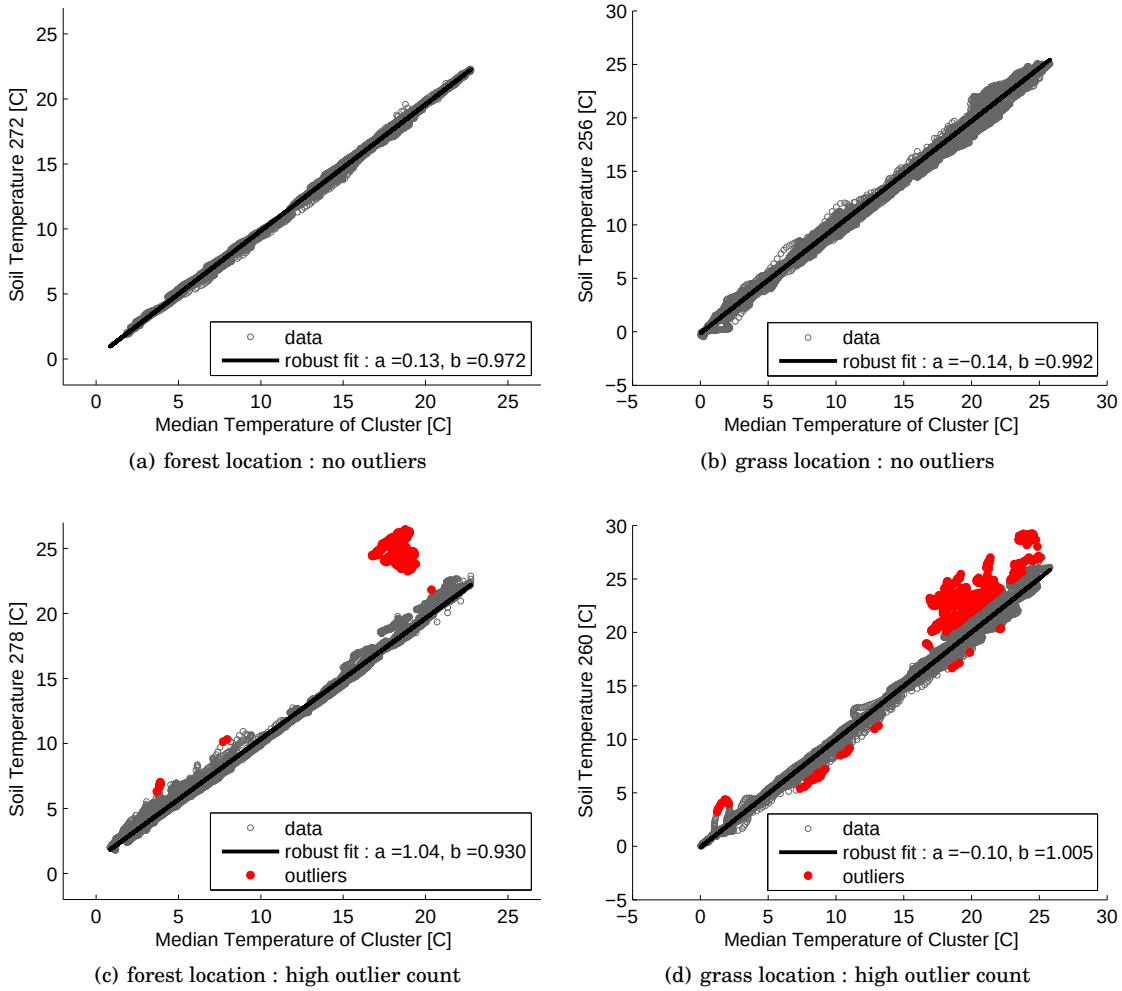
$$\forall t \in \mathbf{T}, \mathbf{c} \in \mathbf{C} : m_{t,\mathbf{c}} = \mathbf{MEDIAN}_{l \in \mathbf{c}} \{o_{t,l}\} \tag{6.1}$$

where $\mathbf{T}$ represents the set of all time instances for which data is available, $\mathbf{C}$ represents the set of all clusters, l represents all locations within cluster $\mathbf{c}$ and $o_{t,l}$ represents the measurement obtained at time t from location l. The timeseries given by $m_{t,\mathbf{c}}$ represents the median cluster-specific signal. Note that we could have used mean to represent this information but we choose median because of its robustness to outliers.

Location Specific Model

We observe that measurements from each location are strongly correlated with its corresponding cluster median value. Figures 6.8(a) and 6.8(b) demonstrate this correlation for a forest location and a grass location respectively. One notes that this relationship is approximated well by a linear fit. Mathematically, this can be expressed as follows

$$\widehat{o_{t,l}} = a_l * m_{t,\mathbf{c}} + b_l \tag{6.2}$$



(a) forest location : no outliers

(b) grass location : no outliers

(c) forest location : high outlier count

(d) grass location : high outlier count

Figure 6.8: Figure shows the correlation between the data at different locations with the cluster median. Figures 6.8(a) and 6.8(b) represent locations in forest and grass respectively that have no outliers. Figures 6.8(c) and 6.8(d) show locations in forest and grass respectively with a high count of outlying measurements.

where $\widehat{o_{t,l}}$ represents an estimate of $o_{t,l}$ and $[a_l \ b_l]$ represents the location-specific parameters that need to be estimated.

Estimating parameters of the linear fit using the classic least-square [20] procedure will result in poor estimates due to the presence of outliers. In order to obtain robust estimates of the parameters, we follow the iteratively reweighted least-square methodology outlined by

Holland et al. [30]. The bisquare weighting scheme, described in [24] is used as the objective function in the minimization formulation. It is worth pointing out that the classic least-square procedure estimates parameters by minimizing the sum of squares of the residuals. In the presence of outliers, this sum will be dominated by these extreme values, resulting in a poor fit. The application of this approach to two forest and two grass locations is shown in Figure 6.8. I would particularly like to draw attention to Figures 6.8(c) and 6.8(d). In both examples, the procedure is able to achieve a good fit even though the data is strongly contaminated by outlying data points. These outliers were validated by visually inspecting the timeseries plots.

Estimation of Threshold Parameters

A threshold is used to determine whether a data point is to be considered faulty. This parameter is estimated per cluster. The overall idea is to use the tagged data to obtain a distribution for the residuals (deviation from the model predictions). This distribution is then used to assess how outlying a new observation is. The residual, $r_{t,l}$, for each data point is given by $\widehat{o_{t,l}} - o_{t,l}$; where $\widehat{o_{t,l}}$ is given using Equation 6.2 , and $o_{t,l}$ is the value of the measurement at time t and location l.

Recall that we have non-faulty tagged data available to us. The location specific models are applied to these tagged data points and the residuals are recorded. Let $R_{\mathbf{c}}$ denote the set of residuals for the tagged data belonging to a given cluster c. We look at the distribution of these residuals and find that $R_{\mathbf{c}}$ follows a normal distribution. The location (mean, μ_c) and scale (standard deviation, $\sigma_{\mathbf{c}}$) of this distribution are estimated robustly [39]. The thought behind estimating $\mu_{\mathbf{c}}$ and $\sigma_{\mathbf{c}}$ is to transform the residual random variable $R_{\mathbf{c}} \sim N(\mu_{\mathbf{c}}, \sigma_{\mathbf{c}})$, into a variable that follows a standard normal distribution ($\sim N(0, 1)$).

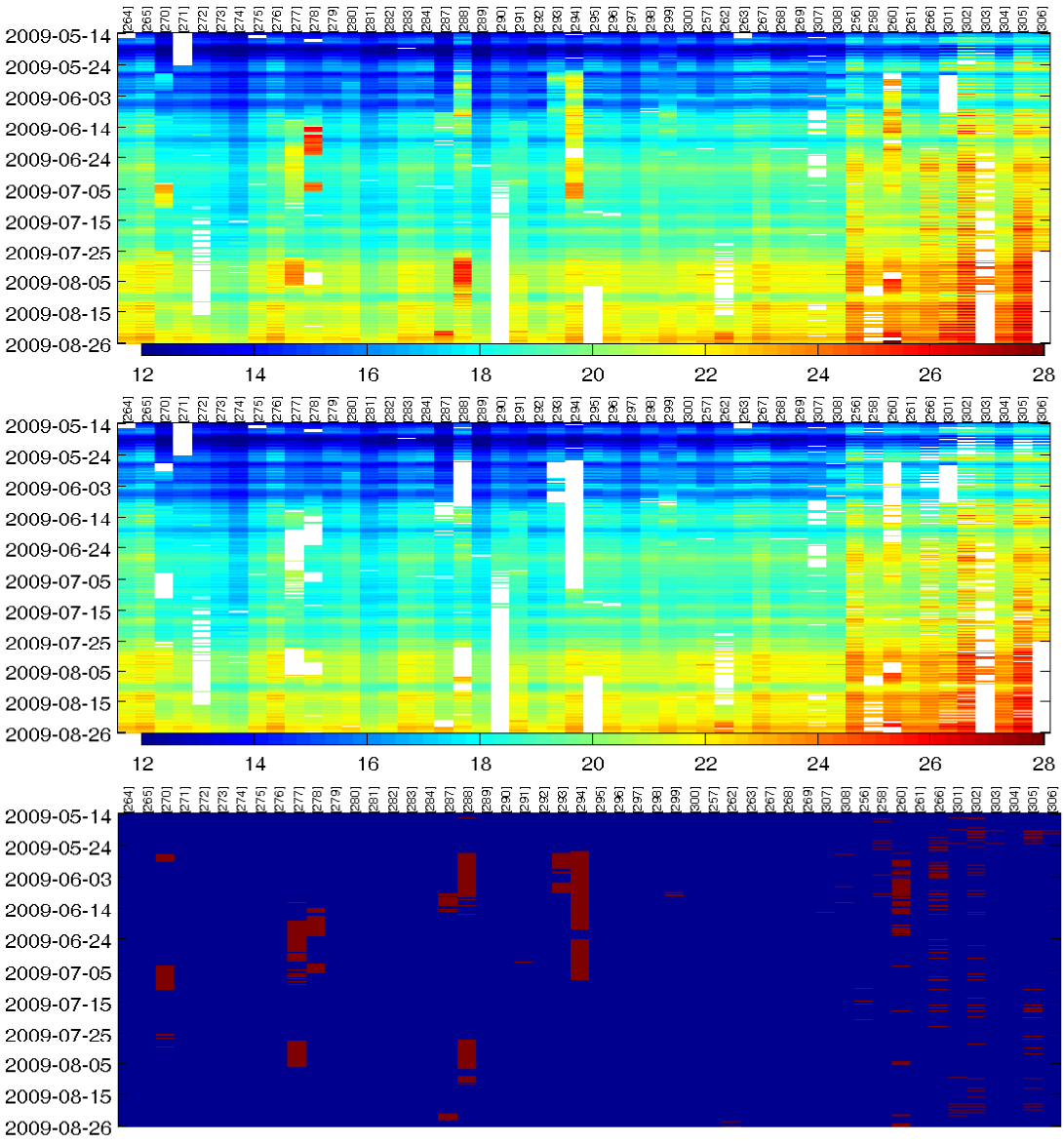


Figure 6.9: The top panel shows the original dataset from 2009-5-14 to 2009-8-26. The middle panel shows the dataset after applying SMADS to it. The red portions in bottom panel shows the locations corresponding to the top panel that were detected as faults by SMADS.

$$Z_{\mathbf{c}} = \frac{R_{\mathbf{c}} - \mu_{\mathbf{c}}}{\sigma_{\mathbf{c}}} \sim N(0,1) \tag{6.3}$$

The values that result from applying this transformation are often referred to as Z-scores

in the literature.

Prediction and Identification

This subsection describes the process of using the model and thresholds to determine whether a measurement is outlying. For each observation ($o_{t,l}$), the residual, $r_{t,l}$, is computed and then converted to a z-score using Equation 6.3. We tag all Z-scores outside of $[-4\ 4]$ as outliers. This interval is highly subjective - it depends on the application and should be tuned for each individual dataset. Note that, theoretically, more than 99% of the measurements of a $N(0,1)$ random variable lie between -3 and 3.

6.3.4 Evaluation and Discussion

Cub hill data from 2009-5-14 to 2009-8-26 is used to evaluate SMADS. This period is chosen for two reasons: (a) Largest variability for soil temperature is observed in the summer months and this period contains a large number of faults; (b) This period is also used to evaluate the adaptive data collection scheme discussed in Section 6.4.

The results of applying SMADS is shown in Figure 6.9. The top panel shows the original data as a heat map. Measurements detected as faults are removed from the original dataset and are shown as missing observations in the middle panel. The bottom panel shows the locations of those measurements that were detected as faults by SMADS. Also, note that outliers shown in Figures 6.8(c) and 6.8(d) are ones detected by SMADS.

The entire period shown in Figure 6.9 consisted of 250050 data points. SMADS detected 9566 of these as faults. It is difficult to evaluate SMADS quantitatively without having access to tagged data. Although, this is a subject that can be explored in great detail, our main objective here was to sanitize our datasets so as to minimize the impact of faulty measure-

ments.

SMADS leverages the spatial correlation among locations and is well suited for environmental datasets that contain multiple sampling locations. SMADS computes the median over all the available data within a cluster. The presence of missing values will tend to bias the median computation, but as such, SMADS is able to deal with the presence of missing values gracefully. A strong criticism of SMADS is that it does not make use of temporal correlation. Therefore, it is not able to effectively deal with time periods that exhibit a transient dip in the signal to noise ratio. Sharma et al refer to such measurements as noise faults [61]. [1] SMADS can be augmented with another method that detects faults using features in the temporal domain.

6.4 Adaptive Data Collection

Energy scarcity is a fundamental problem in long-term deployments. Postmortem timestamping methodologies outlined in Chapters 4 and 5 are examples of systems that trade off accuracy for reduced communication, thereby resulting in power savings. Radio, the largest consumer of power, is most heavily utilized in transferring the outstanding data from the motes to the basestation. Earlier in this chapter, we looked at the prevalence of strong correlations in the data. Using these correlations to reduce the amount of data that needs to be transmitted provides opportunities to achieve power savings.

In this section, we will explore how we can reduce the amount of data retrieved by the system to increase the network lifetime. Using compression schemes is one way to reduce the amount of data that needs to be transferred [11]. In this thesis, a data-driven approach is explored to reduce the amount of data that needs to be retrieved, and this approach would work in conjunction with a compression scheme.

[1] Such type of faults are not commonly observed in the cub hill dataset.

At a high level, correlations in the data can be exploited to identify measurements that capture and account for majority of the information (or variation). By selectively transmitting these measurements and suppressing the remaining ones, the system can achieve a balance between reducing communication costs at the expense of some tolerable loss in accuracy. Intuitively, a higher number of transmitted measurements would result in increased fidelity (lower loss). However, identifying which measurements need to be transmitted under fluctuating environmental conditions presents a non-trivial challenge. In this section, we will explore various methodologies to address this question and simultaneously study the amount of data that needs to be transmitted to achieve a certain level of accuracy.

Before we dive into the details of collecting data adaptively, I'd like to begin by discussing the motivation for studying this topic from the perspective of the LUYF system.

6.4.1 Introduction and Motivation

The LUYF sensor networks use a low-power data retrieval mechanism known as Koala [44]. An overview of the end-to-end system was described in Section 3.3. Recall that the data is locally cached at the motes until the basestation retrieves all the outstanding data and note that the typical sampling rate for the LUYF networks is 10 minutes.

The basestation does a bulk download to retrieve the outstanding data every 12 hours. Motes turn off their radio in between these downloads in order to conserve power. It is not surprising that the amount of time the motes need to keep their radios on is directly proportional to the amount of outstanding data in the network. Stated another way, the duration of time for which a mote need to keep their radio on using Koala is directly related to the number of motes from which the outstanding data is being retrieved (more motes results in more data being downloaded). Figure 6.10 demonstrates the linear relationship using

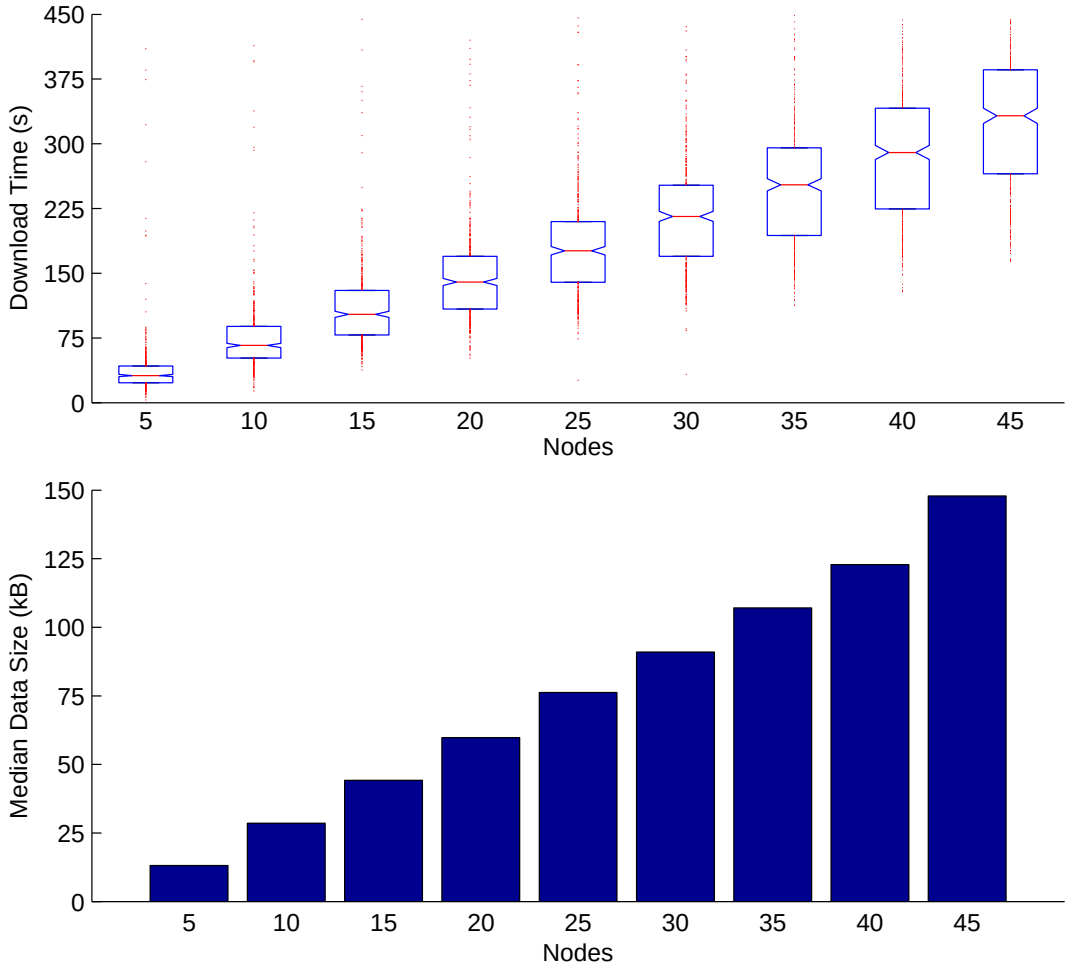


Figure 6.10: Radio usage during each download round. The top panel shows the total download time as a function of the number of nodes during each download round. The bottom panel shows the median data downloaded in each download round as a function of the number of nodes.

actual download times for the Cub hill deployment. This simple observation motivates us to explore the trade off between reducing the amount of data being retrieved by the basestation and studying its impact on information loss.

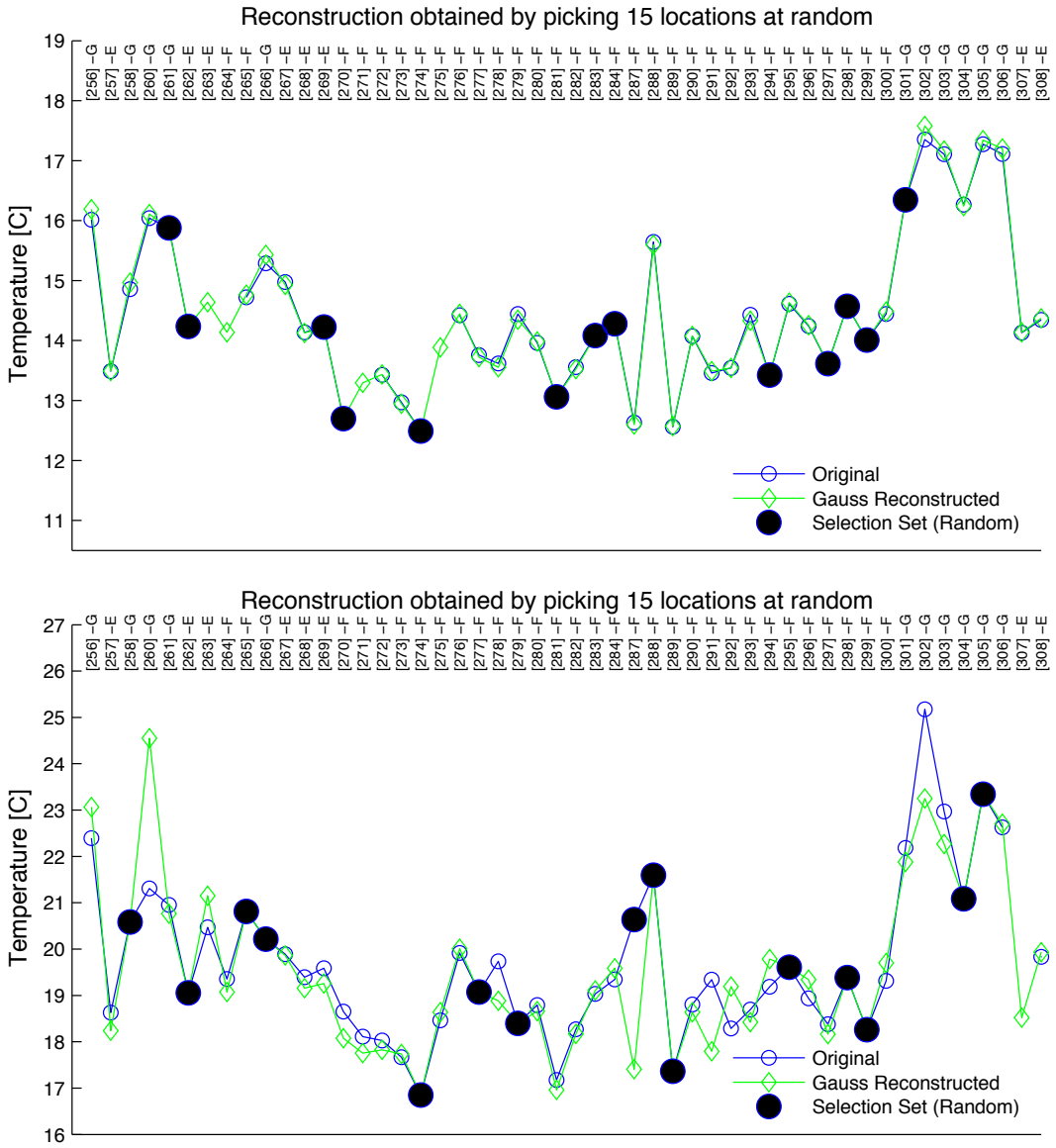


Figure 6.11: The top panel shows the reconstruction of data for a snapshot of the Cub Hill data using 15 randomly selected location. The bottom panel shows the same for another snapshot - the prediction errors for this snapshot are comparatively higher.

Scientist's perspective:

In many deployments, scientists analyse data months after the deployment has terminated.

That said, they are often interested in getting a high-level understanding of how the environ-

ment is varying in a real-time fashion. An effective selective data download system should bridge this tension between the high cost of downloading all data versus reducing costs by selectively downloading data and providing an effective approximation of the data.

In applications where real-time access is crucial, selective data downloading can be used to design a try-before-buy system. The thought here is that not all periods are equally important. Some periods may be of higher interest to scientists (a rare event) in comparison to others. The system could present the scientists with a good approximation of the data initially. If data for this time period is of high interest, measurements from all locations could be retrieved. Such a system would work in applications where there is a big interest in a small number of infrequently occurring events or time periods. For example, an agricultural application designed to monitor the soil moisture conditions might be deployed to obtain the spatial distribution of moisture after big rain events or thunderstorms. In between these events, retrieving data from all locations may be too costly and unnecessary.

Motivating example for LUYF

To motivate this area of research further, consider a snapshot of soil temperature data collected on 2009-5-17 from all active sensors at Cub hill shown in the top panel of Figure 6.11. The blue series with bubbles shows the original data collected by the sensing locations. The figure also shows 15 locations that are selected at random (shown as solid circles) and a prediction (green series with diamonds) at other locations using these 15 locations. It is clear that the data corresponding to this time snapshot is highly predictable and hence collecting data from all locations is overkill. On the other hand, consider the reconstruction obtained for another snapshot (2009-7-19) of data shown in Figure 6.11 (bottom panel) where the predictions are not very accurate. The predicted data at location 260 is $3^{\circ}C$ higher than the actual temperature and predictions at locations 287 and 302 are off by as much as $2^{\circ}C$. This

exercise serves as a motivating example for selectively collecting data from informative locations and adaptively varying the parameters of the data collection scheme to ensure fidelity is maintained.

The topic of adaptive data collection has been studied by a number of researchers. It is impossible to discuss the entire literature but I will discuss some of the major bodies of work in this area and compare and contrast how these systems differ from our system.

6.4.2 Literature Survey

The problem of model-driven data collection in sensor networks was introduced by Deshpande et al. [18]. Their system, referred to as BBQ, is motivated by work done in the approximate query processing database community. The BBQ system accepts a query q and an error tolerance δ. The system models the sensor data (including different sensor types) as a time-varying multivariate Gaussian. Based on δ, the system decides whether the query can be answered using the present model. If this cannot be achieved, its constructs a plan to collect additional samples such that acquisition costs are minimized. BBQ presents a clean way to predict unavailable measurements given the available measurements and their uncertainties. The focus of this system is to reduce acquisition costs to address user queries. This differs from our system goals as the LUYF scientists require all the data to be made available to them at the end of the deployment. Therefore, we cannot save on acquisition costs. In contrast, the goal of our work is to suppress transmission for measurements that can be easily predicted using other measurements in an adaptive fashion.

PRESTO introduces a two tier architecture for data storage and management [35]. The system contain a small fraction of proxy nodes that are rich in computation, communication and storage resources. These proxy nodes build a seasonal autoregressive integrated moving

average (SARIMA, [8]) model based on historical data and push these model-parameters to the sensing nodes. These sensing nodes report an actual sensed value to the proxies only when the measurement lies outside the confidence interval as given by the model. Since the proxy also has knowledge of the model, if a value is not reported by a node, the proxy stores the model prediction as an approximation to the observed value. This work is close in spirit and flavor to our goal. However, the most significant difference is that SARIMA does not capture spatial correlation. As a consequence, each node in PRESTO acts autonomously.

Lance [74] addresses how a system can maximize the overall value of the collected data subject to communication constraints. The design splits up the collected data in units known as application data units (ADU). By scaling the value of each ADU with its cost, an effective value is computed and this value is leveraged to download data using a greedy knapsack approach. This work focusses on system design and policies for networks with a high frequency of sampling. As such they assume that the value of each ADU is a well-defined entity for each application and hence do not provide a general statistical recipe for using spatiotemporal correlation to download data adaptively.

Krause et al. explore the topic of optimal placement of sensors by maximising the mutual information between selected locations and locations which are not chosen [33]. The general problem of optimal placement of sensors is a known NP-complete problem. This work explores various strategies to address this general problem and they derive a polynomial time approximation for the mutual information based approach that is within $(1 - 1/e)$ of the optimal configuration. The goal of our work is to find optimal sensor locations from a subset of already placed locations in an adaptive and online fashion. We make use of the results obtained by Krause at al. to identify these subset of informative sensors. We evaluate various strategies to study the trade off between reconstruction error and reduced communication in the context of the LUYF system.

6.4.3 Finding Informative Locations

Consider that a sensor network consisting of d discrete locations has been deployed. As observed in the Cub hill dataset, there is considerable spatial correlation among these locations. Figure 6.11 further demonstrates the ability to perform accurate prediction at unobserved locations using a subset of observed measurements. The basic idea behind finding informative locations is to determine a small subset k (out of d) of locations that provide good predictions at the other $(d-k)$ locations.

Krause et al. leverage mutual information to determine these informative locations [33]. Let $U : |U| = d$ represent a set of all locations and $S : |S| = k$ represent the set of selected locations. The underlying variable being sensed, $\mathbf{x}_U \in \mathbb{R}^N$, is assumed to belong to a multivariate gaussian distribution. The members of S can be obtained by maximising the mutual information given by $I(\mathbf{x}_S; \mathbf{x}_{U \setminus S})$. Following Krause et al., this can be mathematically represented as follows:

$$S^* = \underset{S \in U : |S| = k}{\operatorname{argmax}} \quad \mathbf{H}(\mathbf{x}_{U \setminus S}) - \mathbf{H}(\mathbf{x}_{U \setminus S} | \mathbf{x}_S) \tag{6.4}$$

where $\mathbf{H}(\mathbf{x}_U)$ represents entropy of the random variable $\mathbf{X}_U$ and $\setminus$ represents the set difference operator. The exact solution to Equation 6.4 is NP-Complete. However, [33] describes an approximate greedy solution by adding sensors that provide maximum increase in the mutual information. The details of this greedy, computationally efficient solver [3] are beyond the scope of this thesis, but the reader should refer to [33] for more details. In our application, we utilize these results and apply them in the context of the LUYF system.

6.4.4 Data Reconstruction Methodologies

In the previous section, we looked at a mutual information based approach to choose a subset of informative locations. Data gathered from these locations are used to predict (or reconstruct) values at locations for which measurements are unavailable. The prediction error provides the system feedback regarding the quality of the selected locations and their ability to capture spatial heterogeneity. For example, in an agricultural application, an irrigation administrator might want to set up an alert whenever the predicted soil moisture goes above a certain value. Here it will be important to predict values accurately. Once the predicate conditions are met, the system can then monitor the environment in more detail by downloading data from all locations that match the predicate (instead of a small subset).

Therefore, we require a methodology to reconstruct data at locations for which measurements are unavailable. In the LUYF project, this reconstruction is meant to serve two purposes

1. To evaluate the effectiveness of the spatial heterogeneity captured by the subset of selected locations.
2. To predict values at unavailable locations for data visualizations, spatial interpolation etc.

Two methods of reconstruction were explored. The first one is based on Principal Component Analysis (PCA, [20]). The second is a well known interpolation scheme for Gaussian processes [54].

PCA Reconstruction

PCA is a well known dimensionality reduction and multivariate analysis technique. I will begin by introducing the main concepts of PCA and proceed to explain how we can use this

method to reconstruct partial vectors.

This method is well-suited for multivariate data in which the individual variables exhibit high correlation. The goal of PCA is to find a small number of uncorrelated directions that capture majority of the variation. Essentially, these directions are translations and rotations of the original directions such that each variable is now represented as a linear combination of these principal directions.

Let $[x_1, x_2, ..., x_n]$ represent a set of points belonging to $\mathbb{R}^d$. Furthermore, let $\mathbf{x} \in \mathbb{R}^d$ represent the multivariate random variable. The goal of PCA is to find directions, $u \in \mathbb{R}^p : p < d$, given by the objective function

$$u^* = \underset{u:u^T u=I}{\operatorname{argmax}} \quad \textsc{Var}(u^T \mathbf{x}) \tag{6.5}$$

where VAR represents the variance operator. The first p principal components of the multivariate random variable $\mathbf{x}$ are given by the p leading eigenvectors (ordered by decreasing eigenvalues) of the covariance matrix ($\mathbb{E}[\mathbf{x}\mathbf{x}^T]$) [20]. For highly correlated data, a very small number of components are able to effectively capture the variation in the data. Stated another way, $p \ll d$.

Obtaining the principal components allows us to represent the original vectors in terms of the new directions.

$$\begin{aligned} x_1 &= \alpha_{11} u_1 + \alpha_{12} u_2 + ... \alpha_{1p} u_p \\ x_2 &= \alpha_{21} u_1 + \alpha_{22} u_2 + ... \alpha_{2p} u_p \\ &... \\ x_n &= \alpha_{n1} u_1 + \alpha_{n2} u_2 + ... \alpha_{np} u_p \end{aligned} \tag{6.6}$$

This can be concisely represented as

$$x_i = \sum_{j=1}^{p} u_j \alpha_{ij} \tag{6.7}$$

where $[\alpha_{i1}, ..., \alpha_{ip}]$ represent the coordinates of x_i in terms of the new directions. These coordinates are also referred to as eigencoefficients.

Now, let us consider another vector $x' \in \mathbb{R}^d$ that is drawn from x and lets suppose that x' contains some missing entries. We can define a mask vector, $w \in \mathbb{R}^d$, indexed by q, such that $w_q = 0$ if x'_q is missing, and $w_q = 1$ if x'_q is present. Note that $q : q \in [1, d]$ is used to represent indices into the entries of w and x'. The basis vectors $(u_1, u_2, ..., u_p)$ and the projections on these vectors $(\alpha'_1, ...\alpha'_p)$ are indexed by $j : j \in [1, p]$. Using these definitions, we can define an objective function as follows:

$$\sum_{q=1}^{d} w_q (x'_q - \sum_{j=1}^{p} u_{qj} \alpha'_j)^2 \tag{6.8}$$

This function can be minimized to solve for each α'_j and these $\alpha' s$ can be used to recover x' using Equation 6.7. The main intuition here is to find $\alpha' s$ that are able to best characterize the data that is available to us. The strong correlation in the data enables us use these $\alpha' s$ to estimate the observations that were masked out. Following Connolly et al. [2], the solution can be written as

$$\alpha'_j = \sum_{k=1}^{p} t^{-1}_{jk} z_k \tag{6.9}$$

where $t_{jk} = \sum_{q=1}^{d} u_{qj} w_q u_{qk}$ and $z_k = \sum_{q=1}^{d} u_{kq} w_q x'_q$. It is worth mentioning that t is proportional to the covariance between the eigencoefficients. If x' did not contain any gaps (no masked region), t would be unity. In practice, we employ a robust and streaming approach for finding the directions of variance. This is done because noisy measurements can severely

impact the quality of the reconstruction. Interested readers should refer to [9] for the mathematical aspects of the robust and streaming PCA.

Gaussian Method of Reconstruction

A multivariate Gaussian random variable is a simple and effective way to model data from all instrumented locations. This methodology has been extensively employed in the sensor network community [18, 33]. We will discuss how this model allows us to utilize observed data to predict data at all locations where no observations are available.

Let $\mathbf{x} \in \mathbb{R}^d$ be a multivariate Gaussian random variable $\sim N(\mu_x, \Sigma_{xx})$ representing data from all instrumented locations. Let S represent the set of size k - locations for which measurements are available, and let $\mathbf{s} \in \mathbb{R}^k$ be the multivariate random variable representing data from all observed locations. S partitions the original set into two parts - locations that belong to S and locations that do not. Let $\mathbf{r} \in \mathbb{R}^{d-k}$ represent the variable representing data from all locations that are not in S.

The original mean and covariance matrix of x can be written as

$$\mu_{\mathbf{x}} = \begin{bmatrix} \mu_{\mathbf{s}} \\ \mu_{\mathbf{r}} \end{bmatrix} \tag{6.10}$$

$$\Sigma_{xx} = \begin{bmatrix} \Sigma_{\mathbf{ss}} & \Sigma_{\mathbf{sr}} \\ \Sigma_{\mathbf{rs}} & \Sigma_{\mathbf{rr}} \end{bmatrix} \tag{6.11}$$

$$\tag{6.12}$$

We are interested in obtaining a posterior distribution after observing data from S. Note that conditioning a Gaussian variable on some attributes results in another Gaussian random variable. Therefore, r, is also Gaussian. Its mean and covariance matrix can be written as:

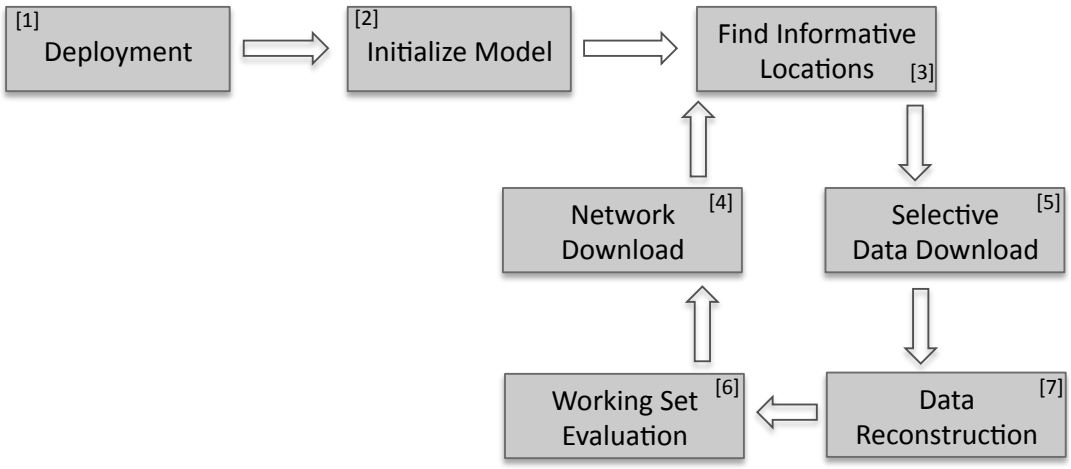


Figure 6.12: Various components in the adaptive data framework

$$\mu_{\mathbf{r}} = \mu_{\mathbf{r}} + \Sigma_{\mathbf{rs}}\Sigma_{\mathbf{ss}}^{-1}(s - \mu_{\mathbf{s}})$$
$$\Sigma_{\mathbf{rr}} = \Sigma_{\mathbf{rr}} - \Sigma_{\mathbf{rs}}\Sigma_{\mathbf{ss}}^{-1}\Sigma_{\mathbf{sr}} \tag{6.13}$$

where s represents the observation vector. Please note that the s (bold) is used to represent the variable and s (plain) is used to represent the observations. It is worth noting that, in addition to obtaining estimates at unobserved locations, this method also provides us the uncertainties in the prediction. For a more complete treatment on Gaussian Processes, please refer to [54].

6.4.5 Overall System Design

In this section, I'd like to present an overview of the relationships between the various adaptive data collection components in the context of the LUYF system. A schematic is shown in Figure 6.12. We will also take a look at how this scheme works in conjunction with the

existing networking infrastructure.

I'll begin by defining some terms that will be used in the remainder of the chapter. Working set is defined as the set of size k that comprises of the most informative locations. Training period is the duration during which the system downloads data from all instrumented locations. This data is used to (a) obtain the working set ; and (b) update the model(s) for predicting values at all locations that are not in the working set. The test period is the time for which the system selectively downloads data from the working set and suppresses data downloads from locations that are not in the working set. An update or snapshot (used interchangeably) refers to a single vector of observations from all instrumented locations. In contrast, the training period consists of many such vectors - all corresponding to different instances of time.

The model initialization component (Step 2 in Figure 6.12) is responsible for gathering data for a small amount of time to initialize the PCA basis and to determine the initial set of informative locations. For the Cub Hill dataset, collecting two days worth of data from all instrumented locations was found to produce reasonably good results (Section 6.5). Using data downloaded from all instrumented locations, the working set is computed using the methodology described in Section 6.4.3. Data from these S (using the nomenclature used earlier) locations are downloaded for a period of time given by the test period. From a LUYF system perspective, the Koala download protocol will selectively retrieve data from nodes at these S locations instead of downloading data from all the nodes. It is crucial to note that this reduction in data gathering is what leads to power savings. Recall that data is collected at each mote at a fixed frequency and is cached locally too making it available for download at a later point in time.

Reconstruction methodologies are used to predict values at locations for which data was not retrieved (Step 7 in Figure 6.12). At this point, the logical next step is to evaluate the

quality of the prediction. Naturally, periodic snapshots of data from all the locations are required to evaluate the efficacy of the model predictions. The journal records (Section 3.2) present us with an opportunity to receive these snapshots piggybacked with mote health information. Because journal records are small in size ($\sim$ 100 bytes) this is a light-weight mechanism for the system to achieve feedback. This work evaluates a few different strategies to take this update information and devise mechanisms to update the model, working set and test period. We will take a closer look at these strategies in Section 6.5. These data-driven strategies determine the amount and frequency with which data needs to be retrieved from the network.

This cycle of performing a full network download, establishing a working set, selectively downloading data and evaluation of the model prediction runs in a loop.

6.5 Adaptive Data Collection Evaluation

In this section, we evaluate the application of the mutual-information based adaptive data collection scheme on the Cub hill dataset (Section 6.2). Data from 2009-5-13 to 2009-8-26 is used for evaluation purposes. The period is chosen because the summer months exhibit larger variability compared to the winter months. We explore the following questions:

- Compare the effectiveness of the mutual information based selection with selecting locations at random.
- Study the impact of number of locations on prediction error.
- Study of impact of changing the test period to understand its effect on prediction error.
- Evaluate three download strategies - Periodic (P), Largest-errors-first (LEF), Event-driven (ED).

6.5.1 Evaluation Metrics

The system is evaluated using two metrics

- Absolute prediction error : The absolute difference between the actual value and the prediction. The median and 95^{th} percentile values are used to report aggregates.
- Data download fraction: The fraction of data downloaded. This is a measure of the power consumption as it capture the amount of work that needs to be done by the radio in transferring data.

The default configurations are as follows:

- Train data period : two days.
- Test data period : 14 days.
- Number of locations for selective data download : 10 (out of 50).
- Reconstruction Method : PCA. Whenever the Gaussian reconstruction method is used, it will be specified.

Compare with Random Selection

The main idea behind this experiment is to evaluate the effectiveness of finding informative locations using mutual information in comparison to the baseline approach of selecting locations at random. We report the prediction error for locations that were not selected.

The median prediction error as a function of the working set size is shown in Figure 6.13. Clearly, errors using mutual information are lower in comparison to the random selection of the working set. This gap is bigger when the size of the working set is small. As the working set increases this gap narrows. This is due to the fact that mutual information is

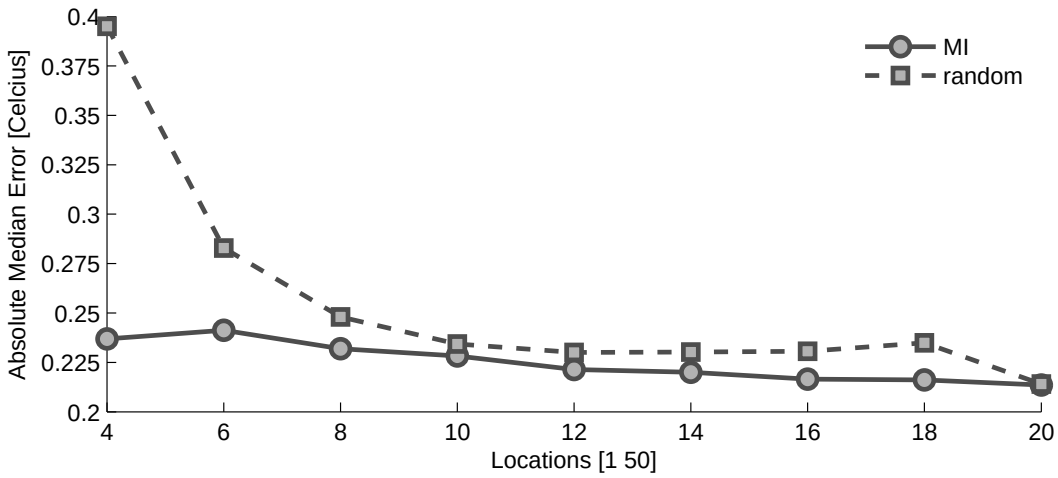


Figure 6.13: Reconstruction errors obtained when selecting locations randomly and using mutual information.

Table 6.1: Download fraction and reconstruction errors as a function of the working set.

Locations	Download Fraction	PCA-Median	PCA-95^{th}	Gauss-Median	Gauss-95^{th}
4	0.255	0.320	3.954	0.173	1.276
6	0.288	0.268	1.197	0.167	1.118
8	0.323	0.234	0.986	0.161	1.051
10	0.357	0.223	0.952	0.167	0.998
12	0.392	0.214	0.929	0.154	0.994
15	0.443	0.204	0.922	0.170	0.985
20	0.517	0.202	0.892	0.164	0.979
25	0.579	0.194	0.876	0.170	1.019

not monotonic. In fact, there is a point beyond which mutual information goes down as the number of selected locations increase. It was found that beyond 25 (out of 50) locations, the mutual information was no longer monotonic.

Impact of Working Set Size

The median and 95th percentile reconstruction errors using PCA and Gaussian reconstruction are shown in Table 6.1. The table also shows the fraction of data that was required to

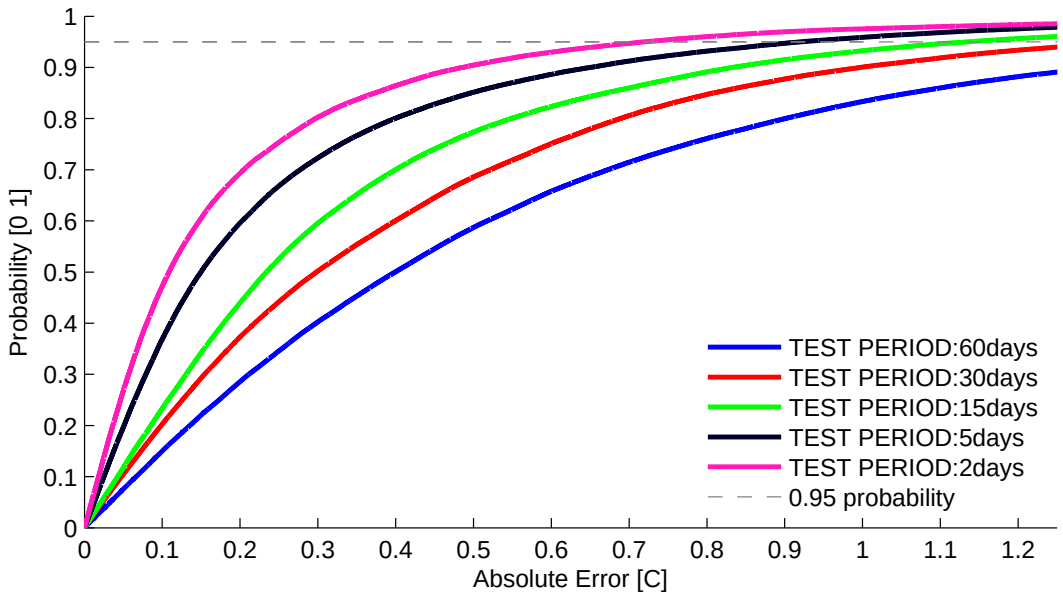


Figure 6.14: The impact of increasing the test period on reconstruction error.

be downloaded as a function of the working set size. For the PCA reconstruction method, one can see the diminishing returns property of adding more locations to the working set. The drop in median error is sharp when the number of locations are small and it flattens out as the size of the working set increases. We also see that beyond 12 locations the median error does not change significantly. This scheme of having a fixed train period and test period is referred to as the periodic methodology. This method serves as the baseline for comparing and evaluating other adaptive downloading strategies.

It is interesting to note that the median prediction error using the Gaussian method is lower than obtained using the PCA method. However, the roles are reversed when we compare the 95^{th} percentile error.

Table 6.2: Download fraction and reconstruction errors for the LLE data retrieval scheme.

δ (° C)	PCA Download Fraction	PCA-Median	PCA-95^{th}	Gauss Download Fraction	Gauss-Median	Gauss-95^{th}
1.25	0.440	0.181	0.679	0.414	0.083	0.463
1.00	0.490	0.161	0.588	0.429	0.077	0.424
0.75	0.547	0.151	0.553	0.465	0.070	0.367
0.50	0.681	0.122	0.476	0.524	0.063	0.324
0.25	0.878	0.100	0.431	0.724	0.044	0.253

Impact of Test Period

The test period is given by the frequency with which the working set is recomputed. Figure 6.14 shows the results of varying the test period. Note that the train period is fixed at two days for each test period shown in Figure 6.14.

As the test period increases, the prediction error increases. If we compare the median errors for Figure 6.14 and Figure 6.13, one observes that the prediction errors appear to be influenced more significantly to changes in the test period compared to the size of the working set.

Additionally, the timeseries of the prediction errors shows some distinct patterns. The working set and reconstruction model are not able to perform a good prediction after big rain events. When the test period is low, the system gets an opportunity to update the working set and reconstruction model more frequently - minimizing the impact of rain events. Figure 6.15 shows the daily 95^{th} percentile error and the soil moisture conditions for test period of 5 days. The sharp spikes in soil moisture correspond to rain events. Note that the spikes in soil moisture align with the high prediction error spikes. Another observation is that the period between 06/20 and 07/20 shows lower errors. This is also the period that corresponds to least rain.

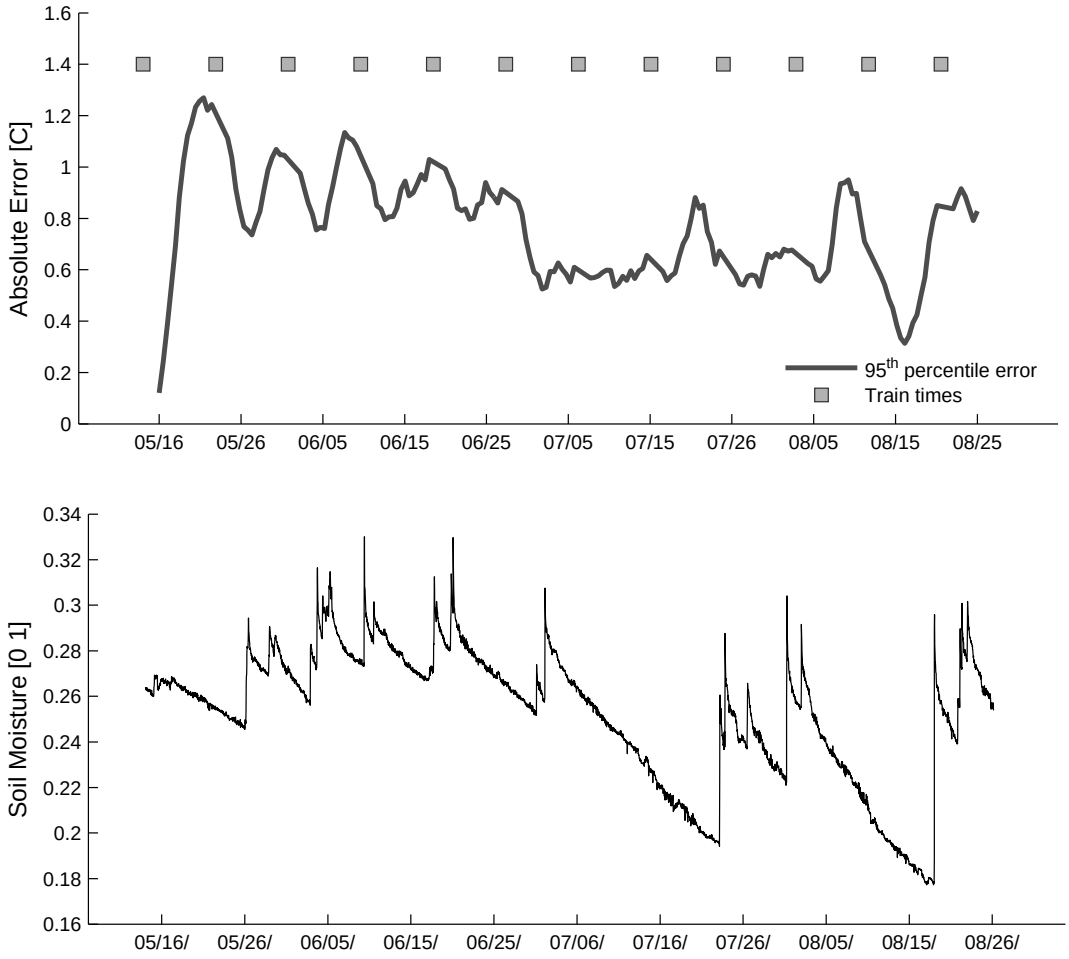


Figure 6.15: **Prediction error as a function of time and rain events for test period of 5 days**

6.5.2 Adaptive Data Collection Strategies

Periodic

The periodic methodology was discussed earlier when we looked at the impact of the working set size. The numbers presented in Table 6.1 serve as the baseline and our goal is to improve on them.

Largest Errors First (LEF) Retrieval

In this scheme, the journal records are used to receive data updates from all locations. The journal records are retrieved every 12 hours. The measurement updates are piggybacked along with these journal records. six measurements - one measurement randomly selected every 2 hours is part of the update packet. These complete vectors are used to update the PCA basis, the mean vector and covariance matrix. Thus these update records serve two purposes:

- Evaluate the effectiveness of the predictions.
- Update the PCA/Gaussian reconstruction model parameters.

The user or administrator specifies a tolerable prediction error δ and the amount of tolerance, ϵ (e.g. 95%) - fraction of errors below δ that are acceptable. Using these journal records, we evaluate the prediction error and compute the percentage of errors that are above δ for each location. If this fraction is higher than ϵ then that particular location is marked. During the next data download round, data from all marked locations are downloaded in addition to the locations in the working set. These marked location increase the effective size of the working set. Note that a location that was previously marked can also be unmarked during the test period if the fraction of errors above δ falls below ϵ.

There are two main intuitions for adopting this approach. First, we observe that there are a small set of locations (five or less) that consistently suffer from high prediction errors. The most likely explanation for this is that the variation in these locations may not be effectively captured by the working set. Downloading data from these locations separately would result in significantly lowering the errors. The second intuition follows from the fact that the prediction error increases as the test period increases. The model is unable to keep up with the changing environment. Frequent snapshots serve to alleviate this problem.

For the evaluation, the minimum number of locations to start off with at the beginning of each test period was set to five. The train period is set to two days and the test period is set to 14 days. The total number of locations part of each test data download fluctuated - depending on the number of marked nodes. ϵ was set at 95% and δ was varied. The results are tabulated in Table 6.2.

A somewhat surprising result is that the Gauss median and 95^{th} percentile errors are significantly lower compared to the reconstruction obtained using PCA. In Table 6.1 we found the Gauss-95^{th} percentile error to be higher than the PCA-95^{th} error. The intuition for this result is as follows. We observed that the Gauss method suffered from high prediction errors at a few locations but had low prediction errors in expectation. Using the LEF scheme, the locations with these high errors are downloaded so their impact is offset, resulting in significantly lower errors.

Comparing these results with the results shown in Table 6.1, we note that this scheme produces significantly lower errors for the same amount of downloaded data. For example, downloading 44.3% of the data using periodic downloads resulted in a median error of 0.17 and a 95^{th} percentile error of 0.98 whereas in the LEF downloading scheme, for a lower download fraction (42.9%), the median error was found to be 0.07 and the 95^{th} percentile error was found to be 0.424, which is significantly lower.

Event-driven Data Retrieval

The quality of the model predictions and effectiveness of the working set are significantly impacted by the arrival of rain events as shown in Figure 6.15. The idea of the event-driven data retrieval is as follows. An event detection algorithm is used to detect the onset of events (in this case, rain events). If the system does not detect an event until the next scheduled network download, it defaults to the LEF retrieval scheme. If an event is detected, the sys-

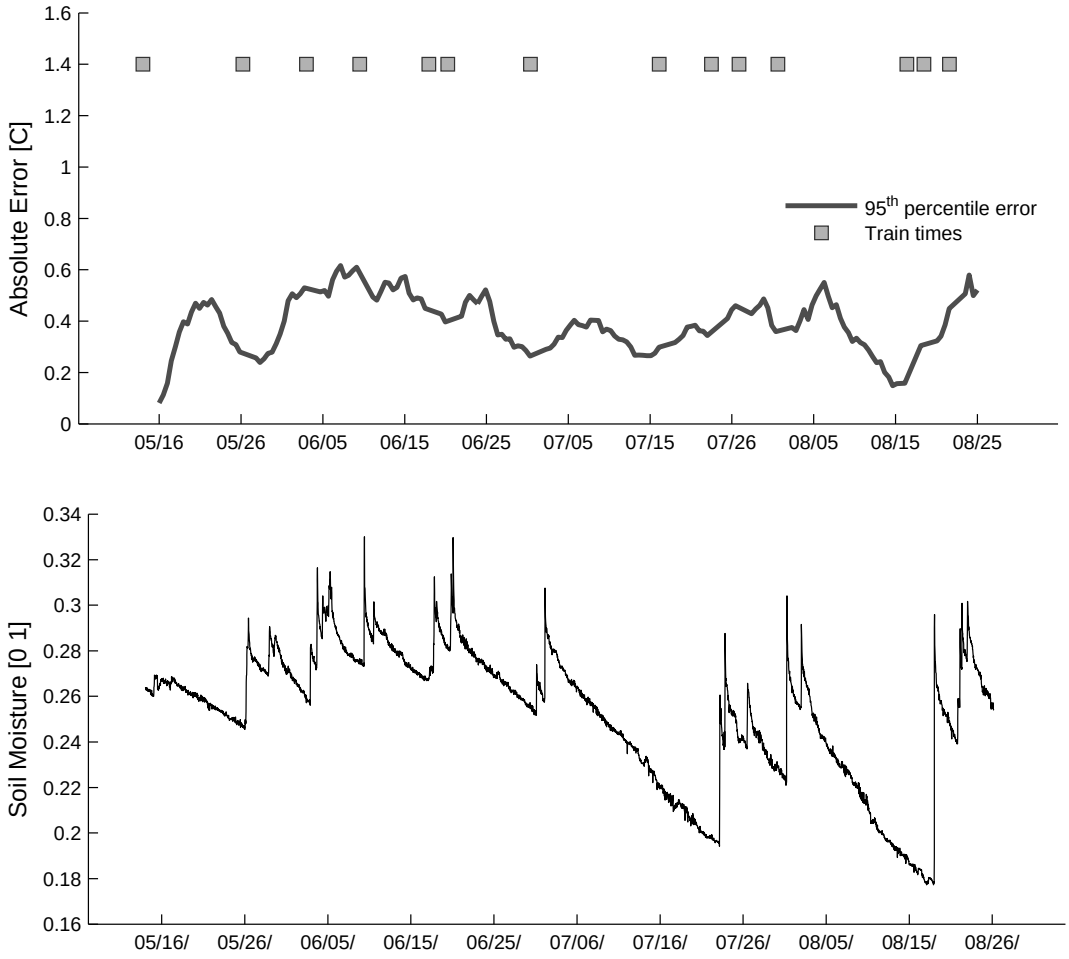


Figure 6.16: Daily 95^{th} percentile errors for event driven data downloads. Compare these errors with 6.15 and notice the low errors after rain events.

tem performs a complete network download for a 12 hour period following the event and recomputes the working set. The main intuition for doing this is that the rain events serve as change points and the model needs to be updated to reduce the prediction error. The differential flow of water has an impact in the way temperature varies at different sampling locations. This changes the way in which locations are correlated and these association needs to be reevaluated.

The daily 95^{th} percentile errors using the event driven retrieval methodology is shown in Figure 6.16. One notes that these errors are significantly lower compared to Figure 6.15 and the errors are not strongly influenced by rain events.

A very basic rain detection scheme is used for this dataset. The idea is that the system looks at successive soil moisture measurements and declares a rain event whenever the difference is above 0.2 units. The soil moisture signal is smoothed using a median filter to minimize the effect of outliers and false positives. Using the Gaussian method of reconstruction, the median absolute error was $0.06°C$, the 95^{th} percentile absolute error was $0.3°C$ and the fraction of data that needed to be downloaded was 56%. The improvement compared to the adaptive LEF scheme is only marginal. However, I argue that this is a more principled away of performing downloads because of the value of the data being downloaded. Capturing data during these events is what many environmental monitoring sensor networks are deployed for.

6.5.3 Energy Savings

I'd like to end this section with a note on the potential energy savings achieved by performing adaptive data downloads. The relationship between the amount of data downloaded and the time required to keep the radio on is shown in Figure 6.10. The desired accuracy of reconstruction determines how much data needs to be downloaded and this relationship is shown in Figure 6.17. When 50% of the data is downloaded, the median accuracy of $0.06°C$ and the 95^{th} percentile error is 0.0325 using the Gaussian method of reconstruction.

At the time of writing this dissertation, the Koala download protocol suffers from a lot of overheads associated with waking up the network, setting up connections, downloading link information etc. A considerable amount of time is spend in packet retransmissions too.

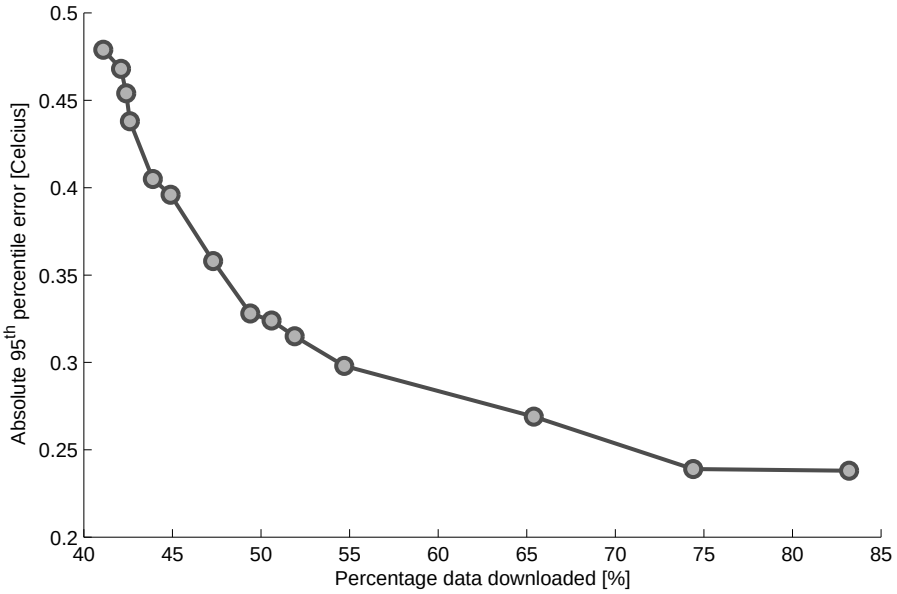


Figure 6.17: Trade-off between reconstruction error and the percentage of data downloaded by the network. The LEF download strategy is used in this figure.

This time is comparable to the amount of time spent in performing data downloads. The duty cycle, at Cub Hill, has been found to be around $2.5\% - 3\%$. This translates to the radios being used for around 1080 seconds in a 12 hour period. Using numbers obtained in Figure 6.10 and assuming these overhead remains constant, the total amount of time the radio would be kept on works out to 855 seconds in a 12 hour period. This is a 20.8% reduction in the amount of time each motes' radio is kept on.

Koala wakes up the entire network and this analysis is very specific to Koala. A download protocol that does not require to wake up the whole network will result in significantly more energy savings.

Chapter 7
Conclusion

In this chapter, I would like to express some closing thoughts on this journey. Specifically, I would like to discuss some interesting lessons learned while on the road, and my opinions on directions that would be interesting for someone addressing similar challenges.

A lot of effort and experience went into the design and maintenance of the two phase data pipeline. However, after over five years of deployment experiences we are still learning and finding ways to improve the system. The collection of metadata information always posed problems. There have been significant delays in updating the metadata tables (location, node, sensor etc.) during the initial stages of deployment and when replacing malfunctioning hardware during the deployment. These delays have often resulted in inconsistencies in the data and difficulties in our ability to trace back the origins of individual measurements. As I have stressed in Chapter 3, designing a system that detects changes in hardware configurations and auto updates the metadata tables would alleviate a lot of these inconsistencies.

Tight coupling of the upload application with the underlying stage database and Koala packet structure resulted in a lot of frustration. Every time the underlying stage database schema changed or the Koala packet structure changed, the upload application would break and require an update. To begin with, data transferred from the basestation to the data store should be free from the underlying download protocol packet structure. The basestation could

use a utility like 'rsync' to upload the outstanding data (or files) to the data store. If a self-describing format is used to describe the records sent by the basestation, a light-weight parser could periodically run through a list of new files and insert the records into the database. This methodology also clearly demarcates the role of the basestation and database server.

Postmortem assignment of timestamps was a fun and challenging problem to work on. Phoenix alleviated a lot of challenges related to random mote reboots and missing global clock sources. By looking at the fit residuals, we had a good way of estimating the accuracy of the skew (α) for a given fit. This metric did not enable us to detect poor estimates of the offset (β). During the Brazil deployment, we encountered that a few motes were advertising a local clock value that was consistently off, whilst maintaining the local clock correctly for its own operation. Whichever segments used these incorrect clock values to obtain a fit suffered from having fits where the β were off by a huge margin. Since each segment could potentially help other segments obtain a clock fit, it resulted in cascading failures. This error was detected when the timestamp assignments were validated by looking at actual timeseries plots. Some signals were consistently out-of-sync with others. In practice, there are a large number of ways of estimating the fit parameters. Each independent path to a node that carries the global clock source provides us with an estimate. We can use this redundancy and employ a voting/collaborative approach to eliminate poor estimates of offsets.

The timestamp assignment problem is a highly focused and specific problem. In contrast, the data-driven adaptive data collection proved to be a subjective and open ended problem. There are many avenues to explore further in this topic. The current system does not consider the amount of work each node needs to do in order to send packets to the basestation. Nodes closer to the basestation do less work and take less time to send data to the basestation compared to those that are multiple hops away. In finding the subset of informative locations, the hop count of the node should be a factor in deciding the importance of the sampling

location.

In many environmental monitoring applications, more than one variable may be of interest to the scientists. Researchers may want to study the variation and distribution of two variables (Soil CO_2 and soil moisture, say) jointly. Adaptively downloading data to match this joint criterion has not been explored in this thesis.

To summarize, this thesis explores challenges related to loading, storing and ensuring the integrity of the data retrieved from typical environmental WSNs. The unique and interesting problem of postmortem timestamp assignment is tackled in depth - both in simulation and in real deployments. Finally, a data-driven approach leveraging the spatial correlation is presented and trade-offs between fidelity and network lifetime are studied.

Bibliography

[1] Baltimore-Washington International airport, weather station. Available at: http://weather.marylandweather.com/cgi-bin/findweather/getForecast?query=BWI.

[2] A Robust Classification of Galaxy Spectra: Dealing with author = Connolly, A. J. and Szalay, A. S., Noisy and Incomplete Data. *The Astronomical Journal*, 117:2052–2062, May 1999.

[3] Andreas Krause. Submodular Function Optimization. Available at http://www.mathworks.com/matlabcentral/fileexchange/20504.

[4] Guillermo Barrenetxea, François Ingelrest, Gunnar Schaefer, and Martin Vetterli. The hitchhiker's guide to successful wireless sensor network deployments. In *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 43–56, New York, NY, USA, 2008. ACM.

[5] Guillermo Barrenetxea, François Ingelrest, Gunnar Schaefer, Martin Vetterli, Olivier Couach, and Marc Parlange. Sensorscope: Out-of-the-box environmental monitoring. In *IPSN '08: Proceedings of the 7th international conference on Information processing in sensor networks*, pages 332–343, Washington, DC, USA, 2008. IEEE Computer Society.

[6] Jug bay wetland sanctuary. Conserving and protecting the jug bay ecosystem. Available at : http://www.jugbay.org/.

[7] Bora Beran and David Valentine and Catharine Van Ingen and Ilya Zaslavsky and Tom Whitenack. Observations Data Model. Available at http://his.cuahsi.org/odmdatabases.html.

[8] P. J. Brockwell, R. A. Davis, and I. Netlibrary. Introduction to time series and forecasting. 2002.

[9] T. Budavári, V. Wild, A. S. Szalay, L. Dobos, and C.-W. Yip. Reliable eigenspectra for new generation surveys. *mnras*, 394:1496–1502, April 2009.

[10] Nicolas Burri, Pascal von Rickenbach, and Roger Wattenhofer. Dozer: ultra-low power data gathering in sensor networks. In *Proceedings of the 6^{th} IPSN Conference*, 2007.

[11] Doug Carlson, Jayant Gupchup, Rob Fatland, and Andreas Terzis. K2: a system for campaign deployments of wireless sensor networks. In *Proceedings of the 4th international conference on Real-world wireless sensor networks*, REALWSN'10, pages 1–12, Berlin, Heidelberg, 2010. Springer-Verlag.

[12] Alberto Cerpa, Jennifer L. Wong, Louane Kuang, Miodrag Potkonjak, and Deborah Estrin. Statistical Model of Lossy Links in Wireless Sensor Networks. In *Proceedings of IPSN 2005*, May 2005.

[13] M. Chang, C. Cornou, K. Madsen, and P. Bonett. Lessons from the Hogthrob Deployments. In *Proceedings of the Second International Workshop on Wireless Sensor Network Deployments (WiDeploy08)*, June 2008.

[14] Yang Chen, Omprakash Gnawali, Maria Kazandjieva, Phil Levis, and John Regehr. Surviving sensor network software faults. In *SIGOPS*, October 2009.

[15] Commonwealth Scientific and Industrial Research Organisation (CSIRO). 2-year progress report: July 2004 to June 2006, 2004.

[16] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition*. McGraw-Hill Science/Engineering/Math, July 2001.

[17] Crossbow Corporation. MICAz Specifications.

[18] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *Proceedings of VLDB*, 2004.

[19] Richard O. Duda and Peter E. Hart. Use of the Hough transformation to detect lines and curves in pictures. *Commun. ACM*, 15(1):11–15, 1972.

[20] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification*. Wiley, 2001.

[21] Prabal Dutta, Jonathan Hui, Jaein Jeong, Sukun Kim, Cory Sharp, Jay Taneja, Gilman Tolle, Kamin Whitehouse, and David Culler. Trio: Enabling sustainable and scalable outdoor wireless sensor network deployments. In *IEEE SPOTS*, pages 407–415, 2006.

[22] J. E. Elson, L. Girod, and D. Estrin. Fine-grained network time synchronization using reference broadcasts. In *Proceedings of the* 5^{th} *Symposium on Operating Systems Design and Implementation (OSDI)*, pages 147–163, December 2002.

[23] William C. Forsythea, Edward J. Rykiel Jr., Randal S. Stahla, Hsin-i Wua, and Robert M. Schoolfield. A model comparison for daylength as a function of latitude and day of year. *ScienceDirect*, 80(1), January 1994.

[24] John Fox. Robust regression: Appendix to an r and s-plus companion to applied regression, 2002.

[25] S. Ganeriwal, R. Kumar, and M. B. Srivastava. Timing-sync protocol for sensor networks. In *Proceedings of the* 1^{st} *ACM Conference on Embedded Networked Sensor System (SenSys)*, pages 138–149, November 2003.

[26] David Gay, Phil Levis, Rob von Behren, Matt Welsh, Eric Brewer, and David Culler. The nesC Language: A Holistic Approach to Networked Embedded Systems. In *Proceedings of Programming Language Design and Implementation (PLDI) 2003*, June 2003.

[27] Jayant Gupchup, Razvan Musaloiu-Elefteri, Alexander S. Szalay, and Andreas Terzis. Sundial: Using sunlight to reconstruct global timestamps. In *EWSN*, pages 183–198, 2009.

[28] Yuan He, Lufeng Mo, Jiliang Wang, Wei Dong, Wei Xi, Tao Chen, Xingfa Shen, Yunhao Liu, Jizhong Zhao, Xiangyang Li, and Guojun Dai. Poster: Why Are Long-Term Large-Scale Sensor Networks Difficult? Lessons Learned from GreenOrbs. In *MobiCom*, 2009.

[29] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for network sensors. In *Proceedings of ASPLOS 2000*, November 2000.

[30] P. W. Holland and R. E. Welsch. Robust regression using iteratively reweighted Least-Squares. *Communications in Statistics: Theory and Methods*, A6:813–827, 1977.

[31] P. Huang, H. Oki, Y. Wang, M. Martonosi, L. Peh, and D. Rubenstein. Energy-efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet. In *Proceedings of the Tenth Internationla Conferece on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X)*, October 2002.

[32] Taesung Kim, Luis Grimaldo, Jayant Gupchup, Andreas Terzis, and Jordan Raddick. A visualization engine for the life under your feet project. http://dracula.cs.jhu.edu/luyf/en/tools/VZTool/Default.aspx.

[33] Andreas Krause, Ajit Singh, and Carlos Guestrin. Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies. *J. Mach. Learn. Res.*, 9:235–284, June 2008.

[34] K. Langendoen, A. Baggio, and O. Visser. Murphy loves potatoes: experiences from a pilot sensor network deployment in precision agriculture. In *Proceedings of the Parallel and Distributed Processing Symposium (IPDPS)*, April 2006.

[35] Ming Li, Deepak Ganesan, and Prashant Shenoy. Presto: feedback-driven data management in sensor networks. In *Proceedings of the 3rd conference on Networked Systems Design & Implementation - Volume 3*, NSDI'06, pages 23–23, Berkeley, CA, USA, 2006. USENIX Association.

[36] Martin Lukac, Paul Davis, Robert Clayton, and Deborah Estrin. Recovering temporal integrity with data driven time synchronization. In *IPSN*, pages 61–72, April 2009.

[37] Liqian Luo, Chengdu Huang, Tarek Abdelzaher, and John Stankovic. EnviroStore: A cooperative storage system for disconnected operation in sensor networks. In *INFOCOM*, 2007.

[38] Alan Mainwaring, Joseph Polastre, Robert Szewczyk, David Culler, and John Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of 2002 ACM International Workshop on Wireless Sensor Networks and Applications*, September 2002.

[39] R. A. Maronna, R. D. Martin, and V. J. Yohai. *Robust statistics*. John Wiley & Sons, 2006.

[40] Miklós Maróti, Branislav Kusy, Gyula Simon, and Ákos Lédeczi. The flooding time synchronization protocol. In *SenSys*, pages 39–49, November 2004.

[41] Warren A. Marrison. The evolution of the quartz crystal clock. *The Bell System Technical Journal*, 27, 1948.

[42] J.R. McNeill and V. Winiwarterv. Breaking the sod: Humankind, history, and soil : Soils: The Final frontier. . *Proceedings of Science*, 2004.

[43] David L. Mills. Internet time synchronization: The network time protocol. *IEEE Transactions on Communications*, 39:1482–1493, 1991.

[44] Razvan Musaloiu-E., Chieh-Jan Liang, and Andreas Terzis. Koala: Ultra-low power data retrieval in wireless sensor networks. In *Proceedings of the 7^{th} international symposium on information processing in sensor networks (IPSN)*, pages 421–432, April 2008.

[45] National Estuarine Research Reserve. Jug Bay weather station (cbmjbwq). Available at http://cdmo.baruch.sc.edu/QueryPages/anychart.cfm.

[46] D.E. Newell and R.H. Bangert. Temperature compensation of quartz crystal oscillators. In *17th Annual Symposium on Frequency Control. 1963*, pages 491–507, 1963.

[47] Kevin Ni, Nithya Ramanathan, Mohamed Nabil Hajj Chehade, Laura Balzano, Sheela Nair, Sadaf Zahedi, Eddie Kohler, Greg Pottie, Mark Hansen, and Mani Srivastava. Sensor network data fault types. *ACM Trans. Sen. Netw.*, 5:25:1–25:29, June 2009.

[48] Consortium of Universities for the Advancement of Hydrologic Science Inc. Observations Data Model. http://his.cuahsi.org/odmdatabases.html.

[49] Joseph Polastre, Jason Hill, and David Culler. Versatile Low Power Media Access for Wireless Sensor Networks. In *Proceedings of the 2^{nd} ACM Sensys Confence*, 2004.

[50] Joseph Polastre, Robert Szewczyk, and David Culler. Telos: Enabling Ultra-Low Power Wireless Research. In *Proceedings of the Fourth International Conference on Information Processing in Sensor Networks: Special track on Platform Tools and Design Methods for Network Embedded Sensors (IPSN/SPOTS)*, April 2005.

[51] M Jordan Raddick, Georgia Bracey, Pamela L Gay, Chris J Lintott, Phil Murray, Kevin Schawinski, Alexander S Szalay, and Jan Vandenberg. Galaxy zoo: Exploring the motivations of citizen science volunteers. *Astronomy Education Review*, 9(1):15, 2009.

[52] Nithya Ramanathan, Thomas Schoellhammer, Eddie Kohler, Kamin Whitehouse, Thomas Harmon, and Deborah Estrin. Suelo: human-assisted sensing for exploratory soil monitoring studies. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, SenSys '09, pages 197–210, New York, NY, USA, 2009. ACM.

[53] Theodore S. Rappaport. *Wireless Communications: Principles and Practice (2nd Edition)*. Prentice Hall PTR, 2 edition, January 2002.

[54] Carl Edward Rasmussen. Gaussian processes for machine learning. MIT Press, 2006.

[55] János Sallai, Branislav Kusy, Ákos Lédeczi, and Prabal Dutta. On the scalability of routing integrated time synchronization. In *EWSN*, volume 3868, pages 115–131. Springer, 2006.

[56] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, 1975.

[57] Andre Santanche, Suman Nath, Jie Liu, Bodhi Priyantha, and Feng Zhao. Senseweb: Browsing the physical world in real time. In *Demo Abstract*, Nashville, TN, April 2006.

[58] The sensor data bus. http://www.sensordatabus.org/default.aspx.

[59] The Sloan Digital Sky Survey SkyServer. 2002.

[60] A. Sharma, L. Golubchik, and R. Govindan. On the prevalence of sensor faults in real world deployments. In *IEEE Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, 2007.

[61] Abhishek B. Sharma, Leana Golubchik, and Ramesh Govindan. Sensor faults: Detection methods and prevalence in real-world datasets. *ACM Trans. Sen. Netw.*, 6:23:1–23:39, June 2010.

[62] Baltimore Ecosystem Study. Research on metropolitan baltimore as an ecological system. Available at : http://www.beslter.org/.

[63] R. Szewczyk, A. Mainwaring, J. Anderson, and D. Culler. An Analysis of a Large Scale Habitat Monitoring Application. In *Proceedings of SenSys 2004*, November 2004.

[64] Robert Szewczyk, Joseph Polastre, Alan Mainwaring, and David Culler. Lessons from a Sensor Network Expedition. In *Proceedings of the* 1^{st} *European Workshop on Wireless Sensor Networks (EWSN '04)*, January 2004.

[65] K. Szlavecz, A. Terzis, R. Musaloiu-E., C.-J. Liang, J. Cogan, A. Szalay, J. Gupchup, J. Klofas, L. Xia, C. Swarth, and S. Matthews. Turtle Nest Monitoring with Wireless Sensor Networks. In *Proceedings of the American Geophysical Union, Fall Meeting*, 2007.

[66] Jay Taneja, Jaein Jeong, and David Culler. Design, modeling, and capacity planning for micro-solar power sensor networks. In *IPSN '08*, pages 407–418, 2008.

[67] A. Terzis, R. Musaloiu-E., J. Cogan, K. Szlavecz, A. Szalay, J. Gray, S. Ozer, M. Liang, J. Gupchup, and R. Burns. Wireless Sensor Networks for Soil Science. *International Journal on Sensor Networks*.

[68] Texas Instruments Incorporated. MSP430 Datasheet.

[69] Gilman Tolle, Joseph Polastre, Robert Szewczyk, Neil Turner, Kevin Tu, Phil Buonadonna, Stephen Burgess, David Gay, Wei Hong, Todd Dawson, and David Culler. A Macroscope in the Redwoods. In *Proceedings of the 3^{rd} ACM SenSys Conference*, November 2005.

[70] Twitter. An information network. Available at http://www.twitter.com/.

[71] Vaisala Industrial Instruments. CARBOCAP Carbon Dioxide Transmitter Series GMT220 specifications. Available at http://www.vaisala.com/instruments/products/gmt220.html.

[72] Jillian C. Wallis, Christine L. Borgman, Matthew S. Mayernik, Alberto Pepe, Nithya Ramanathan, and Mark H. Hansen. Know thy sensor: Trust, data quality, and data integrity in scientific digital libraries. In *ECDL*, pages 380–391, 2007.

[73] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh. Fidelity and Yield in a Volcano Monitoring Sensor Network. In *Proceedings of the 7^{th} USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2006.

[74] Geoffrey Werner-Allen, Stephen Dawson-Haggerty, and Matt Welsh. Lance: optimizing high-resolution signal collection in wireless sensor networks. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, SenSys '08, pages 169–182, New York, NY, USA, 2008. ACM.

[75] Yong Yang, Lili Wang, Dong Kun Noh, Hieu Khac Le, and Tarek F. Abdelzaher. Solarstore: enhancing data reliability in solar-powered storage-centric sensor networks. In *Mobisys*, pages 333–346, New York, NY, USA, 2009. ACM.

[76] Marco Zú Zamalloa and Bhaskar Krishnamachari. An analysis of unreliability and asymmetry in low-power wireless links. *ACM Trans. Sen. Netw.*, 3(2):7, 2007.

[77] Jerry Zhao and Ramesh Govindan. Understanding packet delivery performance in dense wireless sensor networks. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, SenSys '03, pages 1–13, New York, NY, USA, 2003. ACM.

Appendix A

Database Schemas

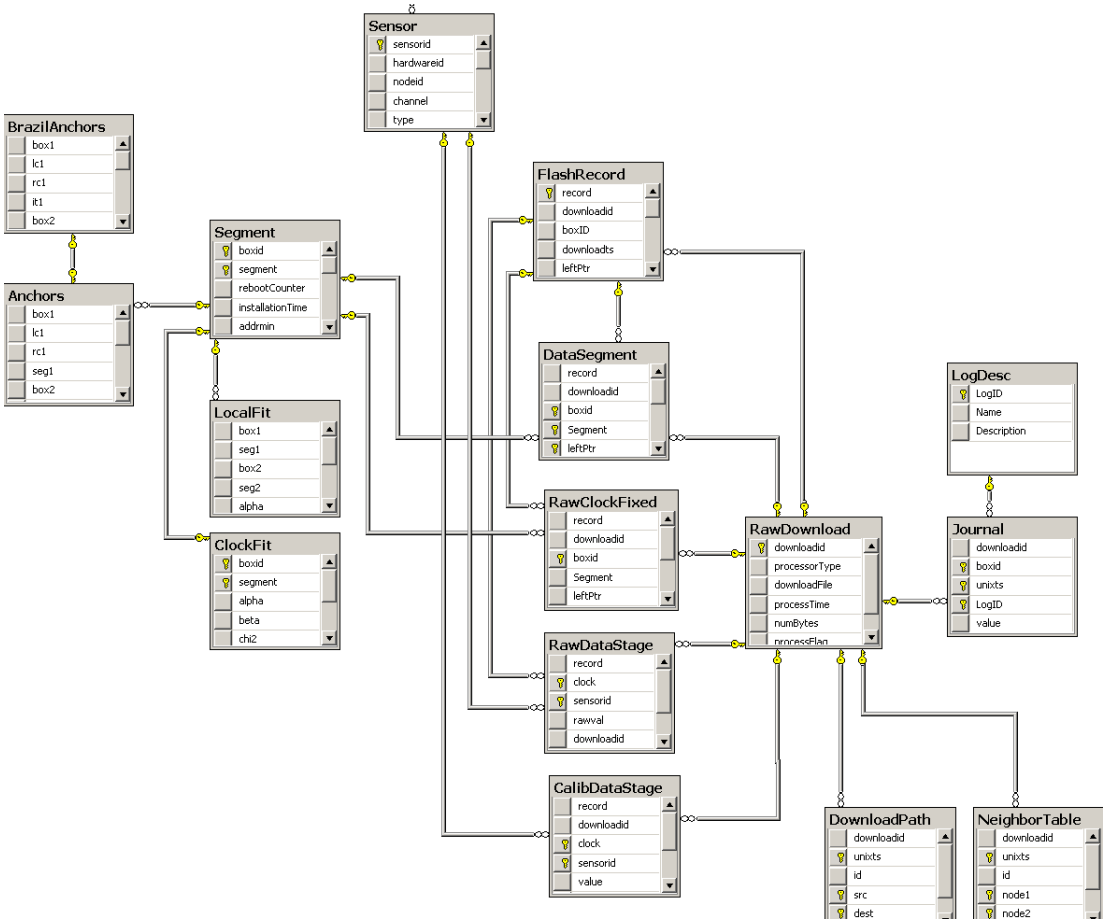


Figure A.1: Schema for the stage database (excludes the metadata tables).

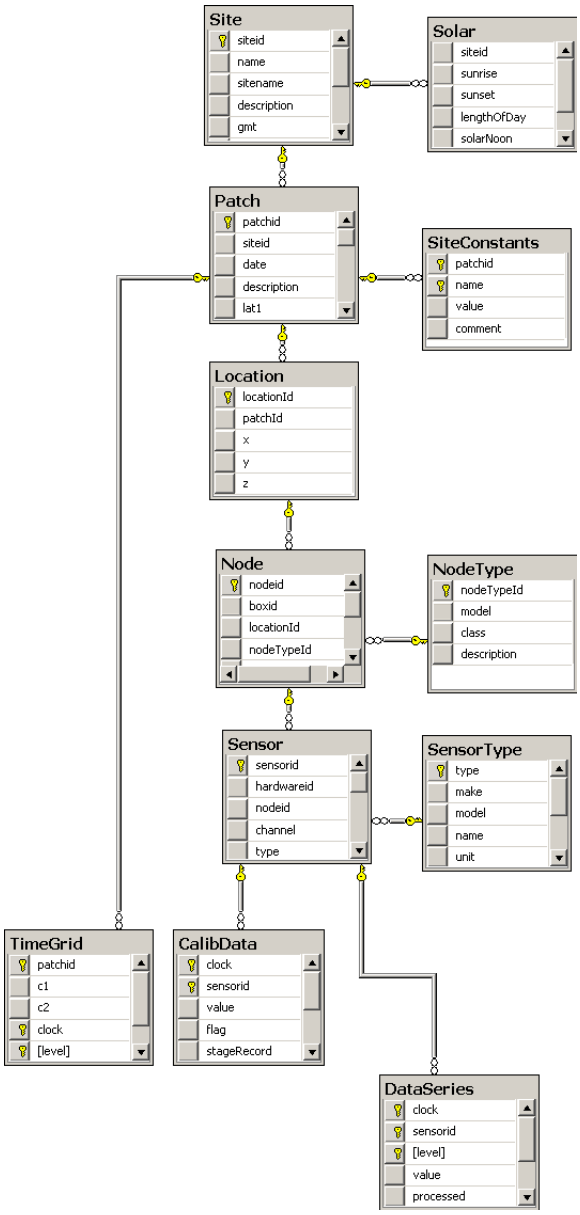


Figure A.2: Schema for the science database.

Vita

Jayant Gupchup received his Bachelors in Computer Engineering from Mumbai University in 2003. From Sep 2003 to July 2005, he worked at the Inter-University Centre for Astronomy and Astrophysics (IUCAA) under the supervision of Prof. Ajit Kembhavi. In Fall of 2005, he began his Ph.D. at the Department of Computer Science at the Johns Hopkins University. His research focusses on data management in long-term environmental monitoring networks, and he is jointly advised by Dr. Andreas Terzis and Prof. Alex Szalay. In 2007, he interned at the Microsoft Bay Area Research Center to work with Dr. Catharine Van Ingen. He received a masters in Applied Mathematics and Statistics in May 2010 under the supervision of Prof. Carey Priebe. After his Ph.D., he will join the parallel data warehousing team at Microsoft.