In-Situ Prediction on Sensor Networks Using Distributed Multiple Linear Regression Models

by

Elizabeth Ann Basha

B.S., Computer Engineering, University of the Pacific (1998)
S.M., Electrical Engineering and Computer Science, Massachusetts Institute of Technology (2005)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2010

Author .....
Department of Electrical Engineering and Computer Science
May 21, 2010

Certified by .....
Daniela Rus
Professor
Thesis Supervisor

Accepted by .....
Terry P. Orlando
Chairman, Department Committee on Graduate Theses

In-Situ Prediction on Sensor Networks Using Distributed Multiple Linear Regression Models

by

Elizabeth Ann Basha

Submitted to the Department of Electrical Engineering and Computer Science
on May 21, 2010, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

Within sensor networks for environmental monitoring, a class of problems exists that requires in-situ control and modeling. In this thesis, we provide a solution to these problems, enabling model-driven computation where complex models are replaced by in-situ sensing and communication. These prediction models utilize low-computation, low-communication, and distributed algorithms suited to autonomous operation and multiple applications. We achieve this through development of new algorithms that enable distributed computation of the pseudoinverse of a matrix on a sensor network, thereby enabling a wide range of prediction methods.

We apply these models to three different application areas: (1) river flooding for early warning, (2) solar recharging current for power management, and (3) job congestion prediction on multi-function device networks for achieving quality of service. Additionally, we use these applications to explore other aspects of sensor networks: river flooding to design a predictive environmental monitoring sensor network, solar current to develop a dynamic version of the model for better fault tolerance, and job congestion to explore modeling multi-function device networks. For each, we comprehensively tested the full solutions. We implemented the river flood prediction and solar current prediction solutions on two different sensor network platforms with full field deployments; we had a final test of over 5 weeks operation for both.

Overall, we achieve the following contributions: (1) distributed algorithms for computing a matrix pseudoinverse and multiple linear regression model on a sensor network, (2) three applications of these algorithms with associated field experiments demonstrating their versatility, (3) a sensor network architecture and implementation for river flood prediction as well as other applications requiring real-time data and a low node count to geographic area ratio, and (4) a MFD simulator predicting and resolving congestion.

Thesis Supervisor: Daniela Rus
Title: Professor

Acknowledgments

I have many people and organizations to acknowledge and thank for their support throughout the adventure that was my doctorate degree. I literally would not have begun without my advisor, Prof. Daniela Rus, who allowed an unknown computer architecture student join her distributed robotics group. I am incredibly grateful for her for taking that chance, supporting me throughout, allowing me to make my own path, and challenging me to achieve more than I thought possible.

My committee of Prof. Piotr Indyk, Prof. Sam Madden, and Dr. Sai Ravela provided support not only in guiding my final thesis work, but during the process as well. My thanks for the discussions we had along the way. Prof. Jacob White provided invaluable academic advice and always supported my work; thank you for both.

I could not have found a better research group than the Distributed Robotics Laboratory and very much enjoyed providing the stationary side of our distributed robotics spectrum. Thank you all for the support, advice, laughter, and knowledge you provided. A special thanks to Carrick Detweiler who helped throughout, whether solving the latest "why does this not work now" problem, planning the best approach to research problems, analyzing the latest technology trends or current events (otherwise known as procrastinating), or simply enjoying the overwhelming sunshine of our shared cubicle area. Marek Doniec helped immensely with random last minute soldering tasks for field deployments, a never ending supply of new project ideas, and an unique perspective that made any discussion entertaining. I am also grateful to Brian Julian for accompanying me on one of my Honduras trips, providing a perpetually cheery attitude and a willingness to do whatever work was needed.

Each of my projects had its own support, both people and financial. For my solar project, I am grateful to CSIRO ICT Centre, Brisbane for allowing me to spend two months working with them, especially Raja Jurdak and Ben Mackey. I appreciate NSF EAPSI program and the Australian Academy of Science for funding the trip and project. For the job congestion project, Xerox Research provided financial and

research support; thank you for granting me the fellowship and the opportunity to work with you. A special thanks to Naveen Sharma and his team for working with me during the summer on the project.

The river flood project truly started me on this path and I have many to thank. For providing financial support, I would like to acknowledge the NSF Graduate Fellowship program, the IDEAS competition, the Carrol L. Wilson Award, Microsoft Research, the Martin Family Society of Fellows for Sustainability, and the MIT Public Service Center. For their advice, support, and continued belief in my work, I am grateful to Steve Banzaert, Dr. James Bates, Alison Hynd, Johanna Kiefner, Sandy Lipnoski, Amy McCreath, Laura Sampath, Amy Smith, and Patricia Weinmann. Sally Susnowitz of the MIT Public Service Center especially supported this work, providing funding and advice at many critical points. However, the project would truly never have succeeded without the many fellow students who worked with me on this project, either through D-Lab or the FloodSafe Honduras student group, including: Patrick Barragan, Bill Boos, Emma Brunskill, Kathleen Connolly, Alejandro Flores, Homero Flores Cervantes, Vanessa Hsu, Teresa Liu, Alejandra Menchaca Brandan, Tony Parolari, Sarah Rich, Adam Rivers, Andrea Rivers, Victor Grau Serrat, Marta Fernandez Suarez, Edgar Terrero, Tadd Truscott, Eric Wang, and Josh Weaver. A few people require individual thanks for their support and friendship, in this project and in so many other instances. Kristen Wendell helped form our student group, found and won funding to support all the work of the group, engineered solutions, traveled numerous times, and always provided a calm, friendly source of advice, support, and fun. Alex Bahr made far too many cables, planned the most transportation-ally diverse trip I have ever taken, helped solve many complicated in-field problems, and supported my work throughout its many transformations. Andrea Llenos was always willing to solve whatever random engineer problem arose, to learn about some random topic the rest of us found boring, to glue more items than we all thought possible, and to join in whatever random plan or adventure I devised (or stumbled into).

Outside of my work, many ensured I did not go crazier. I was especially lucky to have wonderful roommates, including Dave Wentzlaff and Jake Mundt. Audra Bartz spent many hours with me, discussing life and providing perspective. The Warehouse Music Program provided many hours of stress relief and fun; many thanks to Lori Lerman for teaching me how to sing and discussing the vagaries of MIT life. I enjoyed many hours of socializing and spirtualizing with the Tech Catholic Community. Thank you all for your support and letting me participate in so many activities; special thanks go to Father Clancy, Loni Butera, and the 9:30 choir. Through TCC, I met many people who became my family while at MIT: Bridget Englebretson, Steve Englebretson, Scott Litzleman, Andrea Llenos, Maureen Long, Jared Markowitz, Christine Ng, and Ashley Predith. Thank you for sharing many hours of fun, eating all my cooking experiments, and being there through everything. I am grateful to my pre-MIT friends for their understanding and continuing friendship, especially Kayla Davis, Kristen MacNaughtan, and Stephanie Olivera.

Finally, my family dealt with quite a lot and supported me through it all. My thanks to my siblings, Stephanie and Michael, for all their support and love despite not knowing which country I was in at any given time and my decreasing ability to discuss anything other than my thesis. I am grateful to my step-mother, Julie, for ensuring I remembered life exists outside of graduate school and translating the crazy world of graduate school to the rest of my family. To my Dad, thank you for loving and encouraging me despite not quite understanding what I did or why that meant I kept having to leave the country. To my Mom, thank you for getting me through the beginning. I am just so sorry you did not get to see the end.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Sensor networks hold great potential for improving environmental monitoring, helping provide a continuous stream of data to illuminate unknown aspects of our physical world. Continuous data observations made in both time and space supply a huge advantage in all fields where more accurate models improve our understanding and enlighten our actions. With more data we could better understand the existence and impact of climate change, exploring the changes occurring in different ecosystems over time and modeling these changes to better predict future changes. With more data we could better predict natural disasters, reducing the impacts of hurricanes, earthquakes, and forest fires. With more data we could better monitor rainforests, increasing our knowledge of the animals and plants in our world. More data, both temporal and spatial, supplied continuously improves both our knowledge and safety, and sensor networks have the ability to provide exactly the data we need.

Currently, we have many theoretical models, but few data sets. To obtain data, common practice involves manual measurement of small patches of an environment for discrete portions of time, data loggers recording some small part of an environment, or satellites monitoring very large areas. People measuring can never provide a continuous data set, data loggers only record for a small portion of time and fail often, and satellites still need ground truth measurements to be useful. All of this led to the conception of sensor networks over a decade ago, as described in a paper by Estrin *et al.* [31] for example. Sensor networks consist of nodes that measure the

environment and communicate amongst themselves, allowing distribution of the data being collected as well as monitoring of the health of the network.

Since the emergence of sensor networks, research has accomplished much in development of hardware, networking, and middleware. Sensor networks have recorded inside the nests of birds [62], explored the tall heights of the redwoods [92], monitored the coral reefs of the sea [96], and protected people from natural disasters [6]. They have sensed everything from the small world of a single tree [92] to the large world of the Swiss Alps [9].

Yet much of this work focuses on a sense-and-send method of operation, with nodes sensing their environment and sending that information to a single node or office. Many applications operate perfectly well under this method such as those recording a small set of data in an easily accessible location, but many applications could benefit from other operational methods such as those in more remote regions or with larger data requirements. In the example of climate change monitoring, some models require fine-grained, large data sets, but sending all of this data to a single location is too difficult and too expensive. In ecosystem monitoring, to understand the details of certain ecosystems necessitates varying the frequency of the measurements in the presence of specific events, this can require local control to compute how to modify the measurement schedule and also could generate too much data to communicate. In any underwater monitoring, communication of any data is difficult; even the sense-and-send model does not work in this environment. Overall, many applications generate large amounts of local data and need to compute something locally; if the sensor network has to forward all the data and process it offline, the system will not scale to larger network sizes or more complicated network operations. In this thesis, we propose a class of problems requiring both intensive computation and minimal communication. We solve these problems through distributed algorithms utilizing local computations based on locally sensed, shared data and limited neighbor communication.

To illustrate the difficulty of this problem and our contributions, we explore a real-world scenario in ecosystem monitoring, although not one we solve in this thesis,

that of monitoring endangered species. According to the International Union for Conservation of Nature and Natural Resources (IUCN), the status of 15% of the world's mammals is unknown due to insufficient data (categorized by the IUCN as data deficient) [53]. This includes the Manzano Mountain Cottontail, a rabbit inhabiting the Manzano Mountains in New Mexico [84]. This rabbit lives in the high altitude conifer forest region of the mountains, making it hard to monitor, measure, and track, which also complicates the process of determining its threat of extinction.

To gain a more complete picture of the numbers and behaviors of the rabbits, we could install a sensor network in the mountains to record all key variables of the ecosystem as well as camera tracking of the animals. We would need to perform some control of the system to optimize data collection and reduce energy usage to maximize the life of the network. Some of this will have to include detection of the rabbits within the images to decide what images to save and identify relevant rabbit sightings. To avoid constantly taking pictures or missing all the rabbits, we could also consider predicting when rabbits might appear based on our past time history of rabbit sightings in our images. This prediction would allow the sensor network to optimize its measurement schedule and globally ensure a long lived system. Some sensors will need to measure during supposedly unlikely time windows to verify the prediction; which set of sensors this encompasses can rotate among all sensors to maximize the life of the system.

We run into several problems in implementing such a sensor network; the first results from the difficulty of sending the data to a central location. The Manzano Mountain region is potentially too remote for connecting the network to an office (40 km from Albuquerque with very limited road and trail access) as is true for many environmental monitoring sensor networks. If we do engineer a way to send information to an office, whatever solution we generate will have to function in a very remote area with limited power (no grid) and all components hiked in (also limiting the power system size and weight). These are not unique problems to sensor networks as prior system that perform similar tasks (such as [9] and [100]) also experienced these limitations in installing remote systems. When the solution does fail, maintenance

will take time so the sensor network control cannot rely on centralized direction. Finally, the images are too large to send to a single site. (Assume a webcam takes a 320×240 pixel image requiring 8 kbytes with compression. With message sizes of 32 bytes to 128 bytes, this would require approximately 60 to 260 messages, which is a significant number of transmissions given that communication dominates the energy budget of a node.) With no ability to send all the data to a central office, having central control of the algorithms to detect rabbits and predict sightings is improbable and unwieldy. All of these issues require algorithms running autonomously on the sensor network in order to control these, and possibly other, tasks.

Our next problem is how to model the event we wish to predict. We want easily deployable systems that function in a variety of environments, which requires models that do not need pre-existing data sets or expert calibration. This rules out the usual class of physically-based models, which predict based on equations that describe the physical processes leading to an event. To do so, these models utilize data sets describing the specific environment and require hand calibration of their parameters, both factors negating their usability. The first is especially undesirable as it ensures a sensor network system cannot simply be implemented or moved to a new environment, but must have additional calibration and setup beforehand. We also face problems where we have no model such as our rabbit population monitoring problem where we do not know specifics about the species. Avoiding both these problems requires creating models that self-calibrate and require little foreknowledge of the environment. This leads us to statistical models, which utilize a local record of data to learn models that predict events of interest.

In performing this type of prediction, the sensors collectively maintain a set of data measured by the network and define a time history of relevant variables. The sensor network then computes a statistical model using this data set to predict the variable of interest. The model only requires that all variables be described by a time series, removing any need for outside control and only initial expertise in defining the variables to use when first designing the model.

Once we have statistical models running within the sensor network, we could see

an issue with large data sets. Statistical models inherently rely on large quantities of data. The sensor network can gather this data, but often a single sensor network node cannot store the full matrix of data. To store the matrix, the sensor network needs to distribute the data among the nodes, with each storing only a portion of the matrix. Once we distribute the data, distributing the algorithms provides the most efficient approach to computing the model. Without knowing or limiting the model description in advance, our best practice is distributing the model to fit all classes of modeling problems.

We now need to develop new distributed algorithms that compute statistical models within the limitations of a sensor network. We focus on a class of models called multiple linear regression models, a class that only relies on the local data record, requires limited computation for the prediction phase of the model, and self-calibrates to provide good predictions of events. Multiple linear regression models require several matrix operations, the most complex being a matrix pseudoinverse. To the best of our knowledge, distributed computation of the matrix pseudoinverse does not exist for sensor networks; we need to develop these algorithms.

We also need to consider the modified requirements for a sensor network computing complicated distributed algorithms compared to one designed for a strictly sense-and-send operational method. The additional computational complexity, data needs, and distributed algorithms will affect the hardware design in regards to processing, data storage, and communication reliability.

In understanding the Manzano Mountain Cottontail, then, we meet a class of environmental monitoring problems where we achieve the best solution by performing in-situ control and modeling. The nature of the environment, consisting of a remote, inaccessible location with little known about the problem and large amounts of data generated to understand it, requires an autonomous and robust sensor network solution. This solution requires modeling to intelligently utilize resources and, because of the lack of knowledge, sensor network limitations, and modularity requirements for the model, statistical models suggest the best solution. By choosing statistical models, we need to develop algorithms for distributed matrix operations. While this

adds complexity to our work, by choosing statistical models we also gain a modular, self-calibrating mechanism for predicting a wide array of events.

Overall, in this thesis, we focus on adding intelligence to a sensor network, enabling in-situ, distributed prediction of events. We develop distributed algorithms to compute statistical models in the network, models requiring nothing other than the measurements taken by the system. We then apply these models to three different application areas: (1) river flooding for early warning, (2) solar recharging current for power management, and (3) job congestion for achieving quality of service. Additionally, we use these applications to explore other aspects of sensor networks: river flooding to design a predictive environmental monitoring sensor network, solar current to develop a dynamic version of the model for better fault tolerance, and job congestion to explore modeling device networks.

1.1 Distributed Linear Prediction

One of the most useful statistical models is the multiple linear regression model. This model combines any set of input variables to predict an output variable of interest, only requiring that all variables can be represented by a time series. Because of the lack of restrictions on the model, multiple linear regression models can represent a wide variety of scenarios and utilize the key output of a sensor network: local, in-situ environmental measurements. These models also self-calibrate, requiring no expert knowledge or data, and can adapt the model structure to changing network conditions. Running these models on a sensor network requires computing a matrix pseudoinverse, a key step in calibrating the model and a computationally complex operation. Chapter 3 describes our distributed linear prediction algorithms to run these models including our novel methodology to compute the pseudoinverse in a distributed fashion on a sensor network.

We outline the format of multiple linear regression models for the general case including the parameters describing the model, the requirements of the model, and the overall operation of the model. We divide the model into two stages: (1) prediction

and (2) calibration. Our prediction algorithm allows for distributed computation, aggregating the result at the node most interested in the prediction. For calibration, we divide the problem into three areas: (1) algorithms for a distributed pseudoinverse, (2) issues related to large matrix sizes, and (3) optimizations of the distributed pseudoinverse for the case of a linear regression model. In the first area, we develop algorithms for a distributed pseudoinverse based on breaking the problem into three sections: a QR decomposition to reduce issues related to the rectangular nature of the data matrix, a singular value decomposition to compute the needed sub-matrices, and a final pseudoinverse step to combine the sub-matrices into the desired pseudoinverse. Often, in computing the pseudoinverse, issues arise whereby the initial data matrix is too large to store within the network; our second area explores reducing the initial matrix size through random sampling methods. Finally, in the third area, we return our original issue: using the pseudoinverse for calibrating a linear regression model. This allows for several optimizations of the computation including a different form of the pseudoinverse combination step. We test all algorithms in simulation, verifying their functionality. Then, we implement the multiple linear regression model on our sensor network and perform a field experiment to demonstrate the functionality in the general case before moving into our specific applications.

1.2 River Flooding

River flooding seriously threatens the lives of people around the world. According to the United Nations International Strategy for Disaster Reduction, floods accounted for 32% of disasters worldwide from 1991 through 2005, the largest percentage of all disaster types [94]. Mitigating these disasters requires predicting their occurrence, monitoring their occurrence, and monitoring post-disaster conditions to reduce disaster effects. While urban areas of developed countries have systems to provide this information, rural areas and developing countries remain vulnerable, lacking the technology and infrastructure to provide appropriate mitigation information. Chapter 4 outlines our work in developing statistical models for river flood prediction and a

sensor network architecture appropriate for solving this problem.

We describe how to model river flooding and use our distributed algorithms to predict future river level 24 hours in advance. For this prediction, the river level model only requires three local measurements: river level, rainfall, and air temperature. To demonstrate and compare our model, we use sensor data and physically-based river level model results from the Blue River in Oklahoma. We then outline a sensor network architecture for monitoring rivers in remote and large geographic areas as well as computing prediction models in-situ. Our sensor network supports a wide variety of sensor types, heterogeneous communication of real-time data via a two-tier communication structure, and fault tolerance for enabling longer term installations. It operates autonomously to control system behavior and compute event predictions. Finally, we discuss our simulation results proving the effectiveness of our model in predicting future river level, our initial field experiments verifying the sensor network system, and our full system field experiments demonstrating the distributed algorithms operating on the sensor network. We test and deploy our system in two locations: (1) Dover, Massachusetts, our local test site, and (2) the Aguan River in Honduras, a more challenging and realistic test site.

1.3 Solar Current

Ensuring a robust sensor network requires careful power management and many projects have explored different approaches to solving this problem from optimized hardware design for low power operations to software policies for modifying system behavior. However, all of these ignore or overly simplify the future solar recharge current, which defines the future available energy on which the system relies. Knowing the future available energy allows for more intelligent power management and a decrease in extreme behaviors such as full shut-down of the node. Utilizing a distributed model to perform this prediction shares both the computation and the prediction among the nodes, reducing the individual energy consumption required. It also enables a more spatial representation of the existing conditions, which can better

reflect the broader local environment. Chapter 5 describes a prediction model for average daily solar current, providing this input for any power management system.

We model and implement of our distributed algorithms for this problem. In this approach, each node utilizes its own solar current measurements, its neighbors' solar current measurements, and available meteorological measurements from itself or neighbors. This allows for predicting daily average solar current 1 or more days in advance. We describe our simulations to verify the model for this problem, implementation on the Fleck platform (a different architecture than that developed in Chapter 4), and field experiments. We installed 3 nodes in a month-long deployment, predicted future solar current autonomously on the nodes, and verified the overall approach. Finally, we extend our model to a more dynamic version that allows for better fault tolerance and model development. This dynamic version provides external control of the model parameters defining the matrix (such as the meteorological variables used and the calibration time window), which will enable in-network control in future work.

1.4 Job Congestion

Multi-function devices (MFDs) provide print, copy, fax, and scan capabilities along with the ability to perform complicated processing tasks associated with these operations. Networking a group of them together enables sharing the processing tasks among devices and improving the overall system efficiency. However, in cases where only a few devices have the needed capabilities, job congestion can occur with these devices being overloaded and the quality-of-service guarantees no longer being met. Predicting job congestion provides a method for modifying local behaviors to avert global problems as well as avoiding the need for administrative oversight of the network. In this scenario, like wireless sensor networks, these device networks are processing and communication limited in performing these additional tasks. A device must allow the regular operations to access the processor, limiting the amount of side computation it can perform for congestion prediction. Additionally, the devices may

not connect to a high-bandwidth network and need to allow regular network operations to proceed without impact, limiting the amount of network traffic the devices can generate. Due to these limitations, a distributed regression-based model, such as we designed, provides the best solution. Chapter 6 describes how our algorithms can also predict job congestion in this constrained environment.

To verify the functionality, as these device networks are still being developed, we introduce a network simulator to describe their behavior and allow system testing. We define the expected behaviors of a MFD network based on individual device characteristics and develop our MFD-specific simulator in SimPy, a python-based discrete event simulator. In the context of this network simulator, we define a job congestion model where congestion relates to the job queue length, a measurable parameter within each device. Unlike our other applications, our model relies on more abstract parameters that, while measurable, are not sensed values external to the device, but rather internal variables such as job queue length, reputation, and number of jobs of a service type. This abstract parameterization is supported by our model structure and distributed algorithms, which we show through the simulator. We begin by analyzing our network simulator, then verify the model operation through simulation results, and finally implement our congestion model within the network simulator for a complete experiment of our work.

1.5 Thesis Contributions

Overall, our work contributes to the goal of an intelligent, autonomous, and robust environmental sensor network. Specifically, these contributions entail:

- A novel set of distributed algorithms for computing a matrix pseudoinverse and multiple linear regression model on a sensor network

- Exploration of randomized methods for reducing matrix size and implementation optimizations of the pseudoinverse algorithms depending on usage

- In the context of river flooding, design and implementation of a sensor network system (architecture, hardware, and software) optimized for large geographic areas and predictive environmental monitoring
- A statistical model and related implementation on our sensor network for predicting river flooding
- A statistical model and related implementation on the Fleck sensor network for predicting solar current
- A statistical model and related implementation on our network simulator for predicting job congestion on MFD networks
- A network simulator for MFD networks
- Extensive knowledge and experience regarding environmental sensor networks, from field experiments and deployments, providing lessons learned for future sensor network developers

1.6 Thesis Organization Summary

To summarize the organization of this thesis, this thesis consists of 7 chapters.

Chapter 2 outlines related work for all aspects of this thesis.

Chapter 3 presents a novel set of algorithms for computing a matrix pseudoinverse and multiple linear regression model. It explores several issues associated with implementing these algorithms including randomized methods for reducing matrix size and optimizations depending on usage.

Chapter 4 outlines our river flood prediction scenario. In predicting river flooding, we designed a sensor network architecture optimized for large geographic areas and predictive environmental monitoring. We also developed a statistical model for predicting river flooding that performs better than current research models and enables easy deployment in any river basin.

Chapter 5 describes solar current prediction. For solar current prediction, we enable better power management and demonstrate the transferability of the algorithm

to a different sensor network platform. Additionally, we demonstrate a dynamic version of the model which will enable better model development and fault tolerance of the algorithms.

Chapter 6, our last application, extends the work to predicting job congestion on multi-function device networks, which shares the limitations of our sensor networks. This scenario also demonstrates the ability of our models to predict more abstract concepts where the model uses variables that are not physically measured from the environment. To verify this we also develop a network model of these device networks as the physical systems are still under development.

Finally, Chapter 7 summarizes all of this work, outlines future work (including solving the Manzano Mountain Cottontail problem), and draws on the knowledge generated to formulate lessons learned.

Chapter 2

Background and Related Work

In this chapter, we cover the related work for all aspects of our thesis. In Section 2.1, we explore work related to distributed linear prediction on sensor networks. Then, in Sections 2.2, 2.3, and 2.4, we discuss the use of distributed prediction in each of our application areas.

2.1 Distributed Linear Prediction

A rich history exists in computing the matrix pseudoinverse and the linear regression prediction associated with it. We first cover the distributed pseudoinverse as it relates to our problem in the wider sense and then discuss related work specifically in sensor networks.

2.1.1 Distributed Pseudoinverse

Computing a matrix pseudoinverse has interested researchers for quite a while and numerous mathematical solutions exist. Golub and Van Loan [39, 40] introduce and discuss many of these from a centralized and mathematical perspective. Implementing this math through distributed algorithms on a multi-processor system provides an additional set of research questions. In this area, we focus strictly on the earlier methods that utilized processors with limited capabilities, such as those used in the 1980's. The limitations of these systems are similar to the limitations of current sensor

network systems; one could argue that a sensor network is a multi-processor system with an unusual communication system. By focusing on these earlier methods, we do not disregard that the research continues in this area, but rather recognize that the systems such research runs on are powerful enough to allow solutions far too complex for operation on sensor networks.

These earlier methods focus in three areas: (1) computing the pseudoinverse in the context of least squares solutions to overdetermined systems (such as introduced in [40, 70]), (2) solving the underlying SVD with the recognition that the pseudoinverse is defined within the context of the SVD [40], and (3) solving the pseudoinverse directly. The first area utilizes methods that do not directly compute the pseudoinverse, but use the features of the least squares solutions to optimize out the need for that direct computation of it. As we wish to provide both the distributed matrix pseudoinverse computation and a distributed linear regression computation in order to enable a large range of future modeling and computation opportunities, we focus on the latter two areas of solutions.

Distributing the SVD commonly utilizes block Jacobi methods as they allow for easy parallelization. Golub and Van Loan [39] introduce the overall method. Brent *et al.* [17] discusses a method of computing utilizing a fixed hardware system optimized for the SVD. The processors were placed in a grid array with a fixed communication system and only computed the block computation based on their equivalent, fixed location in the data matrix. Processors could compute blocks of size 2×2 and larger, allowing the system to compute a wide array of SVD results although the only solution for rectangle matrices was to pad them with zeros until they were square. Transferring this to more general systems occupied most of the research following this work. Bischof [12] describes methods to compute the block Jacobi on a hypercube multi-processor system; in [13], he utilizes a multiple vector processor machine. Ewerbring and Luk [32] implement a method tailored to the Connection Machine. These latter methods rely on the early work of Brent *et al.* [17] as do we although our method utilizes the unique communication systems of a sensor network as well as dealing with the limitations.

In computing the pseudoinverse directly, approximate solutions exist such as that developed by Benson and Fredrickson [8] to compute a pseudoinverse on a hypercube system. For exact methods, the Gauss-Jordan approach or other elimination methods are often used. Milovanovic *et al.* [64] describes some of the past work with Gauss-Jordan and introduces a method for a linear row of processors. One issue with this type of approach compared to the SVD approach is the numerical issues that can arise, especially in zeroing the elements. With zeroing the elements, the problem arises whereby the computation result almost reach zero do not equal zero, differing just enough to cause the algorithm to fail. Avoiding this requires pivoting [40], which changes rows in order to avoid having to zero elements that would cause problems. Parallel implementations commonly avoid this due to complexity and communication costs.

Instead of these approaches, control projects utilize the Block Matrix Inversion Lemma [22], which divides the original matrix into 4 components: A, which must be invertible and square; B and C, which have no requirements; and D which also must be invertible and square [43]. While this method allows for easy updates of the inverted matrix and does divide the problems into blocks, it does require a large amount of computation within the smaller blocks. Additionally, the requirements on the data to guarantee the invertibility of the sub-matrices do not make it a good choice for our work as we cannot, and do not want to, provide this guarantee.

We need a solution that does not restrict our data matrix and provides an exact, precise solution since we will use it for prediction. Elimination methods do not provide the precision necessary and approximation methods are not exact enough. This leaves orthogonal methods, which we utilize. However, we optimize these methods for a sensor network with its combination of very limited memory and wireless communication where broadcasting to all is easier than point-to-point communication with each node. These differences require innovations in the computation of the distributed pseudoinverse.

2.1.2 Prediction and Regression in Sensor Networks

To our knowledge, no prior work develops a distributed pseudoinverse for wireless sensor networks; this is a useful contribution of our work to the general sensor network community. In the areas of performing distributed regression and prediction on a sensor network, a small amount of research exists.

Delouille *et al.* [28] develop an algorithm to provide the linear minimum mean-squared error estimator to determine the true values of noisy measurements. The algorithm uses a decomposition of a loopy graphical model and an iterative Gauss-Seidel method. This requires a matrix inverse, which they perform centrally on each node. The solution is tested in simulation, but not implemented on a system; it is unclear how they would perform the inverse or computation within a constrained sensor node on a microcontroller.

McConnell and Skillicorn [63] describe a method for prediction on a sensor network whereby each node computes a local prediction and then the centralized system hub votes on the correct method. They do not provide a local prediction algorithm, but use a standard, centralized method in simulation to verify the use of the voting scheme. The approach is distributed only in the sense that a single non-sensing node distributes the vote results; the system does not compute the prediction in a distributed fashion.

Banerjee *et al.* [4] discuss how to perform polynomial regression on a sensor network. The computation requires a pseudoinverse at every calibration step. Testing this in simulation, they compute the pseudoinverse using Mathematica. They then provide a method for updating coefficients that does not require a matrix inverse based on starting from an initial computation that does. This is tested in Matlab using an initial set of temperature data with an initial inverse calibration step. As the error increases over the short time window of the simulation, it is unclear this update could correctly predict in the long term without further centralized calibrations. How to compute the initial calibration with the matrix inverse on the sensor network is not discussed.

Predd *et al.* [72, 73] describe a regularized kernel least-squares regression approach based on reproducing kernel Hilbert spaces from machine learning. They develop an approach more suited to sensor networks by relaxing the constraints on the equivalency of sensor functions. However, calibrating this model requires a pseudoinverse, which they note needs to be computed and suggest can be computed centrally. This primarily theoretical work was tested in simulation to verify operation, but not implemented on a sensor network.

Guestrin *et al.* [41] provide an distributed regression algorithm to help reduce the amount of data the network needs to communicate while allowing reasonable reconstruction of the node measurements back at the source. Specifically, they are utilizing a sensor network of Motes installed in a lab measuring temperature. Given the high correlation of temperature data, they wish to describe the data by a function allowing them to generate the data at the base computer instead of communicating the data. Their approach assumes sparse matrices and utilizes kernel linear regression, a special case of linear regression. In implementing kernel linear regression, they use Gaussian elimination to compute the weighting parameters. Our approach uses a different, orthogonal method of computation; an orthogonal method decreases the limitations on the matrix structure and removes the numerical issues seen with Gaussian elimination [40].

2.2 River Flooding

Previous work covers a wide variety of topics including sensor networks for environmental monitoring, sensor networks for flood detection, and operational flood detection systems.

Sensor Networks for Environmental Monitoring

Several sensor network systems have been designed for outdoor monitoring purposes especially animal monitoring. While this work does not directly relate to ours, implementations sharing some interesting characteristics include cattle ranch monitoring [82], cattle control [18, 79], sheep monitoring [91], zebra herd monitoring [57, 101],

seabird nests [62], and frog vocalizations [47]. Of greater relevance is work in environment monitoring where several projects have implemented related systems.

Tolle *et al.* [92] developed a sensor network to monitor a redwood tree. Installing nodes throughout the height of the 70 m tree, the system measured air temperature, relative humidity, and solar radiation over a 44 day period. The system logged data every 5 minutes and transmitted it via GPRS modem to an external computer. All analysis was performed off-line after the test period.

Selavo *et al.* [80] created a sensor network for measuring light intensity. Each node can connect to 8 resistive or voltage-based sensors, communicating data locally via Zigbee and remotely via a single Stargate at 2.4 GHz with delay tolerance of the data arrival at the base station. They performed a field experiment of 1 day with 7 nodes and have installed 19 sensor nodes in another experiment (but no results were available at time of publication). No data analysis occurred on the nodes.

Guy *et al.* [42] built a sensor network system that has been installed in four different locations to date. In the James Reserve, a forest setting, the system measured temperature, humidity, rain, and wind using up to 27 nodes over 1.5 years. The project installed 2 nodes for 1.5 years in a high-desert farm and 24 nodes in the UCLA Botanical Gardens for 3 months. Finally, a 12-node system was installed in a Bangladesh rice paddy for 2 weeks to measure nitrate, calcium, and phosphate (this experiment also described in [76]). These nodes used 433 MHz communication systems to share the data measured and a base station sent the data for offline analysis. The goal of the researchers for the system was portability and rapid deployment, focusing on a very different set of requirements than our system.

Werner-Allen *et al.* [100] installed a wireless sensor network on a volcano in Ecuador, running 16 nodes for a 19 day test. Their system focused on scientific effectiveness, specifically the quality of the data and quantity measured, allowing for delays in data gathering as long as correctly timestamped. The nodes measured seismic and acoustic data, transmitting to each other at 2.4 GHz and back to the base station through a single repeater node at 900 MHz. Detection of recordable signals did occur on the system, but no further data analysis occurred within the network.

Beutel *et al.* [9] designed a sensor network for measuring permafrost regions. Their latest deployment placed 15 nodes on the Matterhorn starting operation in July 2008. The system focuses on surviving the hostile environment with high quality data measurements.

While the above systems do share some characteristics to the system and problem we describe, none envision the level of heterogeneity our system requires, the minimalistic number of sensors available for the extensive network area, the real-time need for the data, or the computational autonomy and complexity necessary to perform the prediction operation.

Sensor Networks for Flood Detection

Previous work on sensor networks for flood detection is sparse with only three different examples discovered in the literature. Castillo-Effen *et al.* [19] suggested an architecture for a system, but were unclear on the basin characteristics and presented no hardware details. Zhou *et al.* [102] described a project called FloodNet that directly includes sensor network measurements into a central flood model; this project did not operate the model on the sensor network or perform detection within the network making it different from the project we propose.

Closest to our work is a project by Hughes *et al.* [49, 51, 48, 50, 52] and Coulson *et al.* [27], designing a flood-predicting sensor network that uses Gumstix sensor nodes, which require significant power but allow for a Linux operating system to run on the node. Optimizations in [49] reduced the power of the node by including a secondary processor. Field tests have been performed according to [48, 50]. According to the papers, the system would run a point prediction hydrology model with each node computing its own centralize prediction; it is unclear whether it currently runs on their system. From reviewing the related citations [10, 78], it appears their model follows more traditional physically-based hydrology approaches. This solution works well when the necessary calibration data is available; when the data is not available, as in our problem definition, this solution is not possible. In addition to the apparent lack of model implementation on the flood prediction side, the known details of the

hardware platform dismiss it as an immediate solution to the problem introduced here as it has limited geographic range, high cost, and power requirements that may be, in the long-term, unsustainable.

Current Operational Systems for Flood Detection

While not specifically sensor network installations, understanding the current operational systems helps clarify the problem space in which we are working. The lack of published information on operational flood systems makes generalizations difficult, but three systems seem to summarize the approaches currently taken.

One type involves a highly technical solution with significant resource support such as seen in the US. For this system, companies develop sensor, communication, and computation technology based on the ALERT protocol, which defines the data structure and polices of environmental monitoring systems [3]. The US Emergency Alert System provides communication of the alerts throughout the nation using television and radio channels by creating special technology and policies, requiring the installation of the technology in stations across the country along with weekly testing, and ensuring protocol compliance at all levels [34]. Implementation of specific systems trickles through each level of government: federal, state, and county. Given the large number of counties in the US, systems and policies do vary, but the majority rely on large numbers of personnel (some highly technical) and significant technical resources. Usually, counties implement the direct measurement system with help from the USGS and create polices on how their county defines a disaster and evacuation procedures. Actual prediction usually depends on qualified hydrologists examining the data (thus removing measurement errors) and running it through a complicated physical model described below.

Another type is the system commonly seen in Central America, especially Honduras [5], relying on volunteers and limited technology. Volunteers read the river level off of markings painted on bridges and the rain level from water collecting gages (also emptying the gage) at several intervals during a day, radioing that information to a central office run by the government. In that office, a person listens to the radio,

records the values in a book, and compares them to a defined policy whereby the river level measured corresponds to a color alert. This color alert is radioed to the head office of the government branch, which then decides on the need for an evacuation alert in that region and implements some form of emergency alert procedures. Overall this system relies on very little technology and extensive policies to warn communities, working best in small river basins where measurements indicate flooding in that area (as opposed to downstream of the measurement area).

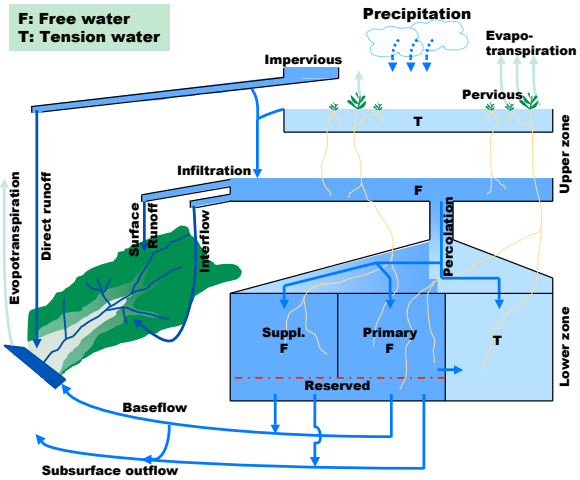

Figure 2.1: Example of Current Physically-based Model: Sacramento Soil Moisture Accounting (SAC-SMA)

A third solution exists in Bangladesh, a country regularly devastated by flooding due to its low sea level and large rivers. To combat this, the Danish Hydraulic Institute initially outfitted the country with local telemetry stations in 1995 and created a MIKE 11-based flood forecasting system [56]. However, this system experienced sustainability problems along with issues due to the fact that the headwaters of its major flood-causing rivers originate in India, creating complexities with monitoring. A solution to this was created by a global community of researchers and government institutions, collating all of the satellite information and forecasts generated by the US to provide short, medium, and long-term flood predictions of the major basins [44, 99]. A system called the Flood Forecasting and Warning Response System distributes the alert through reports submitted to various government agencies along with a variety of public media sources [23]. This takes advantage of the ubiquity of

satellite information, which looks to provide input data for flood forecasting systems of the future [45]. The success of the system does rely on very regular satellite passes, still not common in all parts of the world, and a large amount of US resources, also not available everywhere.

Computation Requirements of Current Operational Flood Prediction Model

We discuss flood prediction models in general in Section 4.1; however of most relevance to this work are current operational models, especially that used on our prototype data sets. In the US, the current operational model works by modeling the different methods of rainfall surface runoff to determine how much water will enter the river, thus increasing the level. Called the Sacramento Soil Moisture Accounting model (SAC-SMA), it predicts runoff out to 12 hours based on rainfall over the area. It creates three different water compartments (see Figure 2.1 [20]): a zone describing the direct runoff from rain falling on impervious soils, a zone describing water flowing into the river after exceeding the soil moisture capacity of pervious soils, and a zone describing runoff occurring after soil moisture capacity is exceeded above water impervious regions [37]. The model describes each zone using several differential equations along with several more complex equations describing the interconnection of these zones into a single surface runoff value. These clearly cannot easily run on a sensor network.

Determining the actual computation time of these equations is difficult as examining two papers that outline some information on this provides different information. Experiments by Vrugt *et al.* [98] on autocalibration methods using this model resulted in 25 minutes for calibration on a Pentium IV 3.4 GHz computer. This calibration appears to involve running the model. Other work by Ajami *et al.* [2], also in the area of parameter autocalibration includes a figure displaying the run time and calibration time for a number of calibration methods where it appears that running the model requires on the order of hours. However the paper doesn't specify numbers. Of the two papers, the first paper is more specific regarding computational numbers but

much less detailed on procedure whereas the second is clearly using the SAC-SMA model over the same data set we also use. Either way, the information from both suggests that the model requires more computational power and time than available on a sensor network.

Additionally, these equations use 11 parameters, not all corresponding to actual physical, measurable quantities [69]. To calibrate these parameters and the model requires at least 8 years of rainfall and runoff data for calibration, ideally 8 years of further data for verification, detailed topographic maps, and hand-calibration by trained hydrologists [35]. The resulting model operates only on that basin; model creation for a different basin requires 8 years of calibration data for the new river and expert hand-calibration. This again does not work for a sensor network nor regions where such data does not exist (and putting sensors in for 8 years to gather enough information is impractical).

2.3 Solar Current

Past research projects into energy management focus on either reducing power consumption through system design and/or through modeling and control. The system design approach minimizes the power usage of the hardware components and optimizes the power harvesting system to maximize the system lifetime, an approach seen in [25, 55, 90]. The other approach focuses on the software side, managing power through modeling and control. Most of these projects use a fixed parameterization of the harvested energy or respond to direct measurements; [33, 61, 65, 97] demonstrate these types of projects.

Very little work complements these approaches by predicting the future harvestable energy to input into the models. Hsu *et al.* [46] and Kansal *et al.* [58, 59] examined a prediction method using an Exponentially Weighted Moving Average Model. This model provides a centralized approach allowing each node to compute its own prediction based on a combination of current observations and past prediction history. Evaluation utilized existing data sets in simulation; no implementation oc-

curred on a platform. We compare this approach in our simulation results in Section 5.2.

Moser *et al.* [66] performed offline linear programming techniques to predict future energy in addition to control methods. The paper describes a simulation of this with no field instantiation.

Our work also provides these complementary methods and provides an energy prediction to input. However, our work utilizes a more complex and richer set of input data, and provides in-situ, local predictions, which we test in a field experiment on a sensor network platform. We provide a prediction of energy that could support any energy management plan on any low-power hardware platform.

2.4 Job Congestion

Predicting congestion on multi function device (MFD) networks is a specialized area of research. MFDs have not existed for very long and little research exists in that area. We can look to two related areas, network congestion and anomaly detection, in addition to the existing research in MFDs.

Network congestion research focuses more on control issues: how can we generate policies that avoid congestion within the network and how can we remove existing congestion. Talaat *et al.* [89] provided a survey of work in congestion control strategies, with an overall interest in media streaming. As the paper points out, the common router approach to congestion is to drop packets, leaving control to the end points of the network. At these endpoints, strategies attempt to reach the best communication rate that allows messages to reach their destination and not induce congestion. Overall, the methods do not try to predict congestion within any given device and strictly focus on congestion mitigation.

Within the sub-area of asynchronous transfer mode (ATM) networks, Corral *et al.* [26] introduced AI techniques to predict congestion. These techniques rely on genetic programming and occur centrally within a node.

Anomaly detection more closely approaches our problem specification in that it

attempts to detect anomalous behavior, in which we could include congestion. Chandola *et al.* [21] provided a comprehensive survey of the area. Within anomaly detection, research approaches of most interest are those that utilize regression. Bianco *et al.* [11] demonstrated an example of this using an ARIMA model, while Galeano *et al.* [36] utilized a vector ARMA model. Common to these approaches is the use of the model either centrally or offline, and the use of the model to demonstrate common behavior where an event not matching the prediction is consider anomalous. This does not use the model to predict the anomaly, but the absence of the anomaly.

All of the above do not quite match the characteristics needed by our problem; most importantly, none focus on distributed approaches. We then can examine the specific area of MFD research, which is a new area of research. Gnanasambandam *et al.* [38] first introduced the concept of networking MFDs. Quiroz *et al.* [74, 75] furthered the concept by looking at clustering techniques to detect anomalies within the network. To the best of our knowledge, our work is the first to explore device congestion prediction within an MFD network.

Chapter 3

Distributed Linear Prediction

We developed a low computation, low communication distributed multiple linear regression algorithm to run on a sensor network. This algorithm is general enough for a wide range of applications and we introduce it as a general format in this chapter.

Overall, our distributed algorithms achieve our goals for operation on a sensor network. We compute the same results as the centralized versions while reducing the individual node computational requirements by O(n) where n typically lies between 5-20. This computational reduction is at the cost of communication, which is one key trade-off between distributed and centralized. However, the ability to actually perform the computation provides another key trade-off. Most prediction scenarios require a reasonable data set on which to calibrate the model. For many scenarios, this reasonable data set would utilize approximately 20 kbytes for centralized computation compared to 2 kbytes for distributed, an O(n) decrease in memory usage. Thus, the centralized data set most likely cannot even be stored on a single node. Memory then becomes the key design point in utilizing centralized prediction models compared to our distributed versions. Our algorithms successfully allow for implementation on current sensor network systems, which centralized would not allow.

We start in Section 3.1 by discussing the model: what parameters describe the model, what requirements the model has, and what is the overall operation of the model. With the model structure in place, Section 3.2 outlines our statistical predic-

tion algorithm and its operation. We next construct a set of tools for the calibration of the statistical prediction algorithm. Section 3.3 describes the algorithm necessary to compute a distributed pseudoinverse, a key tool in calibrating the prediction algorithm. Building on the pseudoinverse, we discuss issues arising due to large matrix sizes in Section 3.4. We then complete the discussion of the algorithms by outlining those necessary to connect the pseudoinverse to calibrating a linear regression model and other implementation-related optimizations in Section 3.5. Finally, we test these algorithms through simulation and implementation in Section 3.7, validating their functionality and use.

3.1 Modeling

Our overall goal is to enable prediction of physical phenomenon on sensor networks, where the phenomenon can be captured by a single time series variable. In performing these computations to achieve this overall goal, our goals are to optimize communication (thus saving energy), optimize computation, and optimize the usefulness of the prediction. Achieving these goals requires distributed algorithms and shared storage of the matrix. A useful prediction model requires incorporating a wide variety of data into the matrix, possibly more data than a single node can store. To store the matrix, we need to distribute the data among the network, which provides a secondary benefit by ensuring the prediction is more robust to node failures. Because we have to distribute the data matrix storage, we can distribute the computation also, sharing the computational load and reducing the communication load. This achieves all our goals and motivates our distributed algorithms, which we outline in detail.

A common argument against distributed computation is the existence of a base station or gateway node of higher computational power nearby, or the nearby presence of a central location to provide the computation. Some deployments do fit these scenarios with systems located on campuses or within communication reach via high-powered gateways. However, some deployments and applications do not, especially in the area of environmental monitoring where we focus. Deployments in remote areas

may not have reasonable communication range back to a central office, may not allow for the power necessary to run a higher power gateway (lugging batteries into forests is not always feasible), or may utilize the same processing unit for the gateway as the rest of the network (allowing redefinition of who is the gateway, reducing gateway power requirements, and simplifying the system, to list a couple reasons for this design choice). In these cases, to provide prediction models improving the overall system performance requires distributed computation as centralized computation is just not reasonable. Where gateways do exist that could provide centralized computation, issues of scalability and fault tolerance arise. At some point, the number of nodes needing individual predictions will exceed even the computational capability of the gateway and/or the communication requirements to reach all the nodes will exceed those needed to perform local distributed computations. Additionally, the gateway will fail; it is just another node, albeit a more powerful one. When this node fails, repair may take several days, depending on location, while the system flounders without the centralized control the gateway provided. All of these reasons suggest the usefulness of distributed algorithms, especially to support the overall general case and allow deployment of sensor networks wherever we want, not limited by the need for gateway communication, easy access, and centralized control.

As a method for achieving this, we choose statistical-based multiple linear regression models. These models provide a clean, generic framework for all forms of phenomenon and leverage the key features of a sensor network: a plethora of local, in-situ sensed data and in-system computation of data statistics. Additionally, these models require no prior historical data for operation, allowing self-calibration to occur after the sensor network collects sufficient data. Multiple linear regression models take the form of:

$$b_i = x_{i0}a_{i0} + x_{i1}a_{i1} + ... + x_{in}a_{in} \tag{3.1}$$

where a_i defines the observed variables, b_i is the prediction of the variable of interest, and x_i represents the model parameters, weighting the variables appropriately to predict b_i.

To use these models, we perform the following steps:

- If it is for a new application area, gather data, determine efficacy of model in simulation, and implement on sensor network
- Monitor variables of interest with sensor network until sufficient data collected
- Calibrate model to determine weighting coefficients
- Predict future phenomenon by weighting most recent measurements from the sensor network and combining these values
- Continue loop of: gather data and predict, recalibrating if prediction error exceeds defined metric

For new application areas, we start by defining the structure of the model. We need to first determine that the problem can be solved using a multiple linear regression model so we gather or locate a relevant data set. With this data set, we simulate the model. This enables us to define the correct variables useful for predicting the phenomenon and the amount of calibration data to use. With a model structure in place, we then implement it on a sensor network and monitor the variables of interest.

This model structure is application specific, but the underlying algorithms are not. As such, we define the prediction algorithm in Section 3.2. Prediction requires coefficients computed during calibration which we describe in 3.3. Both prediction and calibration can occur decentralized from each other, allowing for flexibility in implementing them.

3.2 Distributed Prediction

We predict the phenomenon of interest as a linear combination of scaled variables. While this computation could occur centrally (depending on the number of variables), distributing it shares the computation with limited additional cost as the network already communicates measurements and status; we can simply added the scaled variables to these pre-existing messages with only a small number of byte transmission

costs incurred. At this point, we assume the model structure exists and that some outside entity provides our weighting parameters. Later we use our distributed pseudoinverse to provide these parameters, but they could be predefined or determined using another method. We thus compute:

$$b_{t+L} = x_{t0}a_{t0} + x_{t1}a_{t1} + ... + x_{tn}a_{tn} \tag{3.2}$$

in a distributed fashion using Algorithm 3.1. In this format, a defines a node's variables at time t consisting of observations of the variable we are trying to predict and other related variables. The sub-index j indicates which variable as all a_{tj} values are measured at the same time t. Variable x is a vector of the weighting parameters, provided by our distributed calibration algorithm. Variable b is our prediction.

Algorithm 3.1 Distributed Prediction

a_t : observations, at time t, on node i of prediction variable and related variables
j : number of values node i stores
$t + L$: prediction time
b_{t+L} : prediction

for Each node i **do**
    Measure j values of a_t
    $\tau_i = a_{tj}x_{tj}$
    Transmit τ_i

    Receive from all other nodes $\tau_{k\neq i}$
    $b_{t+L} = \tau_i + \tau_{k\neq i}$
end for

In Algorithm 3.1, each node measures some portion of the a values at current time, t. Here we allow nodes to maintain more than one variable of the computation and, while j suggests that each stores the same number of variables, all of the algorithms allow for an unequal number of variables stored by each node (in cases where this reduces communication for example). The node then multiplies these values by its portion of the stored x values and communicates the result to the other nodes. Each node, upon receiving the prediction components from the other nodes, adds these other components to its own, thus computing its own prediction L time intervals in the future, b_{t+L}.

Nodes have two options for determining a prediction of its own variable of interest. First, assuming the variable of one node correlates well with the other nodes, all nodes can collaborate on predicting the variable of one node, computing the prediction value, b_{t+L}. Each node then uses that value as its prediction. To achieve this, each node maintains some portion of the data and associated weighting coefficients. In a centralized prediction, nodes would have to communicate data values to the central node anyway; this distributed form communicates aggregated weighted values instead, reducing the number of values to communicate and sharing the computation.

Or, if the similarity in the variable of interest does not exist, the nodes each compute their own prediction based on their own measurements. Distribution occurs when including neighbors' measurements of the variable at their location and other environmental variables; the latter should correlate well among the nodes allowing all to share the same measurements. With each node performing a prediction, a node already maintains its values of the variable and just needs to store additional weighting coefficients for its neighbors. Sharing environmental variables allows nodes to divide them equally among the network, storing some portion and associated weighting coefficients. The increase in precision due to the increase in data available to the model offsets the potential increase in communication (potential as the number of values communicated is small and the system can append them to existing messages).

The first option reduces computation and communication while generating a reasonable prediction; the second improves each node's prediction with an increased cost in computation and communication. Choosing between the two depends on the structure of the network, similarities between node placements, and requirements for precision in the predictions.

3.3 Distributed Calibration

The calibration step provides the weighting coefficients by computing:

$$X = ((A^T A)^{-1} A^T) b \tag{3.3}$$

where A is the matrix of historical measurements of our variable we want to predict and related variables, b is the vector of past observations of our prediction variable L time steps after the last prediction variable values in A, and X is the computed coefficients. $((A^T A)^{-1} A^T)$ is the standard Moore-Penrose pseudoinverse of a real matrix, which also includes a computationally intensive square matrix inverse and three matrix multiplication steps [40]. To perform this computation on a sensor network centrally is infeasible and undesirable as A is usually too large (3 months of hourly data and 10 single byte values requires 21.6 kbytes) to store on one sensor node (much less all the additional storage necessary for the pseudoinverse computation); therefore we need a distributed algorithm where each node maintains some number of columns. As no distributed pseudoinverse exists for sensor networks, we need to innovate on existing algorithms.

We break the problem into three steps as shown in Algorithm 3.2: (1) a QR decomposition to reduce issues related to the rectangular nature of A, (2) a singular value decomposition (SVD) to compute the necessary pieces, and (3) a final pseudoinverse step to combine all the pieces into a solution. We discuss each of these algorithmic steps in detail.

Algorithm 3.2 Distributed Pseudoinverse

Input: A
Output: A^+

$[Q, R] = qr(A)$	▷ Perform distributed QR according to Algorithm 3.3
$[U, R, V] = svd(R)$	▷ Perform distributed SVD according to Algorithm 3.4
$A^+ = VR^{-1}(QU)^T$	▷ Finish pseudoinverse according to Algorithm 3.7

3.3.1 QR Decomposition

The QR decomposition requires as input any matrix A, which is of size $m \times n$ where $m \geq n$. It then decomposes A into Q of size $m \times n$ and R of size $n \times n$. This decomposition allows for squaring off and shrinking the matrix used by the SVD, a key component of the pseudoinverse.

We compute the QR using a modified Gram-Schmidt form as outlined in Algorithm 3.3 [40]. This form decomposes the matrix in column order allowing us to store some number of columns per node. To ease explanations, we assign an equal number of columns to each node, specifically $2p$ columns per node. Ensuring the number of columns is even eases ease shuffling of data for the SVD step.

At the beginning of the algorithm, each node has p columns of A, consisting of m past data values already gathered by the node. We perform the computation in place, replacing the columns of A with columns of Q. For additional storage, the computation will need np data values stored for R and m temporary values.

The algorithm begins with Node 0, which is the node storing column 0. This node computes the l^2 norm of the Q_0 column ($||Q_0||_2 = \sqrt{\sum_{i=0}^{m} Q_0(i)^2}$) to fill the R_{00} entry and then divides the Q_0 column by that value to obtain the decomposition final value for Q_0 column. It then communicates the Q_0 column to all other nodes. Each node uses the data to update its own columns. If Node 0 stores more columns ($2p > 1$), it retains control and computes final values for both R_{11} and Q_1, communicating Q_1 to the other nodes. Control switches to Node 1 when the algorithm reaches column $2p$. As it finalizes its columns, it communicates them and the algorithm continues, switching control after every $2p$ columns. The algorithm terminates once all columns have been computed, resulting in a Q matrix of size $m \times n$ and a R matrix of size $n \times n$, with Q completely replacing A. Each node maintains $2p$ columns of Q and R. By performing the QR decomposition, we can focus our remaining operations on the much smaller and square R matrix, simplifying communication and reducing concerns about the structure of the matrix (condition, linear independence, etc.).

Algorithm 3.3 QR Decomposition

Input: A
Output: $[Q, R]$

$Q \leftarrow A$
$R \leftarrow 0$

Node 0 begins:
  $R_{00} = ||Q_0||_2 = \sqrt{\sum_{i=0}^{m} Q_0(i)^2}$
  $Q_0 = Q_0/R_{00}$
  Communicate Q_0 to other nodes

for $Node = 0 : n/2p - 1$ **do**
  Node receives Q_i column
  for $k = 2p(Node) : 2p(Node + 1) - 1$ **do**
    $R_{ik} = Q_i^T Q_k$
    $Q_k = Q_k - R_{ik} Q_i$
    if $(i + 1 == 2p(Node))$ *or* $(i == k)$ **then**
      $R_{kk} = ||Q_k||_2 = \sqrt{\sum_{i=0}^{m} Q_k(i)^2}$
      $Q_k = Q_k/R_{kk}$
      Communicate Q_k to other nodes
    end if
  end for
end for

3.3.2 SVD

The Singular Value Decomposition (SVD) uses R, the smaller matrix, to generate the components of the pseudoinverse: U, R, and V; each is a $n \times n$ matrix. The SVD decomposes a matrix, R, such that $R = UDV^T$ where D is a diagonal matrix of the singular values of R, and both U and V are orthogonal basis vectors. We perform a distributed singular value decomposition (SVD) based on a cyclic Jacobi procedure for mesh processors outlined by Brent *et al.*[17] and generalized by Van Loan [95]. Our work utilizes the base building blocks from Brent *et al.* [17], but introduce a more general rotation algorithm optimized for column storage of data stored on systems broadcasting information, such as a sensor network.

Algorithm 3.4 shows the overall column-based computation of the SVD while Algorithm 3.5 demonstrates the fundamental 2×2 lower level block computation. Algorithm 3.6 outlines the rotation step necessary between iterations. Because the SVD step occurs after the QR decomposition, we ensure the input matrix, R, is square with dimensions $n \times n$. If n is odd, the algorithm pads the matrix with a column and row of zeros to make it even. A maximum of $n/2$ nodes can compute the SVD where each node contains at least two columns of the matrix; if less nodes exist, each node contains an even number of columns. For purposes of explaining the algorithm, we define the node as storing a $n \times 2$ column block. For purposes of analysis, we define all algorithms as storing a $n \times 2p$ column block.

This algorithm works on a block level with the smallest block being a 2×2 matrix. Each node computes its diagonal block of the matrix, computing the $[c^1\ s^1]$ and $[c^2\ s^2]$ values needed to diagonalize the block. Nodes transmit these values so all nodes have full vectors; these vectors scale the non-diagonal blocks of the columns, the U matrix, and the V matrix. Once these computations complete, the node rotates the data values according to a defined method spelled out in Algorithm 3.6. As Brent *et al.* used a mesh of processors with each storing a 2×2 block, they defined the rotation as partly built into the hardware architecture and limited by the mesh communication structure. As we store columns of data, we redefined this algorithm to

Algorithm 3.4 Singular Value Decomposition

Input: R
Output: $[U, R, V]$

$\mathbf{U} \leftarrow \mathbf{I}$
$\mathbf{V} \leftarrow \mathbf{I}$
Node $P_j \leftarrow [R_{2j-1}, R_{2j}]$ where each R_j is a $n \times 2$ column block
$StopCondition = eps * off(\mathbf{R})$
while $off(\mathbf{R}) > StopCondition$ **do**
Each node computes diagonal subproblem ($i = j$) according to Algorithm 3.5:

$$\begin{bmatrix} R'_{2i-1,2i-1} & 0 \\ 0 & R'_{2i,2i} \end{bmatrix} = \begin{bmatrix} c_i^1 & s_i^1 \\ -s_i^1 & c_i^1 \end{bmatrix}^T \begin{bmatrix} R_{2i-1,2i-1} & R_{2i-1,2i} \\ R_{2i,2i-1} & R_{2i,2j} \end{bmatrix} \begin{bmatrix} c_i^2 & s_i^2 \\ -s_i^2 & c_i^2 \end{bmatrix}$$

Transmit $[c_i^1 \ s_i^1]$ and $[c_i^2 \ s_i^2]$

Compute other column blocks ($i \neq j$), U, and V:

$$\begin{bmatrix} R'_{2i-1,2j-1} & R'_{2i-1,2j} \\ R'_{2i,2j-1} & R'_{2j,2j} \end{bmatrix} = \begin{bmatrix} c_i^1 & s_i^1 \\ -s_i^1 & c_i^1 \end{bmatrix}^T \begin{bmatrix} R_{2i-1,2j-1} & R_{2i-1,2j} \\ R_{2i,2j-1} & R_{2i,2j} \end{bmatrix} \begin{bmatrix} c_j^2 & s_j^2 \\ -s_j^2 & c_j^2 \end{bmatrix}$$

$$\begin{bmatrix} U'_{2i-1,2j-1} & U'_{2i-1,2j} \\ U'_{2i,2j-1} & U'_{2j,2j} \end{bmatrix} = \begin{bmatrix} U_{2i-1,2j-1} & U_{2i-1,2j} \\ U_{2i,2j-1} & U_{2i,2j} \end{bmatrix} \begin{bmatrix} c_j^1 & s_j^1 \\ -s_j^1 & c_j^1 \end{bmatrix}$$

$$\begin{bmatrix} V'_{2i-1,2j-1} & V'_{2i-1,2j} \\ V'_{2i,2j-1} & V'_{2j,2j} \end{bmatrix} = \begin{bmatrix} V_{2i-1,2j-1} & V_{2i-1,2j} \\ V_{2i,2j-1} & V_{2i,2j} \end{bmatrix} \begin{bmatrix} c_j^2 & s_j^2 \\ -s_j^2 & c_j^2 \end{bmatrix}$$

Rotate columns among nodes according to Algorithm 3.6
Transmit, Receive, and Load columns for next iteration:
if $j = 1$ **then**
Transmit R'_{2j}, U'_{2j}, and V'_{2j} with index $2j - 1$
Receive R'_{2j+2}, U'_{2j+2}, and V'_{2j+2}
$[R_{2j-1}, R_{2j}] = [R'_{2j-1}, R'_{2j+2}]$
$[U_{2j-1}, U_{2j}] = [U'_{2j-1}, U'_{2j+2}]$
$[V_{2j-1}, V_{2j}] = [V'_{2j-1}, V'_{2j+2}]$
else if $j = n$ **then**
Transmit R'_{2j-1}, U'_{2j-1}, and V'_{2j-1} with index $2j$
Receive R'_{2j-2}, U'_{2j-2}, and V'_{2j-2}
$[R_{2j-1}, R_{2j}] = [R'_{2j-2}, R'_{2j}]$
$[U_{2j-1}, U_{2j}] = [U'_{2j-2}, U'_{2j}]$
$[V_{2j-1}, V_{2j}] = [V'_{2j-2}, V'_{2j}]$
else
Transmit R'_{2j-1}, R'_{2j}, U'_{2j-1}, U'_{2j}, V'_{2j-1}, and V'_{2j}
Receive R'_{2j-3}, R'_{2j+2}, U'_{2j-3}, U'_{2j+2}, V'_{2j-3}, and V'_{2j+2}
$[R_{2j-1}, R_{2j}] = [R'_{2j-3}, R'_{2j+2}]$
$[U_{2j-1}, U_{2j}] = [U'_{2j-3}, U'_{2j+2}]$
$[V_{2j-1}, V_{2j}] = [V'_{2j-3}, V'_{2j+2}]$
end if
end while

Algorithm 3.5 2×2 Subproblem Singular Value Decomposition

Input: $R_{2i-1:i,2j-1:j}$
Output: $[c_1, c_2, s_1, s_2]$

$w = R_{2i-1,2j-1};\ x = R_{2i-1,2j};\ y = R_{2i,2j-1};\ z = R_{2i,2j}$
$flag = 0;$
if $(y == 0)\ and\ (z == 0)$ **then**
  $y = x;\ x = 0;$
  $flag = 1;$
end if
$u_1 = w + z;$
$u_2 = x - y;$
if $|u_2| <= eps * |u_1|$ **then**
  $c_0 = 1;\ s_0 = 0;$
else
  $\rho = u_1/u_2;$
  $s_0 = sign(\rho)/\sqrt{1+\rho^2};$
  $c_0 = s_0\rho;$
end if
$u_1 = s_0(x + y) + c_0(z - w);$
$u_2 = 2(c_0x - s_0z);$
if $|u_2| <= eps * |u_1|$ **then**
  $c_2 = 1;\ s_2 = 0;$
else
  $\rho_2 = u_1/u_2;$
  $t_2 = sign(\rho_2)/(|\rho_2| + \sqrt{1+\rho_2^2});$
  $c_2 = 1/\sqrt{1+t_2^2};$
  $s_2 = c_2t_2;$
end if
$c_1 = c_2c_0 - s_2s_0;$
$s_1 = s_2c_0 + c_2s_0;$
$R'_{2i-1,2j-1} = c_1(wc_2 - xs_2) - s_1(yc_2 - zs_2);$
$R'_{2i,2j} = s_1(ws_2 + xc_2) + c_1(ys_2 + zc_2);$
if $flag == 1$ **then**
  $c_2 = c_1;\ s_2 = s_1;$
  $c_1 = 1;\ s_1 = 0;$
end if

Algorithm 3.6 Rotation for Singular Value Decomposition

Each node computes following for all columns:
$RT_{1,j} = R'_{1,j}$
$UT_{1,j} = U'_{1,j}$
$VT_{1,j} = V'_{1,j}$
$index \leftarrow 1$
$s1 \leftarrow 3$
$s2 \leftarrow 5$
for $i = 2 : n$ **do**
  $index = index + s1$
  $RT_{i,j} = R'_{index,j}$
  $UT_{i,j} = U'_{index,j}$
  $VT_{i,j} = V'_{index,j}$
  $s1 = s1 + s2(-1)^{i-1}$
  if $i >= n - 2$ **then**
    if $s2 < 8$ **then**
      $s2 = s2 - 2$
    else
      $s2 = s2 - 3$
    end if
  else
    if $s2 < 8$ **then**
      $s2 = s2 + 1$
    end if
  end if
end for

better run on our planned system with no hardware requirements and optimized for broadcast communication allowed by sensor networks. Finally, nodes transmit these values and store them, in rotated form, back in the matrix for the next iteration. The stopping condition for this algorithm could be either: (1) a fixed number of iterations or (2) a full reduction of the matrix to diagonal form, identified when the l_2 norm of the non-diagonal values is less than some fixed threshold such as the machine numeric accuracy, as shown in Algorithm 3.4. With the latter condition, each node can compute the value for its columns, transmit that data along with the rotation data, and compute when to stop in a decentralized manner.

3.3.3 Pseudoinverse Step

Once we complete the SVD, we combine the various sub-matrices to achieve our pseudoinverse, computing $A^+ = VR^{-1}(QU)^T$. The resultant A^+ matrix is distributed across the network with each node storing $2p$ rows. This algorithm is our own novel development, optimized for the sensor network scenario of broadcast communication and pre-distributed matrices.

Algorithm 3.7 shows the steps necessary to perform the matrix multiplications, transform, and square inverse in a distributed fashion. At the beginning of the algorithm, each node has $2p$ columns of Q, $2p$ columns of U, $2p$ columns of V, and $2p$ columns of R (which is diagonal so only $2p$ values). In the algorithm, each node performs local multiplications on its column blocks before communicating the results to minimize the communication and ensure that no one needs to store an entire $m \times n$ matrix.

Each node begins by computing its local portion of VR^{-1}, which requires no communication, followed by the local computation of the modified V by U. With this step, each node has a component of the final matrix; the addition of all components will provide $VR^{-1}U^T$. Before performing the addition, at this point midway through the algorithm, nodes switch from maintaining $2p$ columns to $2p$ rows to reduce storage and communication costs. Each node stores a row block of its component matrix, transmits the rest, and adds the other components to finalize the matrix multiplica-

tion. To multiply by Q^T, each node performs a local multiplication by its portion of Q, transmits this Q, and updates its row block. By the end of the algorithm, each node has $2p$ rows of A^+.

Algorithm 3.7 Pseudoinverse Combination

Input: $[U, R, V, Q]$
Output: A^+

Node i computes the following:
$V_i = V_i R_i^{-1}$
for $y = 1 : n$ **do**
  for $z = 1 : n$ **do**
    $t_{yz} = \sum_{w=1}^{2p} V_{yw} U_{zw}$
  end for
end for
Node i stores row block t_i of size $2p \times n$ and transmits the rest of t
Upon receipt of values s_i corresponding to the same block as t_i, $t_i = t_i + s_i$

for $y = 1 : 2p$ **do**
  for $z = 1 : m$ **do**
    $a_{yz} = \sum_{w=1}^{2p} t_{yw} Q_{zw}$
  end for
end for
Node i transmits Q_i
for Each row y received **do**
  for $z = 1 : m$ **do**
    $a_{yz} = a_{yz} + \sum_{w=1}^{2p} t_{yw} Q_{zw}$
  end for
end for

3.3.4 Completeness

We now verify that our distributed algorithms provide the same results as the centralized forms.

QR Decomposition

Theorem 3.3.1 *Our distributed QR Decomposition, Algorithm 3.3, provides the same results as the centralized QR decomposition.*

Proof To prove this, we show that $Q_{ij}^d = Q_{ij}^c$ and $R_{ij}^d = R_{ij}^c$.

We define $i \in (0, m)$ and $j \in (0, n)$. For simplicity, $N = n$ nodes exist in the system, such that column j is stored on Node j.

To prove $R_{ij}^d = R_{ij}^c$, we consider two cases: $i = j$ and $i \neq j$. For the condition where $i = j$, R_{jj}^d is computed using the l^2 norm of Q_j^d, which exists within Node j. This is exactly the same step as computing R_{jj}^c. For the condition where $i \neq j$, computing R_{ij}^d involves a vector multiply of the Q_j^d column stored in Node j and the Q_i^d column received from Node i. To compute R_{ij}^c involves the same vector multiply, but both columns, Q_j^c and Q_i^c, exist within the centralized node.

For the Q case, Q_{ij}^d is the result of subtracting the scaled Q_k^d columns for all $k < j$; these columns are received from the other nodes storing them. The final step is a normalization with R_{jj}^d, which is stored locally. The centralized case also performs the same steps, again with all columns stored centrally and no communication requirements.

SVD

Theorem 3.3.2 *Our distributed SVD, Algorithm 3.4, provides the same results as the centralized SVD.*

Proof Algorithm 3.5 is shown and proven by Brent *et al.* [17] while Algorithm 3.4 originates in Brent *et al.* with clarification from Van Loan [95]. The difference we introduce is Algorithm 3.6, which we must show provides equivalent results to that introduced by Brent *et al.*.

For simplicity, let us fix the array size to 8×8. The rotation is equal for R, U, and V; we will focus on R. The initial R is:

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} & r_{14} & r_{15} & r_{16} & r_{17} & r_{18} \\ r_{21} & r_{22} & r_{23} & r_{24} & r_{25} & r_{26} & r_{27} & r_{28} \\ r_{31} & r_{32} & r_{33} & r_{34} & r_{35} & r_{36} & r_{37} & r_{38} \\ r_{41} & r_{42} & r_{43} & r_{44} & r_{45} & r_{46} & r_{47} & r_{48} \\ r_{51} & r_{52} & r_{53} & r_{54} & r_{55} & r_{56} & r_{57} & r_{58} \\ r_{61} & r_{62} & r_{63} & r_{64} & r_{65} & r_{66} & r_{67} & r_{68} \\ r_{71} & r_{72} & r_{73} & r_{74} & r_{75} & r_{76} & r_{77} & r_{78} \\ r_{81} & r_{82} & r_{83} & r_{84} & r_{85} & r_{86} & r_{87} & r_{88} \end{bmatrix}$$

Brent *et al.*would split this among a processor array of 4, arranged in a 2×2 array:

$$\left[\begin{array}{cccc|cccc} r_{11} & r_{12} & r_{13} & r_{14} & r_{15} & r_{16} & r_{17} & r_{18} \\ r_{21} & r_{22} & r_{23} & r_{24} & r_{25} & r_{26} & r_{27} & r_{28} \\ \hline r_{31} & r_{32} & r_{33} & r_{34} & r_{35} & r_{36} & r_{37} & r_{38} \\ r_{41} & r_{42} & r_{43} & r_{44} & r_{45} & r_{46} & r_{47} & r_{48} \\ \hline r_{51} & r_{52} & r_{53} & r_{54} & r_{55} & r_{56} & r_{57} & r_{58} \\ r_{61} & r_{62} & r_{63} & r_{64} & r_{65} & r_{66} & r_{67} & r_{68} \\ \hline r_{71} & r_{72} & r_{73} & r_{74} & r_{75} & r_{76} & r_{77} & r_{78} \\ r_{81} & r_{82} & r_{83} & r_{84} & r_{85} & r_{86} & r_{87} & r_{88} \end{array}\right]$$

The rotation step occurs in the hardware with fixed interconnects between processors. After this hardware based rotation, the array becomes:

$$\left[\begin{array}{cccc|cccc} r_{11} & r_{14} & r_{12} & r_{16} & r_{13} & r_{18} & r_{15} & r_{17} \\ r_{41} & r_{44} & r_{42} & r_{46} & r_{43} & r_{48} & r_{45} & r_{47} \\ \hline r_{21} & r_{24} & r_{22} & r_{26} & r_{23} & r_{28} & r_{25} & r_{27} \\ r_{61} & r_{64} & r_{62} & r_{66} & r_{63} & r_{68} & r_{65} & r_{67} \\ \hline r_{31} & r_{34} & r_{32} & r_{36} & r_{33} & r_{38} & r_{35} & r_{37} \\ r_{81} & r_{84} & r_{82} & r_{86} & r_{83} & r_{88} & r_{85} & r_{87} \\ \hline r_{51} & r_{54} & r_{52} & r_{56} & r_{53} & r_{58} & r_{55} & r_{57} \\ r_{71} & r_{74} & r_{72} & r_{76} & r_{73} & r_{78} & r_{75} & r_{77} \end{array}\right]$$

Our algorithm distributes the matrix by columns across the nodes; for this example, we assume 4 nodes, giving:

$$\left[\begin{array}{cc|cc|cc|cc} r_{11} & r_{12} & r_{13} & r_{14} & r_{15} & r_{16} & r_{17} & r_{18} \\ r_{21} & r_{22} & r_{23} & r_{24} & r_{25} & r_{26} & r_{27} & r_{28} \\ r_{31} & r_{32} & r_{33} & r_{34} & r_{35} & r_{36} & r_{37} & r_{38} \\ r_{41} & r_{42} & r_{43} & r_{44} & r_{45} & r_{46} & r_{47} & r_{48} \\ r_{51} & r_{52} & r_{53} & r_{54} & r_{55} & r_{56} & r_{57} & r_{58} \\ r_{61} & r_{62} & r_{63} & r_{64} & r_{65} & r_{66} & r_{67} & r_{68} \\ r_{71} & r_{72} & r_{73} & r_{74} & r_{75} & r_{76} & r_{77} & r_{78} \\ r_{81} & r_{82} & r_{83} & r_{84} & r_{85} & r_{86} & r_{87} & r_{88} \end{array}\right]$$

We first rotate within each column, resulting in an array:

$$\left[\begin{array}{cc|cc|cc|cc} r_{11} & r_{12} & r_{13} & r_{14} & r_{15} & r_{16} & r_{17} & r_{18} \\ r_{41} & r_{42} & r_{43} & r_{44} & r_{45} & r_{46} & r_{47} & r_{48} \\ r_{21} & r_{22} & r_{23} & r_{24} & r_{25} & r_{26} & r_{27} & r_{28} \\ r_{61} & r_{62} & r_{63} & r_{64} & r_{65} & r_{66} & r_{67} & r_{68} \\ r_{31} & r_{32} & r_{33} & r_{34} & r_{35} & r_{36} & r_{37} & r_{38} \\ r_{81} & r_{82} & r_{83} & r_{84} & r_{85} & r_{86} & r_{87} & r_{88} \\ r_{51} & r_{52} & r_{53} & r_{54} & r_{55} & r_{56} & r_{57} & r_{58} \\ r_{71} & r_{82} & r_{73} & r_{74} & r_{75} & r_{76} & r_{77} & r_{78} \end{array}\right]$$

Finally, we transmit the columns and each node selectively replaces its columns so that the array becomes:

$$\left[\begin{array}{cc|cc|cc|cc} r_{11} & r_{14} & r_{12} & r_{16} & r_{13} & r_{17} & r_{15} & r_{17} \\ r_{41} & r_{44} & r_{42} & r_{46} & r_{43} & r_{47} & r_{45} & r_{47} \\ r_{21} & r_{24} & r_{22} & r_{26} & r_{23} & r_{27} & r_{25} & r_{27} \\ r_{61} & r_{64} & r_{62} & r_{66} & r_{63} & r_{67} & r_{65} & r_{67} \\ r_{31} & r_{34} & r_{32} & r_{36} & r_{33} & r_{37} & r_{35} & r_{37} \\ r_{81} & r_{84} & r_{82} & r_{86} & r_{83} & r_{87} & r_{85} & r_{87} \\ r_{51} & r_{54} & r_{52} & r_{56} & r_{53} & r_{57} & r_{55} & r_{57} \\ r_{71} & r_{84} & r_{72} & r_{76} & r_{73} & r_{77} & r_{75} & r_{77} \end{array}\right]$$

This matches the result of Brent *et al.* as we needed to show.

Pseudoinverse Step

Theorem 3.3.3 *Our distributed pseudoinverse step, Algorithm 3.7, provides the same results as the centralized pseudoinverse step.*

Proof To prove this, we show that $A_{ij}^{+d} = A_{ij}^{+c}$.

Proof follows similar lines as for Theorem 3.3.1. All mathematical operations are identical for both algorithms; differences arise through distributed storage and communication.

3.3.5 Analysis

We would like to analyze each of the main portions of the distributed calibration algorithm, focusing on requirements for computation, memory, and communication. As we defined it, the original matrix A is $m \times n$ with each node storing a column blocks of width $2p$.

Computation

In examining the computational requirements, we divide operations into: multiply/divide, add/subtract, and exponentiation (such as square or square root). Table 3.1 shows the results for each of the three primary algorithms.

With the SVD, we assume the worst case scenario with the rotation step, assuming that n is greater than or equal to 8. S defines the number of iterations necessary for convergence to a threshold-based stopping condition, which is $\mathrm{O}(n^c)$ where $1 \leq c \leq 2$.

Surprisingly, the SVD requires the least operations, $\mathrm{O}(Sn)$ compared to $\mathrm{O}(n^2 m)$ for the QR and $\mathrm{O}(pnm)$ for the PI combination. This results from the matrix sizes necessary for the other two algorithms and reflects the reasoning behind the QR, which reduces the computational requirements of the SVD.

Algorithm	$\times/\div$	$+/-$	$\boldsymbol{x^y}$
QR Decomposition	$(n^2+n)m$	$\frac{n^2-n}{2}(m-1)^2 + n(m-1)$	1
SVD	$S(17n-5)$	$S(10n+6)$	$S(n+5)$
Pseudoinverse Combination	$2pn^2 + 2pn + 2pmn$	$(4p-1)n^2 - 4p^2 n + 2pmn - 2pm$	0

Table 3.1: Computation Analysis Results

Memory

For the memory requirements, we only examine the matrix needs and ignore temporary variables needed for things such as counters and indices. We make no assumptions on the space available but outline the requirements for each algorithm. Depending on the implementation and whether there is a need for the original A matrix, the QR computation could use the original matrix for storing Q, which reduces its footprint. If the system needs A, requiring a second matrix space, the size of m defines whether the system needs a more complicated memory system to store everything. The SVD does not touch the Q matrix and, therefore, uses the smaller n for defining its space requirements. In most cases, this will allow the entire SVD computation to occur in on-chip memory. The pseudoinverse combination step does require the Q matrix, which may require more complicated memory systems dependent on m.

Algorithm	**Results Storage**	**Temporary Storage**
QR Decomposition	Q=$m \times 2p$	None
	R=$n \times 2p$	
SVD	R=$n \times 2p$	$n \times 2p$
	U=$n \times 2p$	$n \times 2p$
	V=$n \times 2p$	$n \times 2p$
Pseudoinverse Combination	$A^+ = 2p \times m$	$n \times n$

Table 3.2: Memory Analysis Results

Communication

Communication analysis explores the messages necessary between nodes. We do not include overhead involved in the messaging protocol and make no assumptions about the protocol; we only define the number of values transmitted for each computation. If $m >> n$, the QR will dominate communications with its transmissions of O(pm) values. Otherwise, the SVD computation will dominate as each iteration requires O(np) values transmitted.

3.4 Randomization and Dimension Reduction

In defining our algorithms for distributed computation of the pseudoinverse, we made no assumptions about the underlying matrix except that $m >= n$. However, con-

Algorithm	**Messages**
QR Decomposition	$2pm$ Q values
SVD	S messages of: 4 c/s values if $i \neq 1$: $4np$ R, U, and V values if $i = 1$: $2np$ R, U, and V values
Pseudoinverse Combination	$(n - 2p)n$ temporary values, $2pn$ Q values

Table 3.3: Communication Analysis Results

sidering the application of the pseudoinverse to linear regression and similar overdetermined problems, the matrix could be larger than available memory or reasonable communication. In this case, we want to consider additional steps to reduce the size of the matrix. We need to do so in a way that preserves the distributed computation, meaning that the algorithm needs to preserve the data structure while not requiring significant additional communication. This leads us to randomized algorithms where we can obtain consistent results by initializing all nodes to the same random seed and still reduce the training set without incorporating unacceptable errors into the calibration step.

We specifically examine a random algorithm described by Drineas *et al.*[30]. This theoretical approach to speeding least squares approximation introduces a set of random operations to shrink the A matrix and b vector before computing the coefficients (see Algorithm 3.8).

However this is a centralized approach and still more complicated than our microprocessors can compute. We can develop a distributed version of this with each node using the same random seed as Algorithm 3.9 depicts. A node, upon measuring a new data value, determines in an online fashion if the associated matrix row will remain in the scaled-down matrix and then performs the necessary transformations on that row. These transformations, based on the order of the matrix multiplications, operate on columns of A and can occur independently on each node. This provides a randomly scaled matrix distributed in a column-wise fashion across the nodes of the network where each node can maintain the columns within internal processor memory, the form our distributed pseudoinverse algorithm requires.

Algorithm 3.8 Random Sampling

Input: $[A, b]$
Output: x_{opt}

$r = O(dlog(n)log(dlog(n)))/\epsilon$

$t = random(0, 1)$
if $(t \leq r/n)$ **then**
    $S'_{ii} = \sqrt{n/r}$
    $S_{\forall i,jsti \neq j} = 0$
else
    $S' = 0$
end if

$j \leftarrow 0$
for $i = 1 : length(S')$ **do**
    if $(S'_i \neq 0)$ **then**
        $S_j \leftarrow S'_i$
        $j = j + 1$
    end if
end for

if $length(S) > 10r$ **then**
    Abort
end if

$H \leftarrow$ normalized Hadamard transform matrix of size n

$t = random(0, 1)$
if $(t \leq 0.5)$ **then**
    $D_{ii} = 0.5$
else
    $D_{ii} = -0.5$
end if

$\mathcal{H} \leftarrow HD$
$x_{opt} \leftarrow (S\mathcal{H}A)^{+}S\mathcal{H}b$

Algorithm 3.9 Distributed Random Sampling

Output: A vector

Measure a_j
if $random(0,1) \leq r/n$ **then**
    Keep Row
    if $i == j$ **then** ▷ Value for our row
        $S_{ii} = \sqrt{n/r}$
    end if
    if $(random(0,1) \leq 0.5)$ **then** ▷ Scale by D
        if $i == j$ **then**
            $a_j = 0.5a_j$
        end if
    else
        if $i == j$ **then**
            $a_j = -0.5a_j$
        end if
    end if
else
    Discard Row
end if

if Calibration Starts **then**
    $H \leftarrow$ Column j of normalized Hadamard transform matrix of size n

    $H = S_{ii}H$
    $A = HA$
end if

3.5 Pseudoinverse Computation for Special Cases

We can optimize our pseudoinverse calibration computation for two different scenarios: (1) computation of a linear regression model and (2) computation of a matrix where n is small (at the point where maintaining several $n \times n$ matrices is feasible within a single microprocessor).

3.5.1 Linear Regression

If the pseudoinverse is used for calibration of a linear regression model, we can reduce much of the computation and communication by multiplying Q by b before other operations on Q. In this scenario, we compute $x_{opt} = VR^{-1}(QU)^T b$. The addition of the b allows us to modify Algorithm 3.7 to that shown in Algorithm 3.10. Each node stores the same data as described in Section 3.3.3 along with b, a vector of length m. Algorithm 3.7 multiplied our temporary row block by local versions of Q, communicated the complete Q row column, and then finished computation of the row block of A^+. Instead we can first multiply the local row column of Q by the local b_i value, reducing it to a single value. Each node communicates its single value and can store the complete Qb_i vector, which it then uses to multiply by the temporary row block to compute the x_{opt}. At the end of the algorithm, each node has $2p$ values of x_{opt}.

Comparing this to the original pseudoinverse, we see the analysis results shown in Tables 3.4, 3.5, and 3.6. This change reduces computation by $2pm(n-1)$ multiply/divides and $2pn(m-1)-2p(m-2)$ add/subtract operations, memory storage by a factor of m in results storage, and communication by n Q values; the latter being the most important as it uses the most energy and also provides the highest potential for failure.

Algorithm 3.10 Pseudoinverse Combination for Linear Regression

Input: $[U, R, V, Q, b]$
Output: x_{opt}

Node i computes the following:
$V_i = V_i R_i^- 1$

for $y = 1 : n$ **do**
  for $z = 1 : n$ **do**
    $t_{yz} = \sum_{w=1}^{2p} V_{yw} U_{zw}$
  end for
end for
Node i stores row block t_i of size $2p \times n$ and transmits the rest of t
Upon receipt of values s_i corresponding to the same block as t_i, $t_i = t_i + s_i$

$q_i = Q_i^T b_i$
Each node transmits q_i after which every node has a vector of length n
$x_{opt} = t_i q_i$

Algorithm	$\times/\div$	$+/-$	$\boldsymbol{x^y}$
Pseudoinverse Combination	$2pn^2 + 2pn + 2pmn$	$(4p-1)n^2 - 4p^2n + 2pmn - 2pm$	0
Linear Regression	$2pn^2 + 4pn + 2pm$	$(4p-1)n^2 + (2p - 4p^2)n - 4p + 2pm$	0

Table 3.4: Pseudoinverse and Linear Regression: Computation Analysis Results

Algorithm	**Results Storage**	**Temporary Storage**
Pseudoinverse Combination	$A^+ = 2p \times m$	$n \times n$
Linear Regression	$x_{opt} = 2p \times 1$	$n \times n$

Table 3.5: Pseudoinverse and Linear Regression: Memory Analysis Results

Algorithm	**Messages**
Pseudoinverse Combination	$(n-2p)n$ temporary values, $2pn$ Q values
Linear Regression	$(n-2p)n$ temporary values, $2p$ Q values

Table 3.6: Pseudoinverse and Linear Regression: Communication Analysis Results

3.5.2 Centralized SVD

With sufficiently small n, we can consider running the SVD centrally on a node in order to reduce communication compared to distributed. The basic 2×2 computation remains the same with the rotation differing in that no communication occurs. Instead of column computation, we simplify the rotation to a row-based scheme outlined in Algorithm 3.11. This simplifies the computation to the original centralized form developed by Golub [40].

Tables 3.7, 3.8, and 3.9 outline the differences between the two algorithms. As expected, the two trade-off between communication and local computation. Scenarios where a matrix of size $n \times n$ fits in on-chip memory and communication is expensive (mostly likely from an energy standpoint) lend themselves to centralized SVD computation while cheap communication or too large an n require distributed. Depending on the number of nodes and size of n, memory usage could remain almost the same due to the reuse of the temporary data structure for all three matrices.

Algorithm	$\times/\div$	$+/-$	$\boldsymbol{x^y}$
SVD	$S(17n-5)$	$S(10n+6)$	$S(n+5)$
Centralized SVD	$S(20n^3+8n^2-28n)$	$S(8n^3+10n^2-18n)$	$S(6n^2-6n)$

Table 3.7: Centralized SVD: Computation Analysis Results

Algorithm	**Results Storage**	**Temporary Storage**
SVD	R=$n \times 2p$	$n \times 2p$
	U=$n \times 2p$	$n \times 2p$
	V=$n \times 2p$	$n \times 2p$
Centralized SVD	R=$n \times n$	$n \times n$
	U=$n \times n$	
	V=$n \times n$	

Table 3.8: Centralized SVD: Memory Analysis Results

Algorithm	**Messages**
SVD	S messages of: 4 c/s values
	if $i \neq 1$: $4np$ R, U, and V values
	if $i = 1$: $2np$ R, U, and V values
Centralized SVD	None

Table 3.9: Centralized SVD: Communication Analysis Results

Algorithm 3.11 Centralized Singular Value Decomposition

Input: R
Output: $[U, R, V]$

$StopCondition = eps * off(\mathbf{R})$
while $off(\mathbf{R}) > StopCondition$ **do**
  for $i = 1 : n - 1$ **do**
    for $j = i + 1 : n$ **do**
      $w = R_{i,i};\ x = R_{i,j};\ y = R_{j,i};\ z = R_{j,j}$
      Compute subproblem according to Algorithm 3.5

      Normalize results:
      if $abs(d_2) > abs(d_1)$ **then**
        $\tau = c_1;\ c_1 = -s_1;\ s_1 = \tau$
        $\tau = c_2;\ c_2 = -s_2;\ s_2 = \tau$
        $\tau = d_1;\ d_1 = d_2;\ d_2 = \tau$
      end if
      $\kappa = 1$
      if $d_1 < 0$ **then**
        $d_1 = -d_1;\ c_1 = -c_1;\ s_1 = -s_1;\ \kappa = -\kappa$
      end if
      if $d_2 < 0$ **then**
        $d_2 = -d_2;\ \kappa = -\kappa$
      end if

      $\tau = R$
      for $k = 1 : n$ **do**
        $\tau_{k,i} = R_{k,i}c_2 - R_{k,j}s_2$
        $\tau_{k,j} = R_{k,i}s_2 + R_{k,j}c_2$
      end for
      $R = \tau$
      for $k = 1 : n$ **do**
        $R_{i,k} = \tau_{i,k}c_1 - \tau_{j,k}s_1$
        $R_{j,k} = \tau_{i,k}s_1\kappa + \tau_{j,k}c_1\kappa$
      end for
      $\tau = U$
      for $k = 1 : n$ **do**
        $U_{k,i} = \tau_{k,i}c_1 - \tau_{k,j}s_1$
        $U_{k,j} = \tau_{k,i}s_1\kappa + \tau_{k,j}c_1\kappa$
      end for
      $\tau = V$
      for $k = 1 : n$ **do**
        $V_{k,i} = \tau_{k,i}c_2 - \tau_{k,j}s_2$
        $V_{k,j} = \tau_{k,i}s_2 + \tau_{k,j}c_2$
      end for
    end for
  end for
end while

3.5.3 Semi-Distributed Linear Regression

If we compute the SVD centrally, we also should consider a centralized computation of the pseudoinverse combination algorithm as the R, U, and V matrices will only exist on one node. In the case where we calibrate a linear regression model, this results in Algorithm 3.12. We can split the computation into a centralized portion computed by Node 0 and a distributed portion computed by all nodes. Node 0 computed the centralized SVD so has matrices U, V, and R (which is diagonal so only n values). It first computes $VR^{-1}U^T$, resulting in a $n \times n$ matrix. Knowing the final multiplication by b allows us to optimize the computation by multiplying Q^T by b before communicating these values. Each node has p columns of Q and $2p$ values of b; the transpose allows each node to compute Q^Tb (Qb would require storing rows and additional communication to reshuffle Q after Algorithm 3.3). Nodes then communicate their portion to Node 0, which performs the last multiplication step to compute the weighting coefficients. The system completes calibration by communicating these values to those nodes participating in the distributed prediction.

Algorithm 3.12 Semi-Distributed Pseudoinverse Combination for Linear Regression

Input: $[U, R, V, Q, b]$
Output: x_{opt}

Central node computes the following:
$V = VR^{-}1$
$T = VU^T$

Each node i computes the following:
$q_i = Q_i^T b_i$
Each node transmits q_i with the central node storing the result vector of length n

$x_{opt} = Tq_i$

Tables 3.10, 3.11, and 3.12 compare this version to the distributed linear regression from Algorithm 3.10 and the original distributed pseudoinverse combination Algorithm 3.7. Compared to the distributed linear regression form (Algorithm 3.10), this change increases computation by a factor of n, while decreasing memory storage

by a factor of n^2 and communication by a factor of n^2. In cases where the communication uses significant energy and/or the on-chip memory is insufficient, centralizing the pseudoinverse significantly reduces both.

Algorithm	$\times/\div$	$+/-$	x^y
Pseudoinverse Combination	$2pn^2 + 2pn + 2pmn$	$(4p-1)n^2 - 4p^2n + 2pmn - 2pm$	0
Linear Regression	$2pn^2 + 4pn + 2pm$	$(4p-1)n^2 + (2p-4p^2)n - 4p + 2pm$	0
Semi-Distributed	$n^3 + 2n^2 + 2pm$	$n^3 - n + 2p(m-1)$	0

Table 3.10: Semi-Distributed Linear Regression: Computation Analysis Results

Algorithm	**Results Storage**	**Temporary Storage**
Pseudoinverse Combination	$A^+ = 2p \times m$	$n \times n$
Linear Regression	$x_{opt} = 2p \times 1$	$n \times n$
Semi-Distributed	$x_{opt} = n \times 1$	None

Table 3.11: Semi-Distributed Linear Regression: Memory Analysis Results

Algorithm	**Messages**
Pseudoinverse Combination	$(n-2p)n$ temporary values, $2pn$ Q values
Linear Regression	$(n-2p)n$ temporary values, $2p$ Q values
Semi-Distributed	$2p$ Q values

Table 3.12: Semi-Distributed Linear Regression: Communication Analysis Results

3.6 Analysis: Understanding and Intuition

We introduced a number of algorithms and analyzed each in terms of computation, memory, and communication. To better understand this analysis from a practical standpoint, Table 3.13 outlines the approximate values for these categories for a number of array sizes. Table 3.14 shows the results for the overall algorithms, both the pseudoinverse computation of A^+ and the linear regression computation of x_{opt}. Each number for computation, memory, and communication is a single value with no assumptions made about the number of bytes needed to represent the value. For both tables, we use $S = 10$ and $p = 1$. The latter implies that each node stores 2 columns of the matrices, requiring, for example, 5 nodes for the 10 column case, which is a reasonable number for a sensor network.

Algorithm	Matrix Size	Computation	Memory	Communication
Distributed QR	10×10	4836	40	20
	100×10	453036	220	200
	100×20	1906171	240	200
	1000×10	45030036	2020	2000
Centralized QR	10×10	48360	200	0
	100×10	4530360	10100	0
	100×20	38123420	10400	0
	1000×10	450300360	1000100	0
Distributed SVD	10×10	2860	120	240
	100×10	2860	120	240
	100×20	5660	240	440
	1000×10	2860	120	240
Centralized SVD	10×10	298800	400	0
	100×10	298800	400	0
	100×20	2325600	1600	0
	1000×10	298800	400	0
Distributed Pseudoinverse Combination	10×10	860	120	100
	100×10	4280	300	100
	100×20	9760	600	400
	1000×10	38480	2100	100
Centralized Pseudoinverse Combination	10×10	3900	300	0
	100×10	38100	2100	0
	100×20	156400	4400	0
	1000×10	380100	20100	0
Distributed Linear Regression	10×10	556	102	82
	100×10	916	102	82
	100×20	2436	402	362
	1000×10	4516	102	82
Semi-Distributed Linear Regression	10×10	2228	10	2
	100×10	2588	10	2
	100×20	17178	20	2
	1000×10	6188	10	2
Centralized Linear Regression	10×10	4090	310	0
	100×10	40090	2110	0
	100×20	160380	4420	0
	1000×10	400090	20110	0

Table 3.13: Intuition of Analysis Results for Each Algorithm

Algorithm	Matrix Size	Computation	Memory	Communication
Distributed Pseudoinverse	10×10	8556	280	360
	100×10	460176	640	540
	100×20	1921591	1080	1040
	1000×10	45071376	4240	2340
Centralized Pseudoinverse	10×10	351060	900	0
	100×10	4867260	12600	0
	100×20	40605420	16400	0
	1000×10	450979260	1020600	0
Distributed Linear Regression Model	10×10	8252	262	360
	100×10	456812	442	540
	100×20	1914267	882	1040
	1000×10	45037412	2242	2340
Semi-Distributed Linear Regression	10×10	305864	450	22
	100×10	754424	630	202
	100×20	4248949	1860	202
	1000×10	45335024	2430	2002
Centralized Linear Regression	10×10	351250	910	0
	100×10	4869250	12610	0
	100×20	40609400	16420	0
	1000×10	450999250	1020610	0

Table 3.14: Intuition of Analysis Results for Overall Algorithm

In examining these tables, we see a clear trade-off between computation, memory, and communication. Centralizing the algorithm results in increased computation, increased memory, and decreased communication. Computation increases by an order of magnitude when we centralize the algorithms, while memory increases by one to two orders of magnitude depending on the algorithm. This memory increase exceeds the storage capabilities of a sensor node even for a small matrix such as 100×10. Thus, ensuring the algorithm can run on a sensor network requires distributing the algorithm, even though this does have a communication cost.

Additionally, the SVD and semi-distributed linear regression are especially sensitive to increased n values, making it important to keep this number small. However, the overall computation is dominated by the QR, which depends on m. Distributing the QR and semi-distributing the linear regression algorithms have a larger impact due to the m value than distributing the SVD, increasing the importance of ensuring these algorithms function on a sensor network compared to the distributed SVD (in standard cases where n remains small).

3.7 Testing: Simulation and Implementation

With the algorithm outlined and analyzed, we can test it through simulation and implementation. First we use Matlab to generate random data and simulate the algorithm to verify functionality. We next explore two side issues: experimentally verifying the SVD stopping condition and exploring the random sampling approach to shrinking the data set. Finally we implement the algorithm on our sensor network.

3.7.1 Simulation

We use Matlab to verify the functionality of the algorithms. We start by generating random matrices of a variety of sizes where entry values range from -1 to 1. With the A matrix generated, we generate a random vector for x_{opt} and multiply it with A to compute b. This provides all the pieces needed for testing. We first examine the SVD portion of the algorithm as it is the most complicated, followed by the pseudoinverse, and ending with the full linear regression computation.

Distributed SVD

For the SVD, we run 100 iterations for a variety of n values. To verify functionality, we compute the root mean square (RMS) error of the difference between the original A matrix and the A matrix generated by the SVD. We compare these errors for the distributed SVD algorithm (Alg. 3.4) with the centralized SVD algorithm (Alg. 3.11) and the internal Matlab SVD. Table 3.15 shows the results, which demonstrate that Algorithms 3.4 and 3.11 do compute the proper SVD result as all three generate errors within 10 times the numeric accuracy of the computer.

	n						
Algorithm	4	5	6	8	32	64	100
Dist. SVD	1.01e-15	1.32e-15	1.86e-15	2.69e-15	1.28e-14	2.65e-14	4.29e-14
Cent. SVD	1.11e-15	1.32e-15	1.90e-15	2.60e-15	1.26e-14	2.63e-14	4.21e-14
Matlab	8.21e-16	9.22e-16	1.26e-15	1.64e-15	4.25e-15	5.81e-15	7.49e-15

Table 3.15: SVD Test Results

Distributed Pseudoinverse

We implement Algorithm 3.2 in Matlab and test with 100 iterations over a variety of $m \times n$ values. As we do not have an original matrix to compare to, we use the internal Matlab pseudoinverse and compute the RMS error between that and the A^+ computed by the algorithm. Table 3.16 displays the results. Again the results are within an order of magnitude of the numeric accuracy of the computer.

	$\boldsymbol{m \times n}$					
Algorithm	10×4	10×6	11×6	100×4	100×20	1000×20
Our Pseudoinverse	5.12e-16	8.49e-16	6.72e-16	9.97e-17	1.44e-16	3.67e-17

Table 3.16: Pseudoinverse Test Results

Distributed Multiple Linear Regression

In some instances we use the distributed pseudoinverse for multiple linear regression. We implement this case of Algorithm 3.2 where Algorithm 3.10 replaces Algorithm 3.7. To evaluate this functionality, we randomly generate a coefficient vector, x_{opt}, in addition to our A matrix; multiplying the two provides our b vector. We compute the RMS error between the original x_{opt} and our computation as well as x_{opt} and Matlab's computation, giving us the results shown in Table 3.17.

	$\boldsymbol{m \times n}$					
Algorithm	10×4	10×6	11×6	100×4	100×20	1000×20
Our Linear Regression	2.36e-16	4.72e-16	4.65e-16	2.69e-16	8.35e-16	9.92e-16
Matlab	3.52e-16	5.93e-16	5.75e-16	4.86e-16	1.01e-15	1.15e-15

Table 3.17: Linear Regression Test Results

3.7.2 SVD Stopping Condition

In this section we outline the experimental bounds on the SVD stopping condition. We use our randomly generated matrices from before and now monitor the average number of iterations needed to reach the condition that $off(R) > eps * off(R)$ or $1 > eps$, where eps is the numerical accuracy of the computer's float representation. Table 3.18 shows the results.

	n							
Algorithm	4	5	6	8	16	32	64	100
Dist. SVD	10.9	22.1	23.7	37.6	97.2	231.5	533.4	892.5

Table 3.18: SVD Stopping Condition Test Results

These numbers clearly demonstrate the need for reducing the size of the matrix upon which we compute the SVD, strengthening our argument for first computing the QR Decomposition. Since the QR generates a matrix of size $n \times n$ to input into the SVD and, in the majority of applications, n will be small (on the order of 5-20), the SVD can complete in a reasonable time on a sensor network.

3.7.3 Random Sampling

We now explore the random scaling algorithms, Algorithms 3.8 and 3.9, introduced in Section 3.4 and see if they provide similar results to the original results despite the smaller matrix. To determine the necessity of all the transforms involved in the random scaling, we also compare the algorithm to simpler methods: randomly choosing with no additional transformations and choosing every nth row. To test this, we cannot use a random data set as random sampling relies on some connections between the data; additionally, analyzing the effects of randomly sampling random data most likely will provide a bad test case. Instead, we use an existing data set from a sensor network in Okalahoma. This data set consists of 7 years of data containing river flow, rainfall, and air temperature from the Blue River [68, 93] (we also utilize this data set in Chapter 4). Table 3.19 shows our results comparing these values. In this test, we fixed all model parameters except for the amount of data used by each method.

Our metrics are the modified correlation coefficient, false positive count, and false negative count (from Section 4.1). Examining the results, for the modified correlation coefficient and false negative count, the random algorithms all work about equally well, and as well as using all the data. For the random algorithms compared to using all the data, the difference appears in the false positives with the random algorithms generating far more false positives than using all the data. This means we predict

Method	**Modified Correlation Coefficient**			**False Positives**			**False Negatives**		
Fraction of Data Used	0.2	0.6	1	0.2	0.6	1	0.2	0.6	1
All Data			0.484			58			13
Random Sampling	0.486	0.484	0.484	67.8	58	58	12.8	13	13
Random Choice	0.471	0.485	0.484	83.5	61	58	12.5	13	13
Choose every nth	0.486	0.485	0.484	99	60	58	12	12	13

Table 3.19: Comparison of Random Sampling Methods

events that do not occur, but do not miss any more events than using all the data. If we need to employ these algorithms to reduce the matrix size, we now know that there will be an increase in false predictions of events, which we must take into account in formulating our response to these predictions.

For understanding the similarities between the three random methods, probabilistic analysis of the utility of points chosen sheds some light on the problem. We consider the utility of a given row where we base utility on the likelihood of a point indicating a flood. These points best capture the events of interest that lead to coefficients that allow the prediction to match the observed record and provide a reasonable modified correlation coefficient value. Without these points, the model could only capture the very low level values, missing the peaks, and returning a low correlation coefficient.

To show this probabilistically, we define the following variables:

F = number of points indicating flood

T = total number of data points

c = fraction of points used in any sampling algorithms

For the Drineas algorithm,

$$r = c * T$$

$$P(choose\ point \cap point\ indicates\ flood) = P(choose\ point)P(point\ indicates\ flood)$$

$$= (r/T) * (F/T)$$

$$= c * (F/T)$$

For randomly choosing based on c or deterministically choosing every $1/c$ row,

$$P(choose\ point \cap point\ indicates\ flood) = P(choose\ point)P(point\ indicates\ flood)$$

$$= c * (F/T)$$

As the probabilities are identical, all methods are equally likely to choose valuable rows and, probabilistically, will perform equally well.

3.7.4 Sensor Network Test

We then implemented these algorithms on our sensor network, described in Section 4.2. While in this chapter we described many algorithmic variations, we chose to implement the semi-distributed linear regression algorithm outlined in Section 3.5.3. We did this because our tests and applications outlined in this thesis all use the linear regression form for prediction and have a small n value, suggesting this is the best implementation for our sensor network. We suggest potential future applications where the other variations may make more sense in Chapter 7.

To start, each node knows what columns of the matrix it maintains, granting the node with column 0 master control of the operation. We will denote each node as x_i where i defines the order each node has in storing the columns of the matrix (i.e. x_0 stores the first set, x_1 stores the second, etc...). While for this test we fixed these values, the nodes could dynamically decide this placement. Each node then runs a state machine-like control loop defining where it is in the algorithm as demonstrated in Figure 3.1. It begins with all nodes in the waiting state. To determine when to start, x_0 performs a check of the defined calibration window and proceeds to the next state if that event window has occurred. Node x_0 then transmits a command to load the data for the start of the algorithm causing all nodes to transition to the QR algorithm state, where each operates as outlined in Algorithm 3.3. Upon completion of the QR algorithm, x_0 requests the R and the $Q_i^T b_i$ values from the other nodes and commences the SVD state. The other nodes return to waiting state at this point. x_0 then completes the SVD state and pseudoinverse state, concluding

with a transmission of the new coefficients for the prediction algorithm to the other nodes. All nodes save their portion of the coefficients and x_0 transitions to the waiting state, completing the algorithm. Once the next calibration window occurs, due to any number of policies such as an increase in the prediction error or a scheduled cyclic calibration event, the algorithm and state machine begin again.

In addition to implementing these algorithms, we ensured fault detection, correction, and tolerance in this implementation. The network will handle any of the following issues:

- All zeros in the first column data
- Q columns received out of order
- Q column not received
- R column not received
- Coefficients all zero

If the data is all zero in the first column, the algorithm will not commence and will retry later. For errors where columns are not received or arrive out of order, the algorithm will fail gracefully and retry later. If at the end of the algorithm, the coefficients received are all zeros, the nodes will not load these values, but maintain the old values and wait for a retry of the algorithm. Currently there is no data replication to ensure successful completion of the algorithm upon complete failure of a node; we leave this for future work, but believe there are many simple policies to fix this.

We implemented this on three nodes, fixing the input data to test functionality and repeatability. We set the data matrix at 10×6, with the first two columns containing actual measurements from the internal temperature sensors and the other four chosen to be dissimilar from that data and testing the issues arising from having all the same data in a column. Each node stored 2 columns and ran the algorithm every 5 minutes for 7 hours and 24 minutes. Over this time, the nodes attempted and successfully completed the algorithm 75 times, correctly computing the coefficients each time. There were no errors. We also ran directed tests to verify the fault tolerance and each time saw clean recovery of the system. Overall these tests successfully proved the operation of these algorithms on a sensor network.

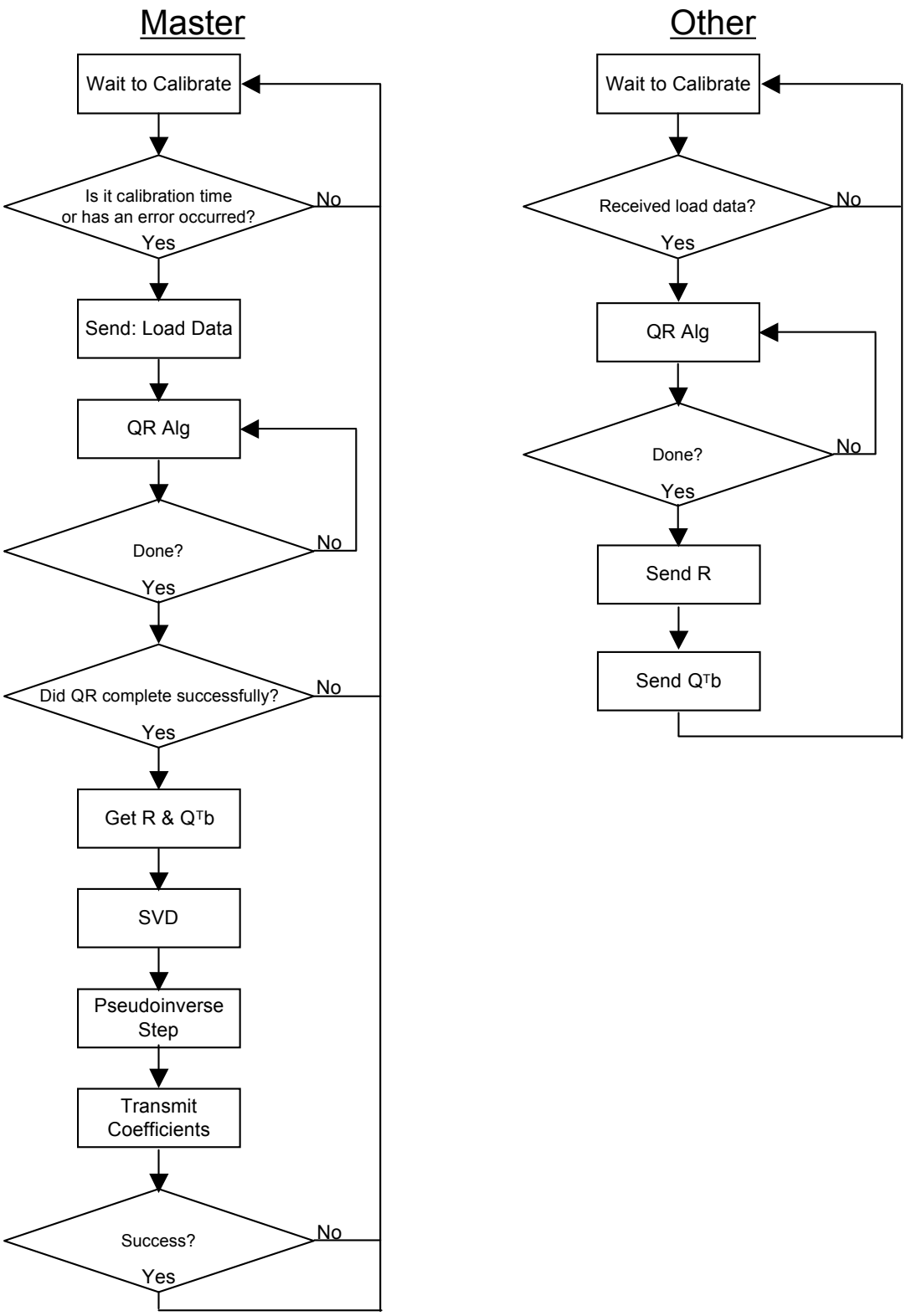

Figure 3.1: Control Flow for Master and Other Implementations

3.8 Conclusions

In this chapter, we introduce distributed algorithms to compute: QR decomposition, SVD, matrix pseudoinverse, and multiple linear regression. We also suggest optimizations capitalizing on the structure of the matrix to utilize a centralized SVD and perform a semi-distributed linear regression. For all of these algorithms, we analyze the computation, memory, and communication costs, providing alternatives for different problem specifications. We also suggest methods for reducing the size of the matrix in cases where it cannot be maintained within the processor. Finally, we verify all the algorithms through simulation and implement them on a sensor network, again verifying the functionality of the algorithms.

Chapter 4

River Flood Prediction

In the previous chapter, we discussed the algorithms needed for building a toolbox to enable a smarter sensor networks. We now utilize those algorithms to predict river flooding and additionally consider the sensor network architecture issues associated with such prediction problems (portions of this work were previously published in [6]). We focus on developing and deploying systems to monitor large environmental events, and to deal with system constraints required for real-world use of these networks.

Predictive environmental sensor networks require addressing several complicated design requirements. The network must cope with element exposure, node failures, limited power, and prolonged use. When the event damages the environment, such as the case with floods or hurricanes, this further complicates the requirements. This system must withstand the event, which usually poses a hazard to network survival especially to those nodes directly measuring the event. Additionally, the system must operate throughout long disaster-free periods, measure a variety of variables contributing to the disaster, thereby requiring heterogeneous sensor support, and communicate over the large geographical regions in which these events occur. Our specific application is river flooding with a main deployment target for the system of rural and developing regions. With this application in mind, we can further define the system requirements. The system must withstand river flooding and the severe storms causing the floods, monitor and communicate over a 10000 km^2 river basin, predict flooding autonomously, and limit costs allowing feasible implementation of the system in a developing country.

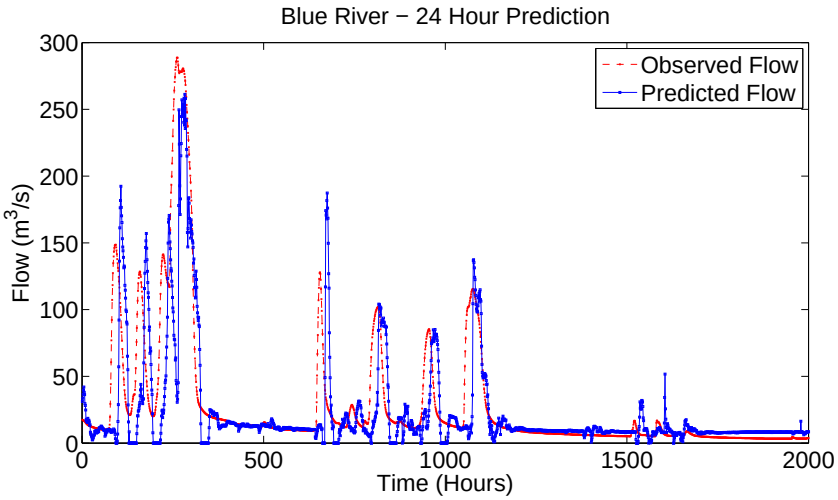

Figure 4.1: Blue River 24 Hour Prediction Results on Verification Data Set

We built a sensor network for flood prediction that consists of 9 nodes. Once the system meets these fundamental sensing design requirements, it then needs to actually predict the event of interest. Most algorithms for this do not conform easily to a sensor network, instead focusing on a centralized computing system with significant processing power and complex system models. This sort of computational power does not exist everywhere we might want to install such a prediction network, especially rural and developing countries, nor do we want to install such computing power. We instead would like to parsimoniously use the computing power on the sensor network to perform this prediction by adaptively sampling data from the network. Computing models on a network suggests executing a simplified form of the underlying physical model and developing distributed implementations. Key to this process involves eventually connecting the model to the data collection such that the data drives when and what is measured, how often the model computes predictions, and when the system communicates predictions; project goals not fully achieved in this thesis.

In this chapter we present a sensor network architecture and instantiation for developing regions, a statistical modeling algorithm for river flood prediction, and evaluations of both. The flood prediction algorithm is based on our regression model of Chapter 3. It performs significantly better than current hydrology research ver-

sions at 1 hour predictions and nearly as good as these same research versions when our model predicts 24 hours and those models predict 1 hour. For prototyping and validation purposes, we tested this model using 7 years of data from the Blue River in Oklahoma. We split the 7 years into 1 year of training data and 6 non-intersecting years of verification data. Figure 4.1 demonstrates our modeling results for a 24 hour prediction on a portion of the verification data set, showing how our predicted peaks coincide with the observed reality.

An instantiation of the sensor network was installed on the Charles River at Dover, Massachusetts in 3 different deployments and gathered 5 weeks of data each time. This data was later run through our model. Chosen primarily for practicality reasons and speed of prototyping, the Charles encompasses a basin of 1000 km^2, only one order of magnitude less than our proposed basin, which is important as it shares time scale characteristics regarding hydrological rates of change with our proposed basin to which even smaller basins do not compare. It also provides support from the United States Geological Survey (USGS) and verification of our measurements through their sensors. A simplified version of the 9-node sensor network was also deployed on the Aguán River basin in northern Honduras, which we use as our test basin as this region provides a representative case of a developing region with serious flood problems (see Figure 4.2). This installation was used to test the sensing, networking, deployment, and maintenance issues in rural Honduras. We then had a followup deployment running the flood prediction algorithm in-situ in Honduras during March 2009 and Dover during November 2009.

The rest of the chapter is organized as follows. Section 4.1 describes the prediction algorithm. Section 4.2 discusses issues related to designing and building a sensor network capable of running the flood prediction algorithm from Section 4.1 in-situ. Section 4.3 discusses the field installation and experimental results.

Figure 4.2: Aftermath of Hurricane Mitch in 1998 in Northern Honduras

4.1 Prediction Model

In this section we describe a model and an efficient algorithm for flood prediction based on Algorithms 3.1 and 3.2 that uses data from the nodes of a spatially distributed sensor network. This approach is computationally leaner than conventional approaches to flood modeling and prediction, utilizing real-time data from multiple sensor nodes.

Rainfall driven floods[1] are the most common seasonal events. They occur when the soil no longer has the capacity to absorb rainfall. To predict flooding, a model requires knowing how much rain falls and what the soil's time-dependent response to the rainfall will be.

Current physically-based models compute the rainfall-induced above and below ground flows of water into the river, and subsequent stream flow using numerical implementations of the equations governing transport through the soil and the river channels [77, 85] (see Figure 2.1 for an example [20]). Modeling these processes using physics creates a challenge from a simulation point of view. The model requires details of the topography, soil composition, and land cover, along with meteorological conditions and hydrometeorological quantities such as soil moisture [54].

In the development of these rainfall-runoff models, ongoing work covers a range of models from lumped to spatially-distributed variations [77, 85]. Although popular in academic research, the need for calibrating spatially-distributed models to individ-

[1]Storm surges are not considered in this work.

ual basins, model sensitivity to basin conditions, and the tremendous computational burden involved in running them makes wide-spread application complicated and, in resource-strapped underdeveloped areas, nearly impossible.

In contrast, statistics gleaned from the observed record can lead to the development of low-dimensional distributed models, which are local in the sense of being valid for a given site. Such models intrinsically self-calibrate because the evolving record of observations allows them to adapt to the latest conditions. This creates portability from one locality to the next, from one season to the next, and from one climate regime to the next. Statistical models can yield low computational complexity, making them well suited for on-site and real-time implementations. Several of such statistical models running on different portions of the basin can collaborate in a distributed inference network to estimate flow at unobserved portions of the basin. Thus statistical models can also yield spatially-extended estimates. These benefits cut across the traditional justification for physically-based models and motivate their use in our work.

A growing body of evidence indicates that statistical models are useful in earth systems. This is true of flood prediction and, although the evidence [15, 16, 81, 86, 87, 88] here is sparse, we can see mature applications in other areas. In particular, statistical models have proven among the best in forecasting hurricane intensity (which presents similar challenges to flood forecasting) [29] and are used for guidance in operational cyclone forecasting [60].

The simplest set of statistical models is that of linear regressions [14], appearing in various forms for hydrological modeling [15]. Within this category, multiple linear regression models assume that a linear equation can describe the system behavior, weighting the past N observations of all relevant input variables taken at time t to produce a prediction of the output variable at time $t + T_L$. We can also consider adding past predictions and/or past prediction errors as inputs within this model. To determine the weighting factors, some amount of data is designated as the training set for the model, defined here as the data seen in time T_T (an application-defined parameter), and a simple inversion-multiply operation provides the coefficients from

Algorithm 4.1 Flood Prediction Algorithm

ϕ : past flow
θ : air temperature
ρ : rainfall
N : # past flow values used
Q : # rainfall values used
P : # air temperature values used
Y : predicted flow
e : prediction error
T_T : training time window
T_L : prediction lead time
T_R : recalibration time window

$T_{TL} = T_T - T_L$;
▷ Compute initial coefficients and prediction
$\phi_N \leftarrow [\phi(1 : T_{TL} - N), ..., \phi(1 + N : T_{TL})]$
$\theta_P \leftarrow [\theta(1 : T_{TL} - P), ..., \theta(1 + P : T_{TL})]$
$\rho_Q \leftarrow [\rho(1 : T_{TL} - Q), ..., \rho(1 + Q : T_{TL})]$
$X \leftarrow [\phi_N, \theta_P, \rho_Q]$
$C = ((X * X^T)^{-1} * X^T) * Y(1 + T_L : T_T)$
$Y(1 + T_L : T_T) = X * C$
▷ Recompute using prediction error
$e = Y(1 + T_L : T_T) - \phi(1 : T_T - T_L)$
$X \leftarrow [\phi_N, e, \theta_P, \rho_Q]$
$C = ((X * X^T)^{-1} * X^T) * Y(1 + T_L : T_T)$
$Y(1 + T_L : T_T) = X * C$

for $t = T_T + 1$ to ... **do** ▷ Forecast
    if $(t\%T_R) == 0$ **then**
        ▷ Recalibrate coefficients
        $e = Y(t - T_T : t) - \phi(t - T_T - T_L : t - T_L)$
        $\phi_N \leftarrow [\phi(t - T_{TL} : t - N), ..., \phi(t - T_{TL} + N : t)]$
        $\theta_P \leftarrow [\theta(t - T_{TL} : t - P), ..., \theta(t - T_{TL} + P : t)]$
        $\rho_Q \leftarrow [\rho(t - T_{TL} : t - Q), ..., \rho(t - T_{TL} + Q : t)]$
        $X \leftarrow [\phi_N, e, \theta_P, \rho_Q]$
        $C = ((X * X^T)^{-1} * X^T) * Y(t - T_T : t)$
    end if
    ▷ Compute Forecast
    $e = Y(t) - \phi(t - T_L)$
    $\phi_N \leftarrow [\phi(t - N), ..., \phi(t)]$
    $\theta_P \leftarrow [\theta(t - P), ..., \theta(t)]$
    $\rho_Q \leftarrow [\rho(t - Q), ..., \rho(t)]$
    $X \leftarrow [\phi_N, e, \theta_P, \rho_Q]$
    $Y(t + T_L) = X * C$
end for

this data, which is the prediction model until recalibration occurs, defined as a time window of length T_R. In case the data provided contains local perturbations limiting the effectiveness of the coefficients, we can smooth the data using a low-pass filter.

We developed a model using this technique, with inputs of past flow (ϕ), air temperature (θ), and rainfall (ρ), defining their orders as N, P, and Q respectively, and a single output, predicted river flow (Y). Algorithm 4.1 outlines this model. Lines 15 through 18 setup the calibration matrix. For each variable, we can use any number of past values to define the linear prediction; in calibration each past value becomes a column within the matrix. For example, in using the past river flow observations, we could use only that occurring at t, or we could add $t-1$, $t-2$, etc. as we design the actual implementation of the model. In our algorithm description, we define the number of past values used as N, Q, and P for the node's past river flow (ϕ), air temperature (θ), and rainfall (ρ), respectively. We load the matrix, X, with this data set over the calibration window defined. Line 19 performs the calibration step and generates a coefficient vector, C. To include the prediction error in the model, we then predict over the training window we just used by multiplying our calibration matrix by our coefficients (Line 21). We subtract the observed record from this prediction, thus generating our error, and include this in our new calibration matrix of Line 23. Recomputing the coefficients based on the new calibration matrix occurs in Line 24 and, with these coefficients, we begin predicting the future river flow in Line 27. This loop continues forever, recalibrating after a full recalibration time window passes (Line 29), computing the prediction error based on the latest observation (Line 38), and predicting future river flow (Line 43).

In using this model to predict river flooding, we use a thresholding approach. Based on existing data or community knowledge (if no data exists), we define two important river levels for each location relating its river level to local or downstream flooding. First, we determine an alert level above which conditions most likely lead to flooding and small issues may occur; above this value government organizations should pay attention. Second, we determine a flood level above which regions are flooded and action must occur. By providing a complete time series in addition to

indicator levels, we allow for human self-checking of levels, foreknowledge of other interesting events such as the receding of water levels, and the opportunity for tiered predictions where downstream locations use upstream river predictions in addition to the measured data.

This model, as we implemented it, self-calibrates, can use very little training data (on the order of weeks), performs a very simple set of operations, and requires storing only the amount of data necessary for training. Considering the complexity of the current model as explained in Section 2.2 and the goal of computation on the sensor network, the use of such simple models is easily motivated. We now consider the requirements for implementing Algorithm 4.1 on a sensor network.

4.2 Sensor Network Architecture

Algorithm 4.1 requires rainfall, air temperature, and water flow data collected and transmitted in real-time over the entire river basin area multiple times per hour. Since the flow of a river may change significantly over a period of several minutes, this suggests a sampling rate on the order of minutes. In order to support distributed, robust, real-time data collection, transmission, and processing for large geographic regions corresponding to real river basins, we further define the following system requirements:

- Monitor events over large geographic regions of approximately 10000 km^2
- Provide real-time communication of measurements covering a wide variety of variables contributing to the event occurrence
- Survive (on the order of years) long-term element exposure, the potentially devastating event of interest, and minimal maintenance
- Recover from node losses
- Minimize costs
- Predict the event of interest using a distributed model driven by data collected

The distance requirement, the inability to populate the entire area with sensors, and cost limitations suggest a two-tiered approach with a small number of long-range nodes surrounded by a cluster of sensing nodes[2]. The long-range nodes communicate over long distances on the order of 25 km using 144 MHz radios, have more power, and, therefore, can provide more online time for computation (although utilizing the same processing system). The sensing nodes operate at 900 MHz, cover a more dense area, use low power, and have a smaller physical footprint. In addition to the field nodes, we integrate office and community nodes with the 144 MHz network. These nodes provide user interfaces to the system. We have developed a system capable of running a distributed version of Algorithm 4.1 in-situ. We want to avoid centralized prediction for two reasons: (1) to avoid a single point of failure and (2) to enable data-driven parsimonious calibration on the lower power 900 MHz network. Figure 4.3 shows a concept overview for this collection of four node types and how they communicate.

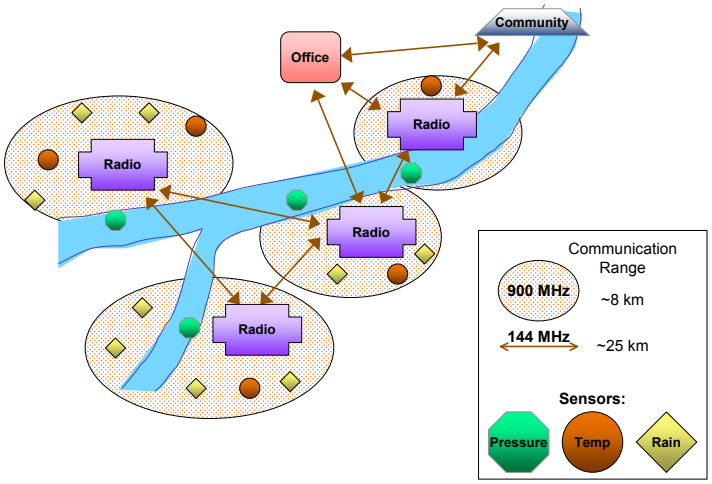

Figure 4.3: Idealized Sensor Network Consisting of Two Communication Tiers and Four Node Types; Communication Ranges Not to Scale

All four node types have a common base board and architecture that we then expand by daughter-boards as appropriate. The rest of this section describes our design and implementation for these system components. Figure 4.4 gives a more detailed overview.

[2]With these three requirements, it is impractical to implement a homogeneous scattered sensor network.

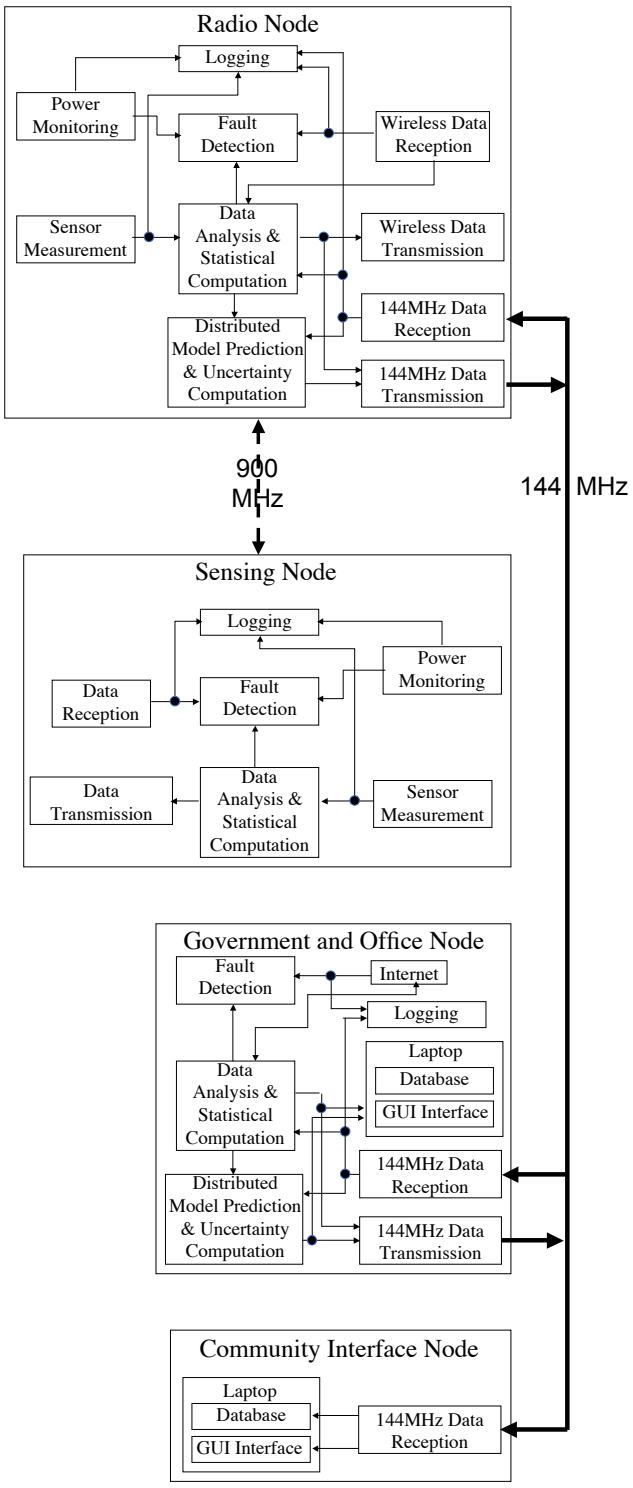

Figure 4.4: Generic Predictive Environmental Sensor Network Architecture Consisting of Sensing, Radio, Government Office, and Community Interface Nodes

4.2.1 Base System

All nodes begin with the same base electronics designed to provide for a variety of options. An ARM7TDMI-S microcontroller core, specifically the LPC2148 from NXP, provides the necessary computation power for the board [71]. This microcontroller has a relatively large amount of on-chip RAM (40K), flash (512K), and input-output pins (along with other features) as well as balancing the trade-off between power usage and processing. Others we considered tended towards the extremes of this spectrum with the ATmega128 on the low end and the Blackfin ADSP-BF533 on the high end.

The LPC2148 has a limited number of physical serial ports, which we extend to 8 by adding a Xilinx CoolRunner-II CPLD to the system and configure it as a serial router. The base board sends all free pins to the daughter-board connectors allowing for a variety of operations (e.g. digital input/output and analog conversion) and potential multiplexing of each I/O on specialized boards. A mini-SD Card and 32 KB FRAM (Ferroelectric Random Access Memory) supply data and configuration storage.

Finally, a charging circuit on the board allows photovoltaic charging of lithium-polymer batteries, which power the system at 3.7 V, along with measurement of charge current.

In addition to the base hardware constructed, the system runs a custom base software package developed in C using the WinARM libraries. This package consists of: serial libraries which hide the underlying CPLD serial router, a custom EFAT file system for SD-Card logging, sensor access libraries, power regulation, and a scheduler system based on the real-time clock and internal timers. Additionally, there is a bootloader program to load new program code into the system. The bootloader reads the program file from the SD card allowing very easy reconfiguration. We update the program by swapping SD cards or by uploading a file via a serial or radio link. If the board does not properly boot, the bootloader has a failsafe system, automatically loading a backup program.

4.2.2 Communication

The two-tier communication structure uses 900 MHz and 144 MHz systems. As the 900 MHz system is used by almost all nodes, the base system provides that communication using a AC4790 900 MHz wireless module [1]. In choosing a radio, the key requirements were an easy usage model and a range of over 5 km. The range requirement eliminates lower power radios such as the Nordic NRF905, Bluetooth, and TI CC2420. The Aerocomm AC4790 radio claims a 20 mile communication range and initial testing showed better performance than 50 mW Zigbee radios available at time of choice. The AC4790 wireless module provides RF and interface protocols, handling issues such as retries, error detection, and peer-to-peer communication. It operates at a fixed data rate, optimally 76.5 kbits/sec but dropping to approximately 7.2 kbits/sec once the internal Aerocomm messaging overhead is considered (duplex mode and 4 transmissions of every message cause the rate to degrade). For the software, much of the transmission protocol is provided by Aerocomm, but we did implement several wrapper libraries for the AC4790 in order to interface with it and add our own simple packet structure.

A small subset of the nodes require 144 MHz communication. This frequency usually provides voice communication, so we designed a modem to allow data communication within this frequency over Kenwood TM271A VHF radios. The modem uses a MX614 Bell 202 compatible integrated circuit to convert 1200 baud serial signals to FSK modulated signals for radio transmission. This allows for data transmission across a cheap, long-range communication method without the recurring costs of a satellite or mobile telephone system. For the radio, we added several libraries to the existing base software structure, most specifically wrapper functions to use the modem at a higher level than raw serial data. We developed our own very simple packet structure and communication protocol, defining addresses based on the pre-existing unique base board identifiers and setting a fixed number of retries.

To power the radio, the system needs 12 V instead of the base board requirement of 3.7 V so we use a daughter-board to power the base board for these nodes. This

allows us to completely power-down the radio when not transmitting, maximizing the lifetime of these nodes. The 12 V power requirement of the radio means system cannot use the same lithium-polymer batteries as the sensing nodes so this system uses lead-acid batteries along with 6 W photovoltaic panels. Additionally, to ensure radio communication over the 25 km range, these systems ideally need antennas located at least 5 m high in the air, requiring antenna towers for the system with added benefits of ensuring proper sunlight for the photovoltaic panels and theft protection for the system.

4.2.3 Sensing Nodes

Sensing nodes measure the variables needed to detect and predict the event of interest. In addition to the measurements, the nodes log the raw data, compute data statistics over each hour and inter-transmission time period, and analyze data for indications of potential sensor failures. These nodes regularly transmit via 900 MHz with all nodes in immediate range, creating mini-networks of sensors within the greater system (the combination of few nodes and large areas ensures each node only joins one mini-network). The two operations of transmitting and measuring occur independently, enabling easy modification of the time windows for both the transmission time and the measuring time. By regularly transmitting, nodes provide monitoring of each other through examining the data for errors and noticing the failure of any node not transmitting within an appropriate window. Repeated measurements of odd values such as the maximum possible value of the sensor or rapid rates of change trigger a warning that the sensor may not function anymore, which the node can then transmit via the 900 MHz network to other nodes nearby.

In addition to the standard base board hardware features, this node requires hardware supporting multiple sensors and multiple sensor types. Our nodes accomplish this through a daughter-board attached to the base system that expands the available I/Os through an I2C integrated circuit and creates several ports for sensors ranging from resistive to interrupt to voltage. In case the sensor requires a more complicated interface, we include RS485 and RS232 circuits for external communication to sensors.

We now consider the specific sensors necessary for the application of river flooding: rainfall, air temperature, and water pressure. Other measurements could aid the prediction of river flooding; however, we chose only these three sensor types because of the ease of finding them, connecting to them, and installing them. Additionally, these sensors tend to be inexpensive. So far our modeling work supports using only these three; should this change, our focus on design generality allows for the easy addition of other sensors both from a hardware and a software standpoint.

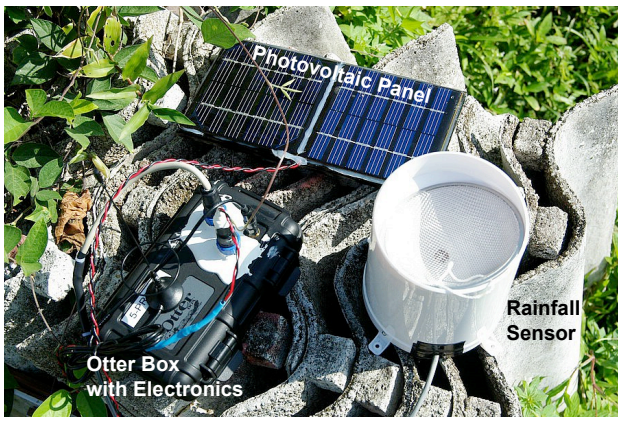

Figure 4.5: Rainfall Sensor Node Consisting of Electronics, Sensor, and Photovoltaic Board

Rainfall sensors measure using reed magnetic switches, which cause an interrupt after every 1 mm of rainfall. Temperature sensors measure resistively, modifying an ADC level, which translates into a temperature after calibration. We placed the electronics within Otter boxes to ensure protection from the elements and added Bulgin connectors for the sensor, antenna, and photovoltaic boards (see Figure 4.5).

Measuring water pressure allows us to compute the water level. While our simulation work described in Section 4.1 uses river flow since that is the data available from the USGS, measuring flow requires several sensors to get a cross-sectional profile of the river in order to understand flow at a single location on the river. Level, however, requires only one measurement to understand the state of the river yet relates to flow through easily defined and understood curves (the USGS actually measures level as well and performs this conversion prior to posting the data online). Therefore we use

the two values interchangeably. To perform the level measurement requires a special underwater installation. In order to maintain solar power and wireless communication, we developed an external pressure sensor box (see Figure 4.6) to communicate via RS485 with the sensing node. Our pressure board consists of another LPC2148 microcontroller, RS485 interface, and instrumentation amplifier. The LPC2148 is much more powerful than necessary, but allows us to maintain a consistent software system. We complete the box by attaching a Honeywell 24PCDFA6A pressure sensor and output the RS485 lines along with power and ground through a Seacon underwater connector. Honeywell's pressure sensor measures 13.8-206.8 MPa of water pressure directly instead of the more typical air pressure, allowing us to bypass the use of extensive tubing to ensure no water touches the sensor.

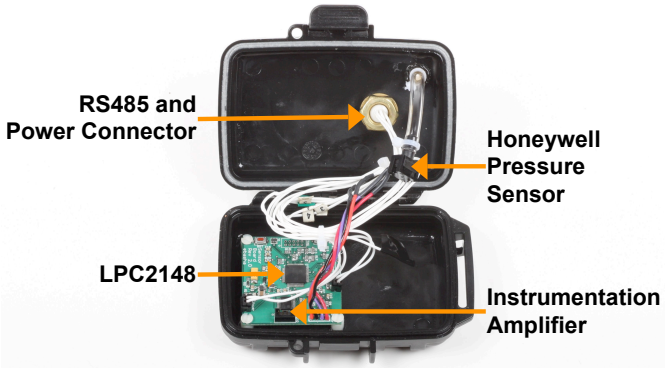

Figure 4.6: Pressure Sensor Box to Communicate with Sensor Node

4.2.4 Radio Nodes

Radio nodes connect the mini-networks of sensors, providing the communication backbone of the system (see Figure 4.7). These nodes will also provide the initiating control for the distributed computation of the prediction. As data arrives from nearby sensors and other radio nodes, the node maintains a record of all values, computes some data statistics, and examines the data correctness. It initiates calibration, computes the final centralized steps of the calibration process, and aggregates the prediction components to generate a local level prediction.

On the communication side, nodes communicate both via the 900 MHz network and to each other via the 144 MHz network. The 144 MHz modem and the power

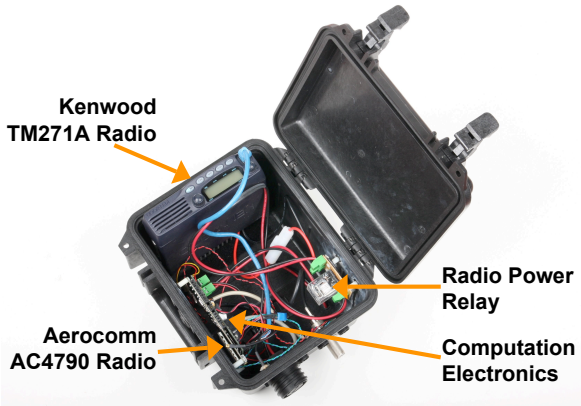

Figure 4.7: Radio Node

switching circuitry for radio control require a different daughter-board from the sensing nodes although we also include the various sensing ports and RS485 communication seen on the other daughter-board.

The communication range requirement drives the separation of these nodes from the sensing nodes with the additional focus of the computation control due to the extra power available from the radios and in order to enable data-driven model calibration within the lower power 900 MHz network. Currently the node uses the same microprocessor as the sensing nodes for prototyping purposes. Should we ever discover a need for more computational power, we could easily add an additional microprocessor to the daughter-board or even a GumStix.

4.2.5 Government Office Interface Nodes

These nodes provide a user interface to the network. We extend our Java development user interface to provide a panel summarizing the information received in a format appropriate for our target audience. This interface focuses on the government and relief agencies who will maintain the system, providing data and predictions regarding the event of interest along with detailed information to monitor the system and display those nodes no longer functioning.

The office nodes communicate via 144 MHz with the radio nodes to provide any external requests for data and receive all of the existing network data. In addition to providing information to the office, receiving all the data will allow the office

nodes to predict for the entire region using a centralized algorithm as a redundancy mechanism to the local distributed predictions (assuming that the computational capability exists, which most likely will not hold true). In our experience, these offices usually exist in the same floodplain as the rest of the system, so we cannot ensure continual operation of the office or the node due to power issues (such as the power grid fails or the batteries are co-opted for a different task) or structural issues (such as the ceiling falls or the building floods). Because of these uncertainties, we cannot rely on these nodes to provide an alert via a centralized algorithm; one of our arguments for distributed solutions.

Additionally, with the possibility of internet access in an office, these nodes could provide external verification for data through online information, using satellite and other remote data available to verify the computation results, checking for errors such as a flood prediction when no rain has fallen. This ideal case has yet to arise in any our deployments, but we look forward to such a possibility and remain prepared to take advantage.

4.2.6 Community Interface Nodes

An effective use of this system requires an intuitive community interface. We have not developed this system component as user-interface issues are not our focus. However, our design requirements for these nodes can be summarized as follows. These nodes provide an interface to the communities interested in the detection and prediction of the events. They utilize the same hardware and base software as the government nodes, but will provide a simpler user interface. The interface will display the known state of the geographic area, event predictions, event detections, and post-event monitoring. To avoid confusion, the interface will not supply detailed information regarding the network, such as node status or the data underlying the computations. Based on the location of the communities within the network, these nodes may also double as any of the other node types.

4.3 Testing: Simulation and Implementation

We designed and performed four sets of experiments with the following goals: (1) test the flood prediction algorithm using a large set of physical river flow data (Section 4.3.1), (2) demonstrate long-term data collection of river flow data with a sensor network and characterize the sensor network (Section 4.3.2), (3) test the networking capabilities of our two-tier sensor network in a rural setting (Section 4.3.3), and demonstrate integrated system performance for flood prediction (Section 4.3.4).

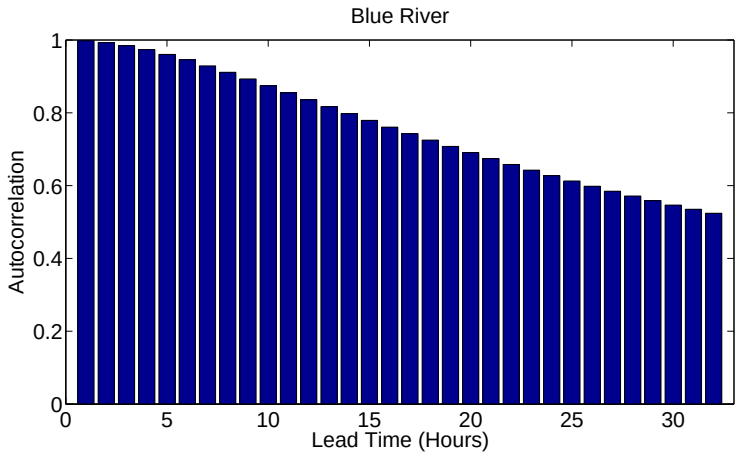

Figure 4.8: Autocorrelation of Blue River Data

4.3.1 Blue River

In this section, we use a large data set of real river data to test the validity of our prediction algorithm. For our river we examine the Blue River in Oklahoma, which encompasses 1233 km^2 at 153 m above sea level. No watersheds exist above this basin. The area has an average summer temperature of 82 °F and an average winter temperature of 36.7 °F with the occasional blizzard and tornados.

Test Data and Setup

To analyze our algorithms, we use 7 years of data from May 1993 to July 2000, measured from 1 river flow sensor, 6 rainfall sensors, and a weather station for the Blue River in Oklahoma [68, 93]. We use only temperature from the weather station as we discovered that the other measurements are highly correlated with temperature.

Training Window (Weeks)	**Order (Memory) of Flow Data**	**Order (Memory) of Temperature Data**	**Order (Memory) of Rainfall Data**
4	5	1	14
12	5	1	14
24	5	1	14
36	2	2	16
52	2	2	16

Table 4.1: Order Calibration Results for Blue River

	Training Window (Weeks)	**Modified Correlation Coefficient**	**False Positives**	**False Negatives**
Our Model	**4, 12, 24, 36, 52**	**0.998**	**0**	**0**
DMIP Ave Uncalibrated		0.58	-	-
DMIP OHD Uncalibrated		0.71	-	-
DMIP LMP Uncalibrated		0.77	-	-
DMIP Ave Calibrated		0.70	-	-
DMIP LMP Calibrated		0.86	-	-
DMIP OHD Calibrated		0.86	-	-
Climatology 1 Hour	52	0.32	0	13
Persistence 1 Hour	0	0.998	0	0

Table 4.2: 1 Hour Prediction Results for Blue River

For the rainfall measurements, we average them before entering them into the model, as we expect to do when collecting the data on the sensor network. This river and data come from an on-going project called the Distributed Model Intercomparison Project (DMIP) run by the National Oceanic and Atmospheric Administration to compare hydrological models [77, 85]. The DMIP test provides more hydrometerological data for the models than our model uses, allows for calibration based on 1 year of data, and requires a 1 hour prediction of river level for assessment [77, 85].

We define three different criteria for determining the quality of our algorithms: the modified correlation coefficient (taken from DMIP [85]), the false positive rate of prediction, and the false negative rate of prediction. For the modified correlation coefficient, as with the standard form used in probability, the value ranges from -1 to 1 with 1 meaning the two data series are identical. With this metric, since we use

the definition from DMIP, we can also compare our models to those listed as a reference of quality. False positive and negative detections provide a more common sense criteria as minimizing these increases the confidence of the end user in the system predictions. To determine the false events, we determine the flood level for the Blue River from the National Weather Service's online prediction work.

Model Calibration

We implemented the model as described in Algorithm 4.1 in Matlab, starting with defining the training window and recalibration. To help define the proper training window, T_T, we ran our experiments over several time windows: one month, one season (3 months), two seasons (6 months), three seasons (9 months), and a complete year. This covers all reasonable time periods for any generic river and any greater period of time becomes intractable for our system. For now, either we do not recalibrate the coefficients after the initial training or we recalibrate after we observe a new full training time window. Figuring out the optimal value for this we leave for future work.

Given these two parameter definitions, we analyzed the remaining parameters describing the models to determine optimal values. To pick the best values, we sweep the order (the number of past values used) for each of the three input variables with and without including the error of the past prediction. We find the optimal for each of the three metrics: the highest modified correlation coefficient, the lowest number of false positives, and the lowest number of false negatives.

In addition to the model with these parameters, we computed predictions using two naive approaches: climatology (or predicting the average of all previously seen flow observations at that hour and date in past years) and persistence (or assuming that the flow will stay at its currently observed value). We also compare to the DMIP results, choosing the models that had the best modified correlation coefficient value for the Blue River (described in Section 4.3.1). The LMP model offers the best results; however this model is an instantiation of the SAC-SMA model described earlier (see Section 2.2), demonstrating the current operational centralized method. For this rea-

son, we include the model built by the National Weather Service Office of Hydrologic Development, called the OHD model, in order to demonstrate the best distributed model in current research. Our testing computes predictions for time periods of 1 hour (for comparison with DMIP) and 24 hours (as a more realistic prediction window).

Autocorrelation

To determine the viability of 24 hour predictions, we examine the autocorrelation of the data set. As Figure 4.8 demonstrates, while the values decrease, at 24 hours the river correlates to itself at a value of 0.627, which implies a reasonable amount of data exists for using past information to predict that range. This also provides room for improvement in the predictability of the river; if the river had autocorrelation values of 0.9 or so at 24 hours, we could simply use persistence to predict, but here clearly we have room for improving the prediction.

Model Results

We began by calibrating the model as described in Section 4.1 to determine the optimal number of past measurements (or order) for each variable for the different training windows. For the 1 hour prediction, all time windows performed equally well using the last flow value, last temperature value, and last rainfall value. In fact, examination of the resulting coefficients demonstrates that only the latest flow value is used; basically, at one hour, persistence provides the best approach. Table 4.1 lists the orders that resulted in the best modified correlation coefficient for the 24 hour prediction case. We determined that better results always occurred when including as an input the error of the prediction associated with the latest observation and that our simple recalibration scheme did not improve results (implying that we either need a more complicated scheme or not much changed in the river basin for the hydrologically short time window of our data). Finally, maximizing the modified correlation coefficient provided the best overall results compared to minimizing the false rates.

Tables 4.2 and 4.3 show the overall results for this river, comparing our model and two naive approaches as well as the best cases for calibrated and uncalibrated DMIP models. In Table 4.2, examining the modified correlation coefficient, persistence and our model perform the best for 1 hour predictions. At 24 hours, our model performs better than persistence at all training time windows and both clearly perform better than climatology. Figure 4.1 demonstrates these results, showing our model with 52 weeks of training data and no recalibration. Additionally, although the DMIP results only apply to 1 hour predictions, comparing DMIP to the 24 hour predictions shows our model and persistence outperforming the average uncalibrated DMIP model. The average calibrated and best models from DMIP outperform our model with the average calibrated better than our model by 3%. This may seem like an unusual comparison; however, the lack of 24 hour prediction DMIP models limits us to this comparison, which does show that our 24 hour prediction is competitive with the 1 hour DMIP predictions.

	Training Window (Weeks)	**Modified Correlation Coefficient**	**False Positives**	**False Negatives**
Our Model	52	0.64	25	9
Our Model	36	0.61	25	9
Our Model	24	0.59	18	14
Our Model	12	0.59	18	14
Our Model	4	0.59	18	14
Climatology	52	0.32	0	13
Persistence	0	0.58	17	7

Table 4.3: 24 Hour Prediction Results for Blue River

4.3.2 Dover Field Test

We tested the long-term behavior of the system, specifically the sensing and 900 MHz communication, at Dover, Massachusetts on the upper Charles River from October through November 2007. This site allows us both to quickly identify any system issues without a trip to Honduras and to run longer tests, collecting data for our prediction modeling work and discovering any long range system issues. The data gathered at Dover allows us to connect the system and modeling work by running the information gathered there with our sensor network system through our Matlab model.

We installed 3 distinct sensor nodes (1 rainfall, 1 temperature, and 1 pressure sensor) within 900 MHz radio communication range at the locations shown in Figure 4.9. At no time could we achieve the claimed 32 km range of the Aerocomm radios, seeing a maximum fully functional range of 1.6 km. We performed tests on development platforms to rule out implementation issues with our system, installed a variety of antennas, utilized the configuration options available, and contacted Aerocomm to discuss the issue. None of our experiments improved the range. However, while not ideal, it is still sufficient for testing purposes; we intend to replace the module in future designs. The pressure sensor we placed within a USGS sensing station, using their concrete shed as a base for our system. The other sensors we located across the river, with the rainfall sensor across from the pressure sensor and the temperature sensor upstream of both. While we could have collocated the rainfall and temperature sensors, we chose to keep them separate in order to maximize our testing of the 900 MHz network and better understand any problems related to the specific sensors.

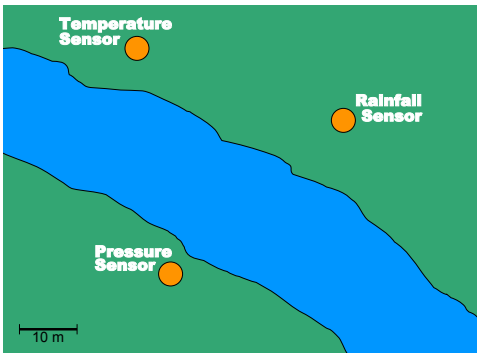

Figure 4.9: Locations of Sensors at Dover Site; Map Based on GPS Measurements and Surveying

With the system we gathered 5 weeks worth of data before ending the field experiment due to winter weather. Figure 4.10 shows the hourly rainfall, pressure, and temperature measured by the nodes over the complete experiment. While no flood occurred during this time period, we do see a variety of interesting behaviors such as a large amount of rainfall occurring at hour 251 and a period of no change occurring right before from hours 90 through 250.

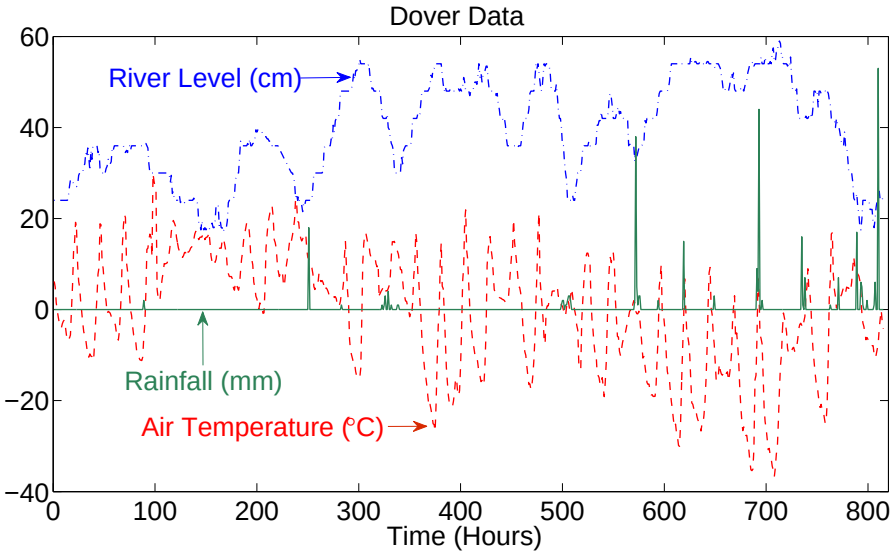

Figure 4.10: Data Collected from Dover Test Site

To explore the usefulness of this data for our modeling efforts, we then run this data through two of the models discussed in Section 4.1 for 3 prediction times using 4 weeks of training data. With 4 weeks of training, we computed the ideal orders of each of the variables as shown in Table 4.4. Table 4.6 shows the results of these modeling runs, demonstrating again that with regression models we can improve over inherent autocorrelation of the river state. Ideally we would have more data in order to use more reasonable test windows and a larger range of training windows as we did with the Blue River, but, with the onset of winter, the upper Charles River freezes, limiting us to this amount of data. As the focus of the experiment was on the functionality of the sensor system and not the simulation results from the data, the important result is that we can operate the system for over 5 weeks and use that data in our model for reasonable results. We also used the results of these model, specifically the coefficients obtained, to test the distributed prediction algorithm through a similar experiment in Dover in fall 2008.

We also characterized our power needs. We quantify some of the node operations in Table 4.5. Over a 5 day period, we saw a total discharge of 3153 mAh, which results in an average current of 26 mA. Charging over the same period resulted in a total of 1248 mAh. Two factors contributed to this less than ideal result: the extensive

Prediction Time	Order of Site Pressure Data	Order of Temperature Data	Order of Rainfall Data
1	1	0	3
16	3	0	5
24	2	0	5

Table 4.4: Order Calibration Results for Dover Site

Component	Current (mA)
Base Board Sleep Mode	2
CPU Low Usage	16
CPU Max Usage	59
Base Sensors	6
900 MHz Radio Receive Only	20
900 MHz Radio Transmit 1 Hz	73
River Extension Board	<1
144 MHz Radio Transmit	5000

Table 4.5: Sensing Node Power Budget

tree cover in the area limited effective panel placement, and many days saw heavy cloud cover and rain. We plan to increase the battery capacity and develop some tree installation strategies for future deployments.

Overall, the field experiment successfully tested the sensing node functionality for over a month, providing insights into the power operation, usefulness of the site for future testing, and the reasonableness of the prediction results for up to 24 hour prediction windows with four weeks of training data. We utilized these results to further develop the system in Honduras.

4.3.3 Honduras Field Tests

Our experiments in Honduras tested the two-tier architecture, deployment and maintenance issues, and issues specific to implementing these systems in developing countries. The work there began in January 2004 with the tower installation occurring in August 2005, the first communication test in March 2006, the test with sensing nodes and the office installations in March 2007, and the water prototypes throughout. All infrastructure remained with only the electronics removed to MIT for further work. We collaborate with a local non-governmental organization, the Fundación San Alonso Rodríguez (FSAR), to install the systems and understand deployment issues.

	Prediction Time (Hours)	**Modified Correlation Coefficient**
Our Model	1	0.9925
Our Model	8	0.770
Our Model	16	0.596
Persistence	1	0.9923
Persistence	8	0.733
Persistence	16	0.554

Table 4.6: Comparison of Model Results for Dover Data

On the communication side, we verified our two-tier approach. We first focused on the usability of the 144 MHz radios. To communicate at these ranges reliably, the radio antennas need line-of-sight high in the air, which requires antenna towers and limits the ability to test this portion of the system in the US. With FSAR help, we arranged access to land and built 5 meter antenna towers at two river sites where we plan to install water level sensors for 144 MHz radio communication (see Figure 4.11) along with 10 meter towers at the FSAR office and the government emergency management office in Tocoa. With these towers, we verified both the communication range and the ability of our modems to communicate data over this range. Sending from our furthest tower 53 km away, with the radios set to the lowest 25 W transmission setting, we received all data packets transmitted. With no towers further away, we were unable to determine the maximum range possible. Due to hurricanes in 2005, we also proved that the towers and antennas will survive hurricane force winds.

Next, we added 4 sensing nodes to the system for a 4 day test. While no interesting weather occurred, this did verify collecting data from the sensing nodes, transmitting that data over 900 MHz to the radio nodes, and retransmitting that data over 144 MHz.

At the offices, in addition to the towers, we worked to design and install secondary solar power systems. We would prefer to use grid power if it exists, but need solar power backup for the daily fluctuations of that system along with the major outages associated with disasters. FSAR worked with a local company to purchase panels,

Figure 4.11: 5 Meter Antenna Tower for Radio Nodes at Saba

batteries, and a charge controller. We added an off-the-shelf inverter, a power strip, and simple custom electronics to switch to solar at the absence of grid power. We installed these systems at both offices and ran long-term usage tests. At the government office, we also installed a permanent radio and laptop for development of that interface, using it both for longer term radio tests and exploring issues with the interface.

Another area of testing was the water measuring system. We created five different prototypes of this system installing each for several months in Honduras with the help of FSAR (see Figure 4.12). Through these prototypes, we settled on measuring water pressure as a method of obtaining river level. Other options such as resistive water level sensors were rejected due to corrosion issues, while ultrasonic sensors were rejected due to the indirect nature of the measurement along with reduced ability in high winds. These prototypes allowed us to understand the complexities of installing something in a flooding river since box movement reduces the efficacy of the measurement. Structures must hold the sensor in a fixed spot while ensuring

Figure 4.12: Installation of Water Level Prototype by FSAR Employees

the system does not sink in the soft ground of the river and that it is retrievable for maintenance. We developed two different solutions allowing us to install the system on a bridge for greater reliability and also in the middle of the river when the situation necessitates.

All of this work helped create the infrastructure necessary to achieve our goal of a demonstration system.

4.3.4 Integrated System Testing

We performed two integrated systems tests demonstrating the full functionality of the system. These tests proved the operation of all node types, the implementation of the distributed prediction and calibration algorithms on the system, and the combined operation of the system. As the focus of the tests was system functionality, having proven the model operation in simulation, we did not try to replicate simulation environments or results. The first test occurred in March 2009 in Honduras, but ran into technical difficulties with recording the data so we repeated the experiment in November 2009 in Dover.

The Honduras field experiment took place in March 2009. The experiment consisted of two sites: Olanchito and Tocoa. The Olanchito site had 2 sensing nodes measuring air temperature and rainfall, and 1 radio node that also measured upstream river level. The Tocoa site held an office node and provided an interface for

the local emergency management agency. While the experiment successfully measured and computed future river level predictions using the distributed algorithms, when the system was recovered at the end of the experiment, the SD cards were corrupted and no data survived. We returned the US and debugged the corruption issue to a non-deterministic hardware problem with exiting power-down. As the system powered on the 900 MHz radio, occasionally the radio caused the overall power level to drop below the minimum for the system, thus putting the SD card in an unknown floating state which corrupted the file system and ruined the data. We fixed this problem and prepared for a second experiment.

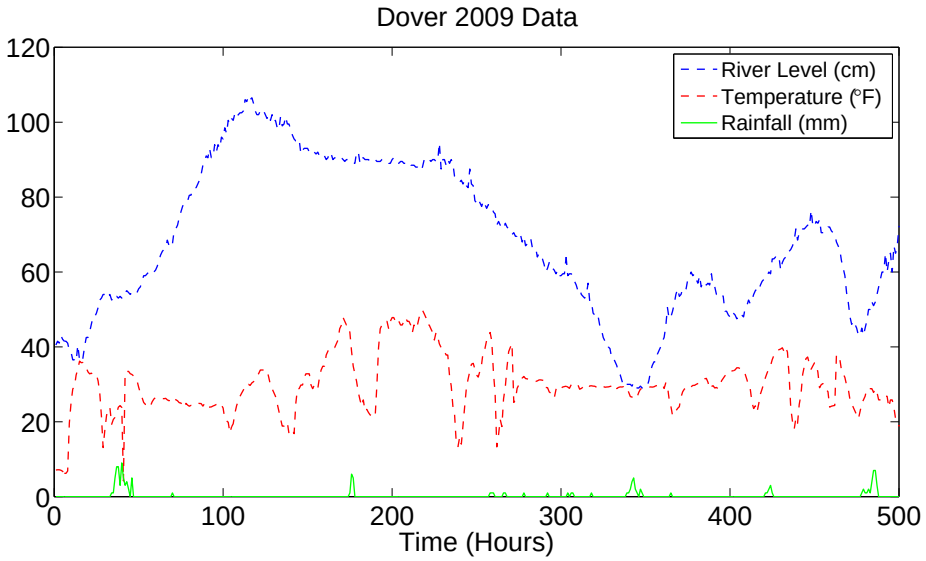

Figure 4.13: Dover 2009 Data

For our second attempt, we decided to perform the experiment in Dover, Massachusetts, removing the complexities of working over a 50 km area and in a developing country. Within a 5 km area, we installed 1 office node, 1 radio node, and 3 sensing nodes measuring air temperature, rainfall, and water level. The sensing nodes measured every 5 minutes, transmitted status information every 10 minutes, and transmitted the average data value every hour. The radio node also initiated the calibration algorithm every hour. Calibration used a training window of 10 hours for a 2 hour prediction, requiring 16 hours to pass before the first calibration could occur (10 hour training plus the 2 hour prediction plus 4 past river level values as

described next). The radio node stored 2 columns containing the last 2 river level values, the pressure sensing node stored the next 2 river level values (so the last 4 values were in the calibration matrix), and the temperature and rainfall sensing nodes both stored their last 2 values, resulting in a 10×8 calibration matrix. With these coefficients, the nodes computed a 2 hour prediction of the river level, which the radio node transmitted via 144 MHz to the office node hourly.

This experiment ran for approximately 5 weeks. Figure 4.13 displays the data gathered in the experiment. We did not design the system to withstand the freezing cold of Massachusetts winter and saw the effects as the experiment ran into December. Toward the end, the cold weather started to affect the sensor performance, with the temperature swinging less and the pressure sensor seeing more variation. Figure 4.14 shows the prediction results compared to the observed river values.

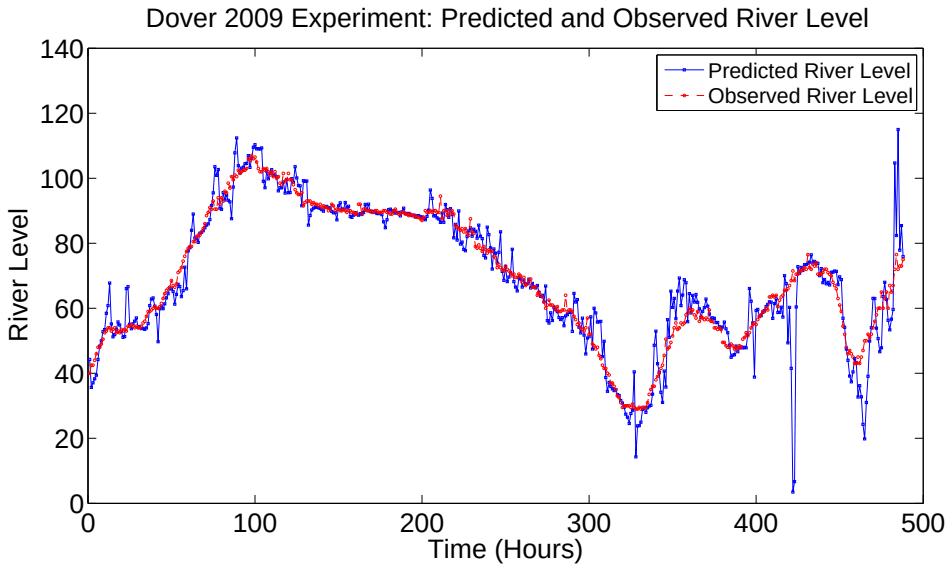

Figure 4.14: Dover 2009 Results: Predicted and Observed River Level

While prediction accuracy was not a goal of this experiment, exploring the reduced accuracy seen in Figure 4.14 provides insight into using regression models for prediction in an operational sense. We can understand the degradation of the prediction toward the end of the experiment by examining Figure 4.15. This figure shows the evolution of the calibration coefficients over time. The points with the most discrepancies between predicted and observed are also the points where the coefficients

weight the prediction more towards temperature and rainfall. This weighting occurs because at this time all three variables are changing at the same time, in the same direction, and with roughly the same rate of change; the calibration algorithm sees all as predictive of the river level and weights them differently. This reflects a common problem with data-based models: they depend on the data. Unusual behaviors and patterns in the data will affect the model. As this also is the key benefit of these models, we should not dismiss the model because of this, but rather include additional controls for restructuring the model and using the results.

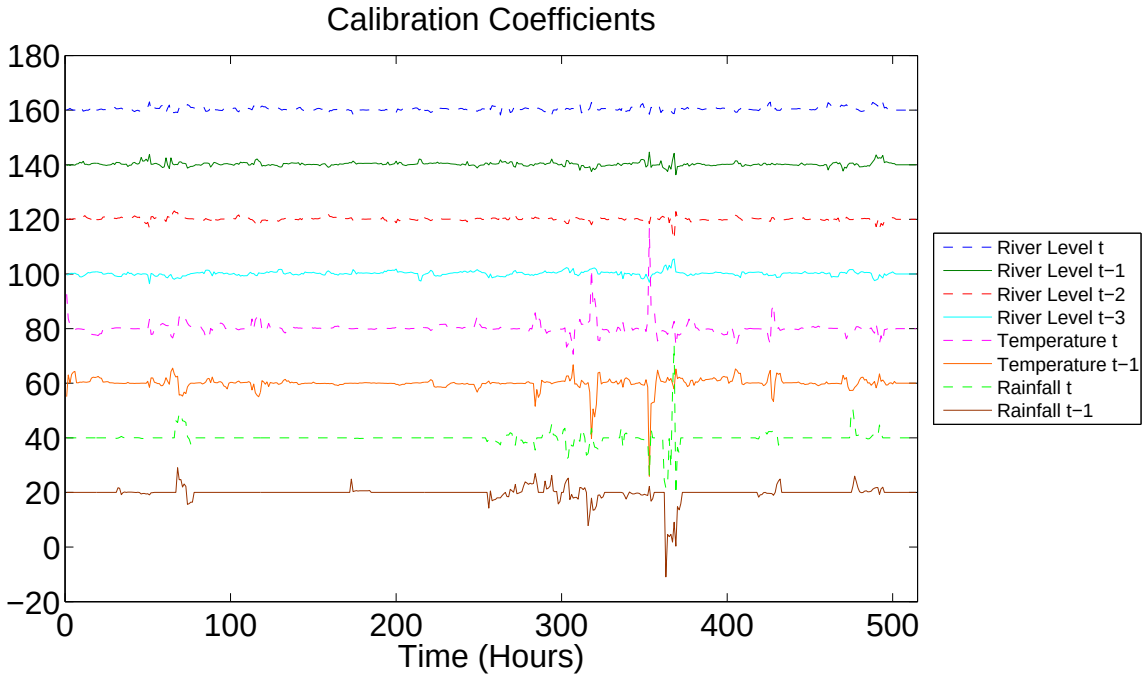

Figure 4.15: Dover 2009 Calibration Coefficients Over Time

If, to avoid this particular weighting issue, we remove temperature and rainfall from the model matrix, we see the cleaner prediction time series shown in Figure 4.16. This suggests future work in additional algorithms for dynamic redefinition of the matrix and post-analysis of the prediction. Prior to calibration and prediction, additional algorithms that examine data trends and redefine the matrix structure (such as variables included in the model matrix and number of calibration rows) would reduce the likelihood of unusual data correlations. Post-processing the prediction through analysis of information such as the rate of change of the prediction and

how the prediction relates to the other measured parameters (for example a flood prediction when no rain has occurred) would provide methods of controlling the use of the model and improve warning alerts. One important detail is that the prediction does recover quickly, a result of re-calibrating so often and another method of correcting possible divergences in the calibration coefficients.

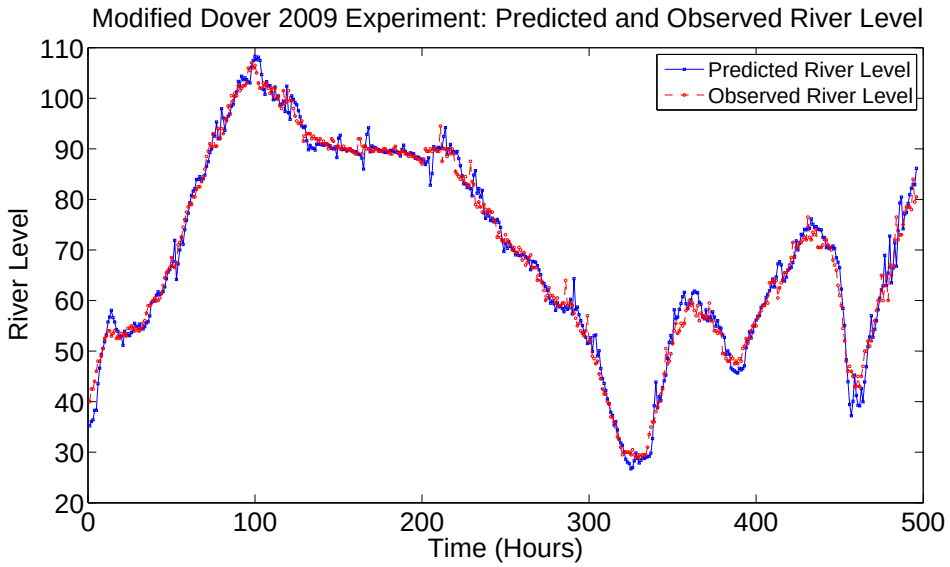

Figure 4.16: Modified Dover 2009 Results: Predicted and Observed River Level

With this experiment, we demonstrated the sensor network's ability to autonomously compute our distributed algorithms, thereby calibrating the prediction model and predicting river level. This validates the full functionality of our proposed solution.

4.4 Conclusions

In this chapter, we described an architecture for predictive environmental sensor networks over large geographic areas. These systems are node-limited due to region size and cost constraints. They also have significant system requirements due to the real-time need for the data, destructive events, and long operational lifetime.

Our sensor network solution addresses these requirements, consisting of two communication tiers, four node types, and support for a variety of sensor types. We focused on the event of river flooding, specifically in Honduras. The chapter describes our work on the flood prediction model utilizing the algorithms of Chapter

3 and the implementation of the sensor network architecture for this application. Locally, we installed 3 nodes on the upper Charles river at Dover and gathered 5 weeks of data, which we ran through our prediction algorithm, demonstrating both our system functionality and algorithmic functionality. In Honduras, we built several key pieces of infrastructure, including the radio antenna towers, and tested several system components. Finally, we performed integrated system tests, combining our distributed algorithms with the sensor network to demonstrate the full functionality of the system.

Chapter 5

Solar Current

In predicting river flooding, we ran into several problems with the power systems. This led us to consider how we could utilize our models to solve these power issues not only for our system, but for the general case.

Sensor networks deployed for extended periods of time need to manage their power usage. In the area of energy management, research has focused on two different areas: (1) reducing power needs of the hardware through careful design of the system and components, and (2) developing models and policies in software to control system behavior. Complementary to these areas is understanding the energy available to recharge the system. This understanding informs the policies running on the low-power hardware systems, helping create a smarter comprehensive energy management strategy. The questions then arise of how much power is available from the environment to recharge the system and how much will be available in future days? To answer these questions, we predict a node's future available harvestable energy and do so in a local, distributed fashion on the sensor network using in-situ sensed values.

Sensor networks monitoring the environment already collect a large amount of information about the surrounding climatic conditions. These conditions shape the amount of energy harvestable, whether that source is sun or wind or water. By utilizing the already existing measurements, we can better understand the future conditions that will effect the energy supply. We combine this information with past energy measurements to create a model, consisting of a set of time series that describe those conditions over several days. We can also include energy measurements

from other nodes and add a spatial aspect to our time series. Collecting all these data and creating these time series then allows a node to locally predict its future available energy either centrally on its own processor or in a distributed manner with its one-hop neighbors. Adding neighbors enables a more spatial representation of the conditions and better reflects the broader local environment.

A node predicts future solar current through the use of multiple linear regression models, specifically the algorithms developed in Chapter 3. To reiterate the basic concept, these statistical models provide a powerful tool for predicting future time series while remaining simple enough to compute on a sensor network. The node gathers an initial set of calibration data, self-calibrates the model, and then predicts the future value of interest. Because the node calibrates the model, whenever the prediction error exceeds some metric or enough time passes, the node can recalibrate the model to ensure adaptation to changing conditions. Additionally, by computing locally, the model can adapt better to local conditions and reduce communication through a sometimes failure prone gateway node. While we could perform this computation centrally on a node, this could affect the performance of that node, the data matrix is often too large to store within a single node, the eventual failure of the single node would halt the prediction process, and the computation may not scale to larger systems. Therefore, we develop a distributed solution that enables use in the larger set of scenarios.

In this chapter, we show the feasibility of this strategy, develop a solar current prediction model, and implement the model on a sensor network. Our models use solar current, humidity, soil moisture, air temperature, and wind speed to predict future daily average solar current. With 7 months of data from an installation in Springbrook, Australia, we improve over basic models by 11% and 13% (in summer and winter respectively). We also implement these models on the Fleck platform and demonstrate their functionality during a 7-week-long test. With the data from this test, we analyze the energy usage of our algorithms, determining they require $4.2 \times 10^{-8}\%$ of the weekly energy gathered by the system while providing an improvement in prediction compared to EWMA models and persistence.

We organize this chapter as follows. Section 5.1 outlines our modeling strategy and distributed algorithms. Section 5.2 discusses our simulation and implementation testing, proving our approach works. Section 5.3 describes extensions to our modeling work to provide a dynamic form of the model allowing for more fault tolerance and model development. We conclude with Section 5.4.

5.1 Prediction Model

In this section we describe a statistical model for predicting future daily average solar current using local measurements from a spatially-distributed sensor network. This prediction can provide a key input into energy management of sensor networks, enabling intelligent policies and efficient usage of this resource.

Using MLR models, we wish to predict:

$$b = f(\phi, \theta, \rho) \tag{5.1}$$

where b is the future average daily solar current. Variable b is a function of ϕ, a time history of average daily solar current, θ, a time history of our neighbors' average daily solar current, and ρ, a time history of the node's other environmental measurements such as humidity, air temperature, leaf wetness, wind speed, and other values. These latter two sets of variables help outline the external factors affecting the amount of energy harvestable by solar including weather conditions and seasonal conditions. The regression model provides flexibility in defining these sets, allowing the sets to reflect the variables available in the network.

Algorithm 5.1 outlines a multiple linear regression model for predicting solar current using past solar current, nearby neighbors' solar current, and any available environmental variables. Lines 15 through 18 setup the calibration matrix. For each variable, we can use any number of past values to define the linear prediction; in calibration each past value becomes a column within the matrix. For example, in using the past solar current observations, we could use only that occurring at t, or

Algorithm 5.1 Solar Current Prediction Model

ϕ : past daily average solar current
θ : vector of other nodes solar current
ρ : vector of other environmental measurements
N : # past solar current values used
Q : # other solar current values used
P : # environmental values used
b : predicted daily average solar current
e : prediction error
T_T : training time window
T_L : prediction lead time
T_R : recalibration time window

$T_{TL} = T_T - T_L$;
▷ Compute initial coefficients and prediction
$\phi_N \leftarrow [\phi(1 : T_{TL} - N), .., \phi(1 + N : T_{TL})]$
$\theta_P \leftarrow [\theta(1 : T_{TL} - P), .., \theta(1 + P : T_{TL})]$
$\rho_Q \leftarrow [\rho(1 : T_{TL} - Q), .., \rho(1 + Q : T_{TL})]$
$X \leftarrow [\phi_N, \theta_P, \rho_Q]$
$C = ((X * X^T)^{-1} * X^T) * b(1 + T_L : T_T)$
$b(1 + T_L : T_T) = X * C$
▷ Recompute using prediction error
$e = b(1 + T_L : T_T) - \phi(1 : T_T - T_L)$
$X \leftarrow [\phi_N, e, \theta_P, \rho_Q]$
$C = ((X * X^T)^{-1} * X^T) * b(1 + T_L : T_T)$

for $t = T_T + 1$ to ... **do** ▷ Forecast
    if $(t\%T_R) == 0$ **then**
        ▷ Recalibrate coefficients
        $e = b(t - T_T : t) - \phi(t - T_T - T_L : t - T_L)$
        $\phi_N \leftarrow [\phi(t - T_{TL} : t - N), .., \phi(t - T_{TL} + N : t)]$
        $\theta_P \leftarrow [\theta(t - T_{TL} : t - P), .., \theta(t - T_{TL} + P : t)]$
        $\rho_Q \leftarrow [\rho(t - T_{TL} : t - Q), .., \rho(t - T_{TL} + Q : t)]$
        $X \leftarrow [\phi_N, e, \theta_P, \rho_Q]$
        $C = ((X * X^T)^{-1} * X^T) * b(t - T_T : t)$
    end if
    ▷ Compute Forecast
    $e = b(t - T_L) - \phi(t)$
    $\phi_N \leftarrow [\phi(t - N), .., \phi(t)]$
    $\theta_P \leftarrow [\theta(t - P), .., \theta(t)]$
    $\rho_Q \leftarrow [\rho(t - Q), .., \rho(t)]$
    $X \leftarrow [\phi_N, e, \theta_P, \rho_Q]$
    $b(t + T_L) = X * C$
end for

we could add $t-1$, $t-2$, etc. as we design the actual implementation of the model. In our algorithm description, we define the number of past values used as N, Q, and P for the node's solar current (ϕ), the neighbors' solar current (θ), and the environmental measurements (ρ), respectively. We load the matrix, X, with this data set over the calibration window defined. Line 19 performs the calibration step and generates a coefficient vector, C. To include the prediction error in the model, we then predict over the training window we just used by multiplying our calibration matrix by our coefficients (Line 20). We subtract the observed record from this prediction, thus generating our error, and include this in our new calibration matrix of Line 23. Recomputing the coefficients based on the new calibration matrix occurs in Line 24 and, with these coefficients, we begin predicting the future solar current in Line 26. This loop continues forever, recalibrating after a full recalibration time window passes (Line 27), computing the prediction error based on the latest observation (Line 37), and predicting future solar current (Line 42).

5.2 Testing: Simulation and Implementation

In this section we describe our testing procedures: first simulations in Matlab to verify the functionality of this model in predicting solar current and then field experiments on the Fleck platform.

5.2.1 Simulation

We believe that multiple linear models can accurately predict solar current, but need to verify this claim. We have an existing data set from a rainforest deployment in Springbrook, Australia that provides the relevant parameters. This data set consists of 1 year of solar current, humidity, soil moisture, air temperature, leaf wetness, wind speed, and wind direction, gathered by 10 sensor nodes. Due to some small gaps in the operation of the network and gateway node, we split the data set into a summer data set of 3 months from January through March and a winter data set of 4 months from June through September. Figure 5.1 displays the solar current, temperature,

Model Type			RMSE (mA)	Maximum Absolute Error
Summer	Our Model	1 Solar, 1 Wind Speed, Use Derivative	16.67	3.40
	Persistence		20.59	6.33
	EWMA		19.68	5.23
Winter	Our Model	3 Solar, 1 Wind Direction, 1 Wind Speed, 1 Leaf Wetness, 1 Soil Moisture, Use Prediction Error, Recalibrate	24.41	10.26
	Persistence		28.06	18.44
	EWMA		29.13	15.22

Table 5.1: Results of Three Different Models Predicting Average Daily Charge Current in Summer and Winter

and humidity measurements for these two data sets. Despite averaging them on daily boundaries, we still see a very non-linear time series with no trends in solar current values from day to day. There also appears to be no obvious correlation between the solar current and the temperature or the humidity.

To understand if we can predict solar current despite the non-linear data, we start by examining the autocorrelation of the solar current data. This measurement demonstrates the amount of information the past measurements of solar current provide to the current measurement. Values closer to 1 indicate the variables correlate well, implying the time series share useful information. As Figure 5.2 shows, the past solar current correlates to the future value reasonably well. The correlation is better than 0.71 out to 11 time steps in the summer and 0.72 out to 3 time steps in the winter. This suggests we can predict solar current with some combination of past values. However, since the values do not correlate perfectly, there is still a need for both models and additional data.

Next we implement the model using Algorithms 3.1 and 3.2 in Matlab. We predict the future average daily solar current 2 days in advance using 7 days of data for calibration (a value chosen based on the amount of data existing, ideally this would increase as more data arrives), starting with the scenario where the model uses envi-

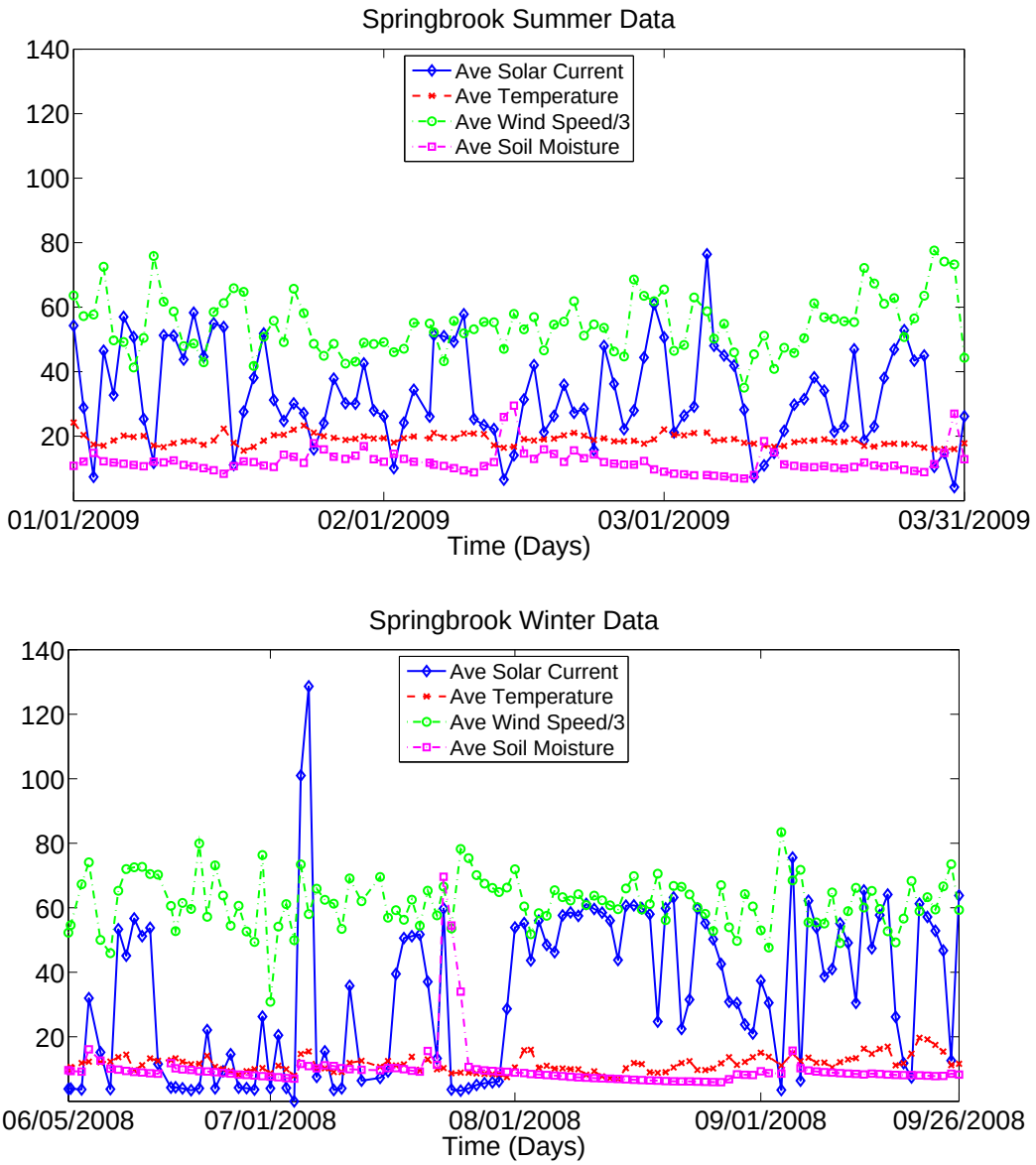

Figure 5.1: Springbrook Data Sets: (a) Summer Data and (b) Winter Data

ronmental variables and no neighbors' information. To the measured values, we also add the three possibilities: (1) recalibrating when the error exceeds a threshold and at least 4 days of new data exist in the matrix, (2) including the prediction error in the calibration matrix, and (3) including the first derivative of the solar current in the calibration matrix. We do not know which other variables most correlate to the solar current so vary all parameters and run the model over each possibility. To determine which combination provides the best prediction, we evaluate the predicted time series using the root mean square error (RMSE) between the predicted and observed as well

as the largest absolute error value. Table 5.1 outlines the combinations with the best results over both metrics.

We also compare our results to two other locally computable methods: persistence and exponentially weighted moving average (as suggested in [46, 58]). Persistence predicts that nothing will change and the future solar current value will be equal to the current solar current value. Exponentially Weighted Moving Average (EWMA) computes $b_{t+L} = \alpha b_{t+L-1} + (1-\alpha)x_t$, a linear combination of the current prediction and the most recent solar current measurement (x_t) that is weighted by a parameter α

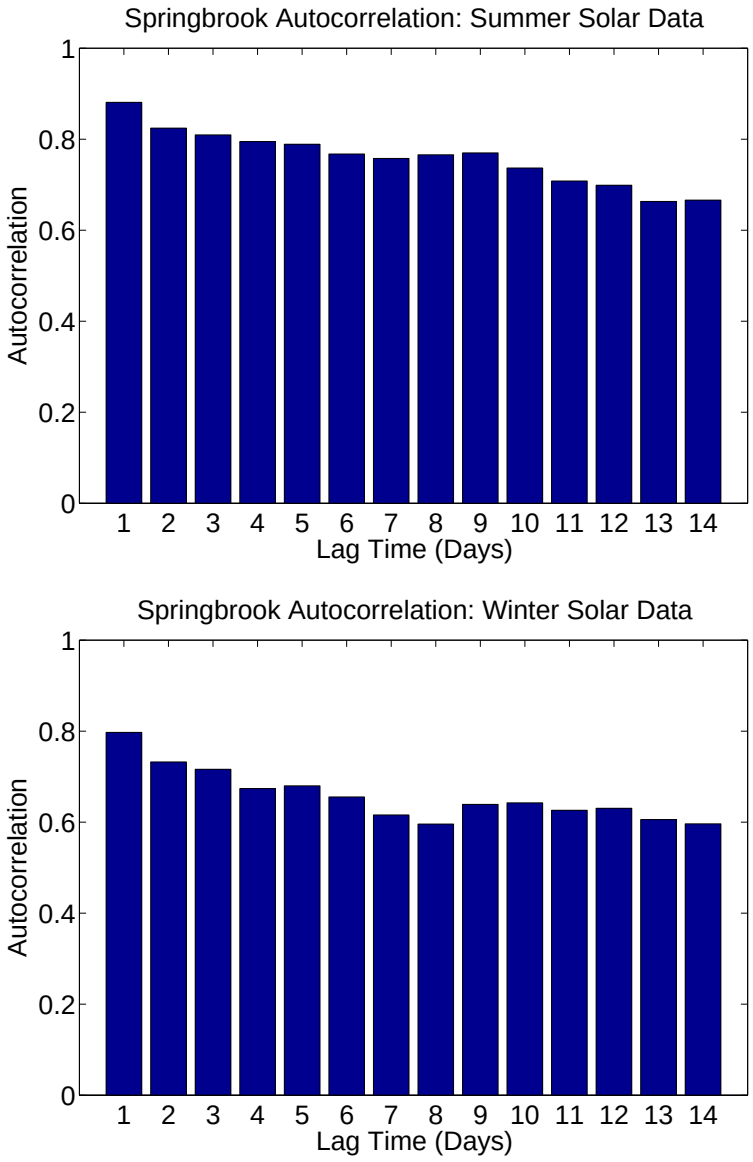

Figure 5.2: Autocorrelation of (a) Summer Data and (b) Winter Data

which we set at 0.15 as outlined in Hsu *et al.* [46]. The results of all these models appears in Table 5.1, which shows our MLR model improves over persistence by 19% and 15% (in summer and winter respectively), and improves over EWMA by 13% and 16%. In this table, it appears that we have several models depending on season. The table shows the best results for combined metrics; actually we have 46 and 2684 variable combinations that give results better than persistence for both metrics in summer and winter respectively. Currently we determine reasonable variable combinations by examining existing data sets and testing over the possibilities; we are working on methods to determine this dynamically.

Next we examine the usefulness of neighbors' solar current measurements within the model. Three other nodes (Nodes 7, 9, and 21) collected data during the summer months of January through March; none were direct neighbors of our primary node (Node 2) and collectively they span the monitoring region. We divide our examination into two sections: (1) is there any gain from including spatial factors into regression models and (2) what effect the spatial factors have in combination with our environmental factors. Table 5.2 shows our results. All three forms of our model improve over persistence and EWMA. Between the two versions of the spatial model, including the environmental factors increases the root mean square error while decreasing the maximum absolute error. These variations are slight, however. This suggests the addition of the environmental variables has little effect on the overall spatial model. The spatial model itself does not outperform the environmental only model, which simply implies that the environmental factors have more of an effect on the future solar current than these neighbors in this case. However, the spatial version still performs better than persistence. In cases where no environmental sensors exist or data storage limits the model to only distributed spatial data, this model can still provide valuable predictions of the future solar current for improving energy management.

5.2.2 Implementation on Fleck Network

We implemented our algorithms on a physical sensor network platform to verify their functionality and feasibility in a real world scenario.

Model Type		RMSE (mA)	Maximum Absolute Error
Our Model: Env. Only	1 Solar, 1 Wind Speed, Use Derivative	16.67	3.40
Our Model: Spatial Only	3 Solar, 10 Node21, 1 Node7, 9 Node9, Use Derivative	17.33	4.75
Our Model: Env. & Spatial	1 Solar, 10 Node21, 8 Node7, 7 Node9, 1 Wind Speed, Use Derivative	17.63	4.29
Persistence		20.59	6.33
EWMA		19.68	5.23

Table 5.2: Results of Three Different Models Predicting Average Daily Charge Current in Summer

Fleck Platform

The implementation uses the Fleck^{TM}3b platform for empirical validation. The Fleck^{TM}3b is a low power wireless sensor network device designed by CSIRO specifically for outdoor applications such as environmental monitoring. The Fleck^{TM}3b employs the ATmega1281 microprocessor running at 8 MHz with 4 Kbytes EEPROM, 8 Kbytes SRAM, and 128 Kbytes program flash. This low power microcontroller is combined with the Nordic NRF905 digital transceiver which enables the Fleck to communicate at 50 Kbps across 1 km with a 915 MHz quarter-wave whip antenna while consuming less than 100 mW of power. This platform can sense onboard temperature and power usage and is easily interfaced to numerous external sensors via the external sensor connector block and the daughterboard expansion bus. On the software side, it runs Fleck OS (FOS) [24], a cooperative threading operating system designed specifically for sensor networks.

Test Results

We tested this implementation on 3 Fleck nodes which we placed on the CSIRO ICT campus as shown in Figure 5.3. The goal of this test is to verify the operation of the algorithms on the Fleck nodes: do nodes correctly transition through the state machine controlling operations, do messages occur at the correct times and do the nodes respond in an appropriate fashion, and do the mathematical operations

Figure 5.3: Fleck Node Installed on Campus

compute correctly (no overflow issues or implementation issues). In achieving this goal, we focus on the implementation and not on replicating our simulation results. As part of this proof-of-concept test of the implementation, we only use solar current measurements in the matrix and have each node maintain a column of 5 values. We set the number of rows in the matrix to a value keeping the message size within 1 packet, allowing us to avoid multi-packet messages. This reduced message overhead and focused our tests on taxing the calibration algorithm, by utilizing a very short calibration window causing multiple iterations of many messages.

We ran the system for over 7 weeks. Initially, the system measured every 15 minutes, predicted every 15 minutes, and calibrated every 90 minutes. The system predicted the future current 2 measurement intervals in the future, or 30 minutes. Figure 5.4 shows one week of the data from this time period, aligned and averaged to the hour. The prediction performs reasonably well compared to the observations. An interesting phenomenon occurs in the graph with two sharp down spikes between the daytime period and the nighttime period. At night, the system actually is measuring the nighttime lights that illuminate the campus (an unforeseen effect). The two spikes then reflect when those lights turn on and off. Our predictions capture this unusual behavior as well.

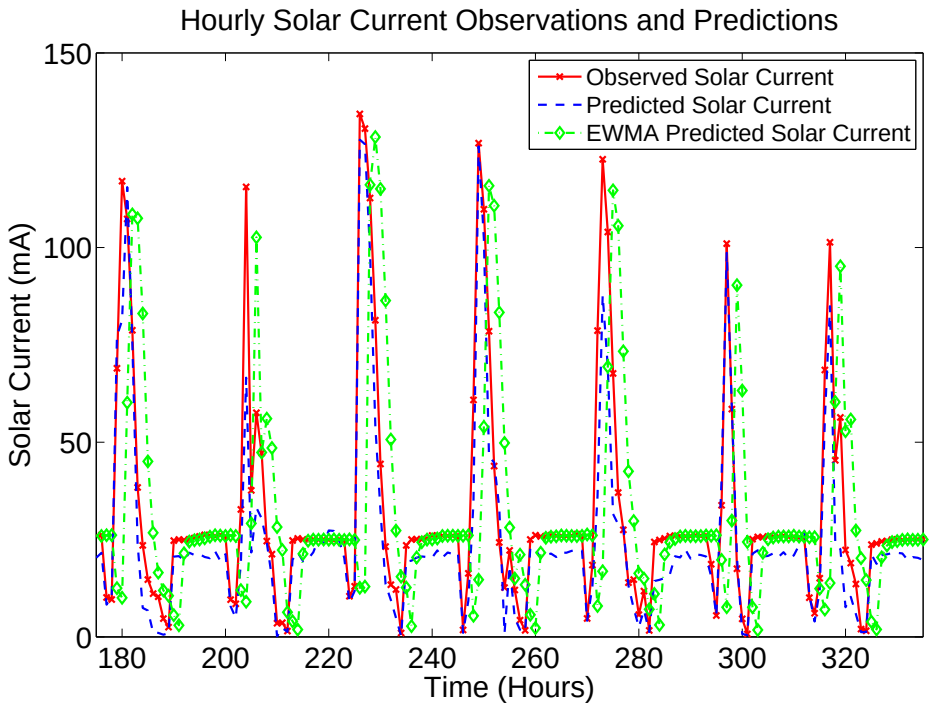

Figure 5.4: One Week of Observed and Predicted Data from CSIRO Test

After approximately 3 weeks, we changed the system parameters to tax the calibration algorithm more rigorously. We shortened the measurement window to 2 minutes and the calibration window to 6 minutes. This also shortened the prediction window to 4 minutes (as the system still defined it as 2 measurement intervals).

Over a two week period, the system attempted 2773 calibrations. 2679 of these were successes and 94 failed, resulting in a 96.6% success rate.The failures all occurred early on as a result of one node having a low initial battery voltage. This node shut down until the next morning when it received sufficient solar energy to resume operation. However, the system correctly identified the failures when the node did not contribute to the algorithm with the remaining active nodes timing out and resuming regular operations. They continued to attempt calibration and eventually succeeded, all without user intervention. These instances of low power states, while not ideal for our operation, also argue the need for smart energy management and the usefulness of our system.

	Hourly		Daily	
Model Type	**RMSE**	**% Error**	**RMSE**	**% Error**
Our Model	2.04	-24.6	0.91	-15.6
Persistence	3.61	91.7	2.52	17.8
EWMA	3.64	105.0	1.51	12.4

Table 5.3: Results of Models Predicting Average Hourly and Daily Charge Current for Data from CSIRO Test

The test achieved our goal, demonstrating the functionality of the implementation on a real sensor network platform. To connect the test to our daily average predictions from simulation in order to show the reasonableness of the prediction, we analyze the overall data set to see how well it predicted the average daily solar current. We average both the observations and predictions; Figure 5.5 shows the both the observed, the predictions from the system, and an offline prediction by the EWMA model (also average from hourly predictions). Our predictions match the observed well and better capture the peaks compared to the EWMA model.

We also analyze the RMSE error for the hourly and daily averaged data sets. Table 5.3 lists these values along with the percent error of each model, which verify the graphical analysis. Our model has the lowest RMSE error for hourly and daily compared to both persistence and EWMA. For hourly percent error, our model underpredicts by 24.6% while persistence and EWMA overpredict by 91.7% and 105.0% respectively. In comparison terms, this means that our model improves over persistence by 67.1% and over EWMA by 80.4%. For daily percent error, our model again underpredicts while the other two overpredict. However, our model provides a 2.2% better prediction compared to persistence and provides a 3.2% worse prediction compared to EWMA. This occurs due to the averaging effects; since we average the hourly predictions to obtain the daily, persistence and EWMA models average out the time delay of their predictions. If we only predicted daily values, both persistence and EWMA would have time delays on the peaks and valleys of the predictions, providing a significantly worse percent error. From an energy standpoint, the important trend of these values is that our model underpredicts compared to the other two

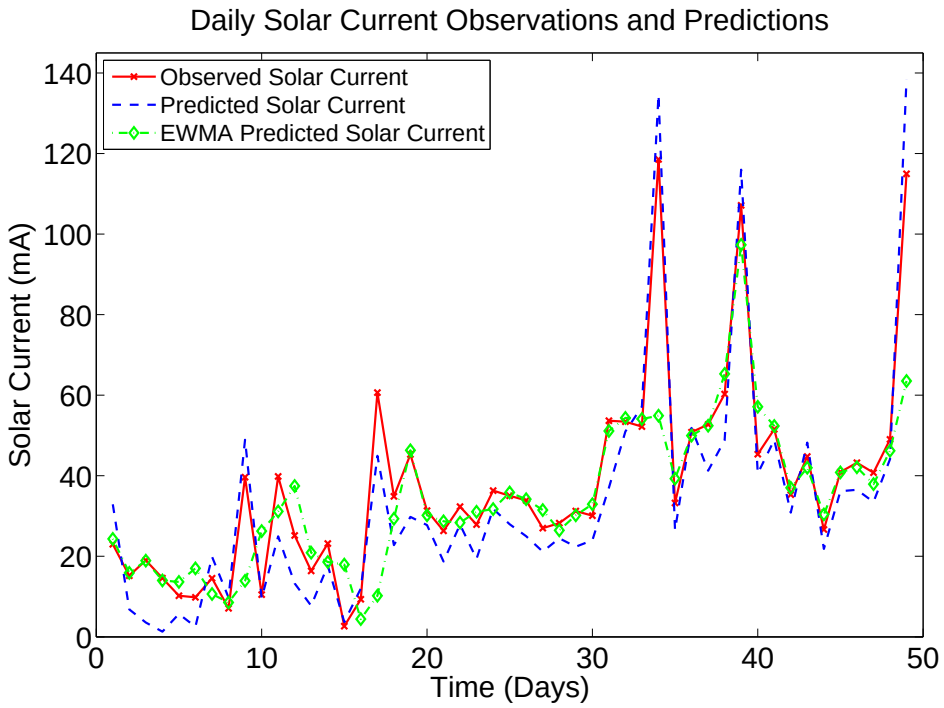

Figure 5.5: Average Daily Solar Current Observed and Predicted from CSIRO Test

overpredicting. Underpredicting ensures the system operates on a lower power budget than actually exists, allowing it to remain operational far longer. Overpredicting will convince the system to use more power than actually exists, reducing the system lifetime. Therefore, having a prediction model that underpredicts is preferable to an overpredicting one. Overall, this test demonstrated a functional and correct implementation of the distributed algorithm on the Fleck platform, connecting the theory to the real operation of the algorithms.

Energy

Despite the improved predictions, we must ask whether it makes sense from a energy standpoint to perform a more complex model. Table 5.4 outlines the relevant numbers.

First, from the prediction side, computationally, persistence performs the best as it requires no computation while our model and EWMA differ by 2 operations, leading to a $9.5 * 10^{-5}$ mJ increase in energy in order to compute our model. Distributing the

Model Type	Comp. Ops	Comp. Energy (mJ)	Comm. Msgs	Comm. Energy (mJ)
Prediction:				
Our Model	5	$2.38 * 10^{-4}$	2	1
Persistence	0	0	0	0
EWMA	3	$1.43 * 10^{-4}$	0	0
Calibration:				
Our Model	8400	0.4	6	3.1

Table 5.4: Energy Requirements for Models

prediction for our model requires 2 32-byte messages, which requires 1 mJ of energy, while the other methods have no communication requirement and, thus, require no energy.

On the calibration side, only our model requires calibration. This calibration, using the parameters of our field test, requires roughly 8400 operations (only 6.7 ms processing time). On the 8MHz Fleck platform operating at 3.3 V and requiring 18 mA active current, this results in 0.4 mJ of energy. Communicating the messages required for calibration, 6 messages total at a maximum transmit power of 100 mW and data rate of 50 Kbps, requires 3.1 mJ of energy. To put these numbers in perspective, during our field test, the system gathered 37.5 mA daily (on average). This resulted in $1.07 * 10^4$ J of energy daily (296.7 Wh). If we performed calibration in accordance to our outlined model, calibrating every 7 days, the calibration operation would use $4.2 * 10^{-8}\%$ of the weekly energy. As the trend for our model is underpredicting, we would easily recover the energy costs of computing our model through the energy savings incurred by using the prediction to manage power. In return for this energy expenditure, we also see an improvement in our solar current prediction over EWMA and persistence. Achieving such an improvement costs so little from an energy standpoint that we conclude it does make sense to compute our more complicated model to aid better power management that could easily recapture the energy expended on prediction.

Discussion

Based on these results, we elaborate on several aspects of the system.

First, we saw single node failures during our testing, but avoided multiple node failures. What would happen if a multiple node failure occurred? Should any nodes other than the master node fail, the network would continue to attempt calibration, miss the needed communication, and cancel the rest of the algorithm. Should the master node fail, the calibration would not occur. Future work entails adding data redundancy to the algorithms such that the failure of a node still allows the calibration algorithms to continue.

Another thing to note regarding these results is that the failure of the calibration algorithms does not mean the failure of the system. In cases where the calibration fails to complete, prediction does continue with the old coefficients. Energy management can still proceed with predictions of future solar current; however the coefficients may be less accurate. Slightly less accurate predictions will not cause our network to fail. Any time series prediction will oscillate around the real values, but the aggregate behavior should match the real solar current behavior; this we see in our work. As long as the aggregate matches and predictions do not oscillate too far from the real behavior (which we do not see in our work), any energy management system will be able to maintain operation. Some days will require using more energy from the power storage system than expected, but this will be corrected on those days where extra power is stored. Overall, the predicted solar current will lead to a more comprehensive energy management system, which should ensure the continual operation of the network.

Additionally, we can explore the trade-offs regarding the time windows for sensing, prediction, and calibration. All the timing is sensitive to the phenomena and data being predicted, but we can make some generalizations. Small time windows for sensing and prediction occasionally result in models too sensitive to perturbations within the data; the system then requires more frequent calibration to adjust and some amount of smoothing. For the case of solar current at a medium time scale (approximately of the order hours), the cyclic nature of the data with high values

during the day but near zero values at night can confuse the model, ensuring it does not accurately predict either. In our results, Figure 5.4 sees the model not quite matching the nighttime values and, on occasion, not quite reaching the heights of the daytime values. Accurate models at these medium time scales might require operating only with one portion of the cycle, but the movement of the transition points between night and day and vice-versa can still cause difficulties. Larger time scales, such as the daily values we use, smooth over the cycles and perturbations to ensure a data set more amenable to time series predictions. However, this does require more time to gather data for calibrating and recalibrating based on seasonal trends in the data. Our choice of daily time windows ensures a reasonable prediction while minimizing the amount of computation (and thus power usage) needed by the model, allowing more time for monitoring the environment and the operational goals of the sensor network.

Finally, we want to consider the definition of the matrix and determination of appropriate policies. We used the existing simulation data to define reasonable model structures and chose fixed windows for calibration and prediction. Ideally, the system would dynamically determine this, either centrally or distributed, allowing for a more adaptable model. This would also allow for a variety of prediction windows and dynamic growth of the calibration window as more data arrives. With dynamically defining these parameters, we need to also decide the optimal strategy for predicting, whether it makes more sense to utilize the same prediction for a node cluster or have each node predict its own current. This relates to node density and the variability of the environment. A more heterogeneous setting may lead to better individual predictions while a homogeneous setting allows for reasonable cluster predictions; a large network most likely will need a combination of approaches. To achieve this, we first must include manual methods for defining these policies and then consider automated methods. We leave this for future work, recognizing the importance of answering these questions to enable a portable prediction model.

5.3 Dynamic Prediction Model

In this section, we describe our initial work to define a dynamic prediction model. We first discuss the design, covering the issues, trade-offs, and requirements. Next, we introduce our implementation on the Fleck and discuss our tests.

5.3.1 Design

One clear issue with using multiple linear models is the definition of the variables stored within the matrix. Our methodology relies on an existing data set upon which to perform simulations in order to define the best matrix structure. However, we would prefer that the system learns the structure of the matrix so that the model and prediction better reflect the existing conditions.

As an initial step towards this goal, we develop tools to allow the user or central location to redefine the model. Redefining the model includes determining which variables the matrix uses, how many past values of each variable, the calibration time window, and how the system distributes these values among the participating nodes; the same variables and values we defined through our simulations. The values define the matrix: the number of rows is equal to the calibration time window, and the number of columns is the total number of past values used for all variables. Each node needs the additional functions to process messages for changing the model, to change the relevant values, and to compute any dependent variables. In addition to changing the variables defining the model, we need to consider how to store the matrices as well as storage optimizations and limitations.

Before addressing these issues, we should discuss our choice of platform as it affects our design options. We intentionally performed these initial experiments into dynamic models using the Fleck platform and not our own. Our platform could support these changes and actually has a couple advantages over the Fleck. The ARM processor we use provides significantly more internal storage than the ATmega1281 used in the Fleck, and we have a FRAM and SD card for even more storage; this would allow for larger structures than the Fleck platform. Additionally, our Aerocomm radio allows

larger packets than the Nordic radio; this also eases implementation by dynamically supporting changing packet sizes as necessary when changing the matrices' sizes. However, the Aerocomm radio also provides the greatest source of problems in our platform with unreliable communication distances and success rates. FOS, the Fleck operating system, also provides support for "remote procedure calls" (RPCs). While fundamentally messages and message handlers, the tools for adding the messages, message handlers, and user interface function calls is easy to use and speeds up development on the central user (or computer) side. Accordingly, we implemented our initial forays into dynamic models on the Fleck platform.

Our first design issue involves defining the matrices' structures and variables. We have 4 permanent matrices: A, R, U, and V, 2 vectors: b and x_{opt}, and some number of temporary matrices and values. Of these, A, R, b, and x_{opt} are distributed among the participating nodes; U and V are stored by the prediction instigating node. Since microcontrollers do not support malloc (or, more precisely, use of malloc tends to cause problems within microcontrollers), our only recourse is to pre-define these matrices to the largest reasonable size while still allowing the code to compile. We can use our experience to create some heuristics to aid defining the "largest reasonable size"; Algorithm 5.2 depicts the on-node variable definitions for this. First, models rarely require a large number of past values for secondary variables. In the case of solar current, the solar current of the nodes and neighboring nodes are primary variables, but the environmental conditions are secondary variables. Second, given a trade-off between more past values of the primary variables and more calibration values, our results suggest having more calibration variables is better. This allows nodes to all have the same size storage for A and other matrices, which will aid in reformatting the matrix. In cases where the number of nodes is greater than or equal to the number of variables, having roughly the same storage also allows assigning one variable per node without worries about lacking storage for them.

In addition to optimizing variable definitions, we consider storage optimizations. Currently, we store data in a circular buffer before loading the A matrix; this eases the computational requirements as it avoids shifting each column of the matrix after every

Algorithm 5.2 On Node Matrix Definition

N_{max}: maximum number of nodes participating in computation
p_{count}: maximum number of possible primary variables per node
p_{max}: maximum number of primary values
s_{count}: maximum number of possible secondary variables per node
s_{max}: maximum number of secondary values
c: maximum number of calibration values

All nodes declare:
$A = zeros(c, p_{max}p_{count} + s_{max}s_{count})$
$R = zeros(N_{max}p_{max}p_{count} + N_{max}s_{max}s_{count}, p_{max}p_{count} + s_{max}s_{count})$
$b = zeros(c/N_{max}, 1)$
$x_{opt} = zeros(p_{max}p_{count} + s_{max}s_{count}, 1)$

Prediction node declares:
$U = zeros(N_{max}p_{max}p_{count} + N_{max}s_{max}s_{count}, N_{max}p_{max}p_{count} + N_{max}s_{max}s_{count})$
$V = zeros(N_{max}p_{max}p_{count} + N_{max}s_{max}s_{count}, N_{max}p_{max}p_{count} + N_{max}s_{max}s_{count})$

observation period. As we reach the limits of our existing storage, we will remove this buffer and store the values directly in the matrix, saving storage space at the expense of computation time. This will also require limiting recalibration until enough time passes to refill the matrix as the operation of calibration will destroy the existing data. Another possibility is utilizing an FRAM to store the values until a calibration event occurs and then use a smaller amount of storage within the microcontroller. The Fleck does not have a FRAM, but we could use this optimization on our platform.

We next need to define the procedures necessary to implement a dynamic model. Algorithm 5.3 outlines the behavior of the model. From the system side, we first need user interface functions to manually reset the parameters of each node, which will trigger a message to that node. Within this message, to correctly compute the algorithms, each node needs to know the total number of columns, m, and the total number of rows, n. Next, each node in the sensor network must support changing the description of which portion of the matrix it stores. For each variable, it stores the variable type, the number of past values, and the location of the columns within the matrix (this defines when it participates in the QR). We ensure in defining the variables that each node understands how to get the variable data it needs, either by

measuring, computing, or monitoring packets in the unlikely case where the system assigns variables for which it has no sensor.

Ideally, since the system receives regular updates of the observations and predictions, we can add an automated analysis of the behavior to define the quality of the prediction and redefine the matrix to improve the prediction. For situations where little is known about the process being predicted, we can use this to automatically run through a variety of model structures to determine the best one, allowing the system sufficient time to run the model, analyzing the results, and changing the model definition in a continuous loop. Determining the best approach for this and designing the algorithms we leave for future work.

Algorithm 5.3 Dynamic Matrix Re-Definition

User or automated system defines matrix at Central Location

Central Location transmits to Node j:
msg = [total columns, total rows, number variables stored, variable, number of values, matrix column location]

Node j receives message and updates:
m =total columns
n =total rows
v_{count} =number variables stored
for $i = 1 : v_{count}$ **do**
    $var[i].type \leftarrow$ variable
    $var[i].number \leftarrow$ number of values
    $var[i].location \leftarrow$ matrix column location
end for

None of this requires modifications to our existing distributed algorithms. From an implementation standpoint, we eventually want to transition to using the fully distributed linear regression calibration algorithm (Algorithm 3.10) as this will enable larger matrices. For now, our existing implementation references variables easily changed outside of the algorithm.

Having defined what we want to achieve and discussed the design issues, we can now implement these operations.

5.3.2 Implementation

We implemented these ideas on the Fleck platform. For storage, we pre-define the possible matrix sizes to the largest possible, keeping our current practice of having a separate data buffer. Based on our results, we limit the number of values for each secondary variable to $s_{max} = 1$ and allow inclusion of $s_{count} = 2$ secondary variables per node. For primary variables, each node stores $p_{count} = 1$ variable with a maximum of $p_{max} = 10$ values. We limit the computation to a total of $N_{max} = 4$ nodes. We also keep our current implementation of the semi-distributed regression model.

We added a remote procedure call (RPC) to change the necessary variables on a individual node basis. This generates a function within the user interface allowing an individual to send a message to each node updating the variables and also generates the node reception of the message for us, dealing with the variable message length automatically in the existing FOS functionality and calling a message handler we define. On the node side, we implemented the message handler to change all the variables and ensure that the calibration re-runs once we change the matrix definition. All the trickiness in applying this dynamic model revolves around ensuring the details are correctly implemented, especially the indexing; not breaking the existing functionality required carefully outlining the meta-definitions of the timing windows and the computation internal loops traversing the stored columns.

We tested this with our nodes. From a central computer, we modified the matrix at random intervals. Everything worked as desired and now allows us to change the model while it runs on the system. We still have more work before we have a fully dynamic model, which we outline in Chapter 7, but this provides a good start toward achieving our goal.

5.4 Conclusions

Optimizing energy usage on sensor nodes is a key issue for sensor networks. In this chapter, we described a model and distributed algorithms for predicting future average daily solar current. We developed a distributed pseudoinverse algorithm usable in a

wide range of applications. We verified the functionality of these algorithms through simulation and a month-long field experiment on the Fleck platform.

Predicting solar current enables better power management in sensor networks. For example, a power management system could use the solar current prediction to determine that insufficient energy will be harvested over the next two days to support the current operations. A power management planner could use this information to plan the system operation with fewer communication rounds, less data, or perhaps compressed data, trading off power and communication for computation, data resolution, and solution accuracy. Overall, better power management means longer operation of the network, providing more monitoring of the environment, more data, and a more useful sensor network. Our work equips sensor networks to provide better energy management and does so in a very usable, general form. Anyone can use our algorithms to predict future energy on any sensor network with solar recharging. The model utilizes any combination of climactic and spatial variables available on the platform. The only requirement is measurement of solar current.

Finally, we outlined initial steps towards a dynamic model, allowing the user to change the model during system operation. This will lead to a more autonomous and robust method of determining the correct model structure.

Chapter 6

Job Congestion in Networked MFDs

Multi-function devices (MFDs) combine the services of print, fax, and scan (among others) to maximize capabilities while minimizing physical space requirements. Networking MFDs could provide services not available to individual machines, not only by providing distributed computation of jobs, but in allowing access to service capabilities provided by only a subset of the devices. However, in connecting such devices, potential problems arise with higher traffic flows, small numbers of the devices providing a much requested service, and network overhead. One common potential problem is congestion, seen by devices with much requested services or located in bottlenecks of improperly balanced networks, which limits the ability of that device to meet its quality-of-service (QoS) requirements. Detecting congestion, while not difficult, implies that the problem of congestion already exists and all system responses attempt recovery at this point. To avoid reaching a recovery point requires predicting and preventing congestion, which poses a more complicated problem. In this chapter, we use our multiple linear regression model to predict congestion, develop a network simulation to test this model, and evaluate our results, seeing that this approach does succeed in predicting future congestion at a device.

Because these networked MFDs do not yet exist beyond research, Section 6.1 describes the network model we developed to validate our model and provide a frame-

work for testing algorithms and verifying behavior theories. Section 6.2 defines congestion metrics and our prediction model. Section 6.3 validates our congestion prediction model through simulation, outlines our implementation of our algorithms in the network model, and verifies the functionality of our algorithms within the network model environment.

6.1 MFD Network Model

Before we can understand congestion on a MFD network, we need to describe and formalize a MFD network. Multi-function devices exist commercially and are widely used; however, MFD networks with capabilities to share services do not exist other than as a research project. Thus the system itself does not exist, so we also need to develop a model of a MFD network on which to verify our assumptions and develop our prediction algorithms. This section formalizes a MFD network and then develops a model of this network.

6.1.1 MFD Network Description

An MFD provides many different services. If we had a network of such devices, we could consider distributed services, such as parallelizing optical character recognition or distributed storage that would allow a user to enter a document at one device and retrieve it from another device at a later time. For congestion and modeling purposes, we abstract our idea of these networks, ignoring the underlying protocols and even the details of such services, using instead a simple notion of jobs.

All jobs begin when an external user instigates them at a particular device; we assume all devices have external user access. A job contains some number of units of work, ranging from a minimum of 1 to a maximum of θ. The device will service any job consisting of a single unit of work. For any job consisting of more than 1 unit of work, the device will split it into multiple sub-jobs and farm out the sub-jobs to neighboring devices. If the job is farmed out, each neighbor processes the sub-job, returning the results to the initial device. The initial device, once it has all results, post-processes the results and provides the output to the user. Figure 6.1 depicts the overall flow of the model.

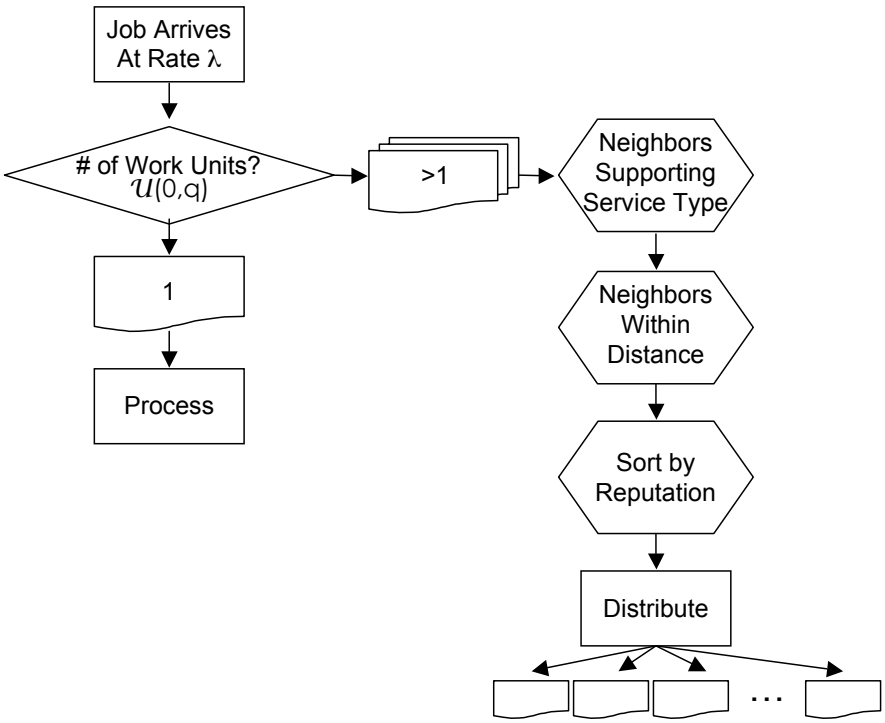

Figure 6.1: Flow Diagram of Model Overview

To help quantify this for modeling, users submit jobs to a device j according to a Poisson arrival process, characterized by parameter λ_j, with jobs either single unit or multi unit with probability α. A multi unit job consists of some number of sub-jobs, randomly determined by a uniform distribution from 2 to θ. Processing single unit jobs requires one step; processing multi unit jobs requires a processing step to divide and farm out the job, processing steps on all devices to complete the sub-jobs, and a processing step to finalize the job. Each processing step takes an amount of time based on a Pareto distribution, characterized by parameters K_j and σ.

Devices distribute sub-jobs based on three behaviors: (1) service capability, location, and reputation. Devices can support any combination of these behaviors, prioritized as introduced. A device supporting all three first defines the sub-set of neighbors that can support the needed services. This sub-set further narrows based on network location. Finally, among the sub-set of equally distant and equally capable neighbors, the device distributes the sub-jobs based on reputation.

In service-based job distribution, a device determines those neighbors capable of processing the job. We define a notion of service types, software applications

performing some set of operations, where each type is independent and self-sufficient. Not every device supports all services, reflecting both an idea of older and newer models in the network, and an idea of purchasing specialized software for only a subset of devices. We assume devices know which services each device in the network can provide. Our network supports some number of services, $\mathcal{S}$, with some percentage, δ, of devices supporting each service. We randomly determine this percentage, δ, based on a uniform distribution between d_{min} and 1. Each device then decides for each service if it is within that percentage of devices.

Location based routing ensures that devices submit sub-jobs to their neighbors based on network hop distance. For our purposes, we assume a device already knows the network topology. Based on this topology, the device generates a routing table defining all neighbors and the shortest hop-count to access each. Devices use the table to determine the closest neighbor capable of processing the job and only send sub-jobs to neighbors within that hop-count. We ignore the path and network costs of routing.

Within their local neighbors, a device partitions a job based on reputation. Reputation reflects the likelihood of computing the sub-job and returning results promptly, which on a real machine would change based on factors such as processor, machine age, machine usage, and machine quality. In our model, we define it based on computation time of sub-jobs, which is determined by the Pareto distribution. If we keep the same parameter, K_j, for every device's Pareto distribution, a device's reputation reflects the random differences between these distributions and device location within the network configuration. To include a notion of machine age and quality, we define a parameter η that adds a random integer between $(0, \eta)$ to K_j, thus "aging" each device (due to the features of the Pareto distribution, a higher η means a younger device). During simulation, each device maintains its own reputation value. After a device determines its set of local neighbors capable of processing the sub-jobs, it then computes a normalized reputation value based on this set of neighbors. The device then sorts the neighbors by normalized reputation and sends a proportional number of sub-jobs to each neighbor.

6.1.2 MFD Network Simulator

With MFD networks defined, we can now discuss our simulator to represent these MFD networks. We need a simulator as sufficiently large networks do not exist in reality and the simulator allows us to test a wide variety of scenarios in order to verify our prediction model.

Our interest in MFD networks lies at the application level, making the safe assumption that such devices will operate using some standard network protocols. Given this, we only want to model this application level, ignoring detailed network implementation issues. This led us to focus on discrete event simulators, which discretize time into unit-less intervals. The simulator processes all tasks occurring within an interval, making the assumption they all can be processed within that interval. Devices cannot communicate within an interval, but send a message to another device with some delay. That device will process the message in the interval in which it arrives. The simulator fast-forwards through intervals with no events; this combined with unit-less intervals allows for simulations running much faster than the real system, the primary benefit of discrete event simulators.

Symbol	**Definition**	**Range**	**Assigned**
N	Number of devices in network	2-22	Runtime
λ_j	Job arrival rate	0.02	Fixed
K_j	Processing scale for Pareto	200	Fixed
α	Percent of multi unit jobs	0-1	Runtime
θ	Max number of sub-jobs in a multi unit job	2-15	Runtime
η	Max machine aging parameter	0-32	Runtime
$\mathcal{S}$	Number of service types	3	Fixed
d_{min}	Min percent supporting each service	0.2	Fixed
δ	Percent of devices supporting service	0.2-1	Generated

Table 6.1: Network Simulator Variables

We chose SimPy, an open-source, Python-based simulator [83]. SimPy provides a rich amount of functionality, especially a large number of functions and hooks for monitoring the internal operations of the simulation. Additionally, it is easy to use and easy to incrementally develop the network simulation.

In SimPy, we model devices as processes containing a job queue. An external user process generates jobs for a device; one external user exists for every device. Both classes operate according to the parameters and distributions we defined above, summarized in Table 6.1. We define the following variables at run time: number of devices within the network (N), arrival rate (λ_j), processing scale (K_j), percent of multi unit jobs (α), maximum number of sub-jobs allowed for a multi unit job (θ), maximum machine aging parameter (η), number of intervals to run the simulation, and network configuration. We fix the other variables describing the network within the simulator to values realistically describing expected networks. Our network supports $\mathcal{S} = 3$ services, which sufficiently covers the set of behaviors within the network while allowing for analysis. The minimum percent of devices supporting each service, d_{min}, is set to 0.2.

To define the parameters, we used measurements from a real test network of 5 MFD devices. With this network, researchers at Xerox computed a job arrival of $\lambda = 0.02$ and a processing time of $K = 200$ and $\sigma = 4$, which, for now, we use as values for all devices. The definition of the Pareto process within SimPy ignores σ and only uses K, so we only supply K in the input parameters. We leave the remaining job parameters, N, α, θ, and η to skew during testing, as we would like to better understand the impact of these parameters on congestion.

For the network configurations, we choose whether to generate a random configuration within the simulator or to use an existing file. If generating a random configuration, the simulator generates a random set of numbers from the maximum number of devices available, ensuring that each device has itself within its list of neighbors for processing sub-jobs. We also supply fixed configurations, using many of the standard network topologies such as rings, stars, and fully connected grids, in addition to randomized configurations (supplied in order to provide repeatability of testing with random configurations) and more realistic ones such as that shown in Figure 6.2 (dotted circles represent subnets within the network and bold lines represent nodes communicating between subnets).

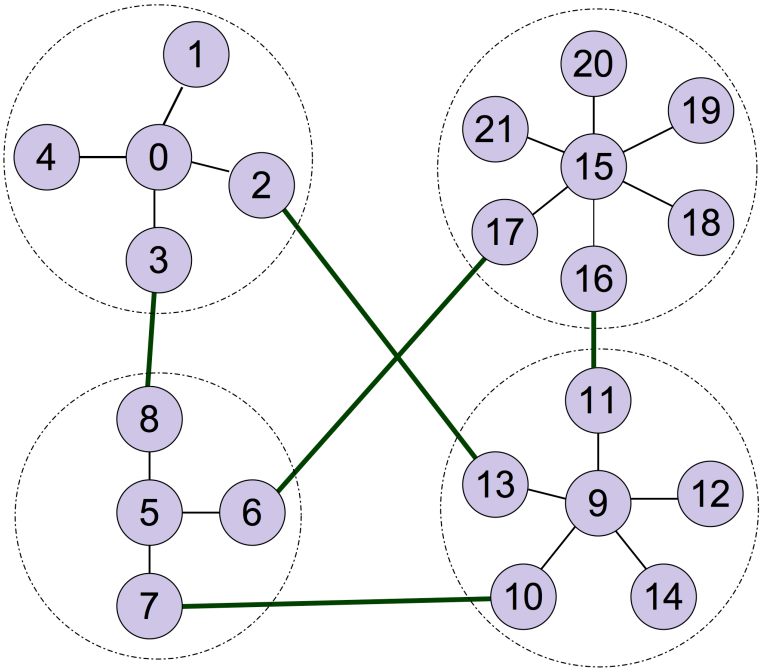

Figure 6.2: "Realistic" Network Configuration

The simulator begins operation by computing the remaining variables. For each service, it computes δ, the percentage of devices supporting that service, randomly subdividing based on a uniform distribution between $d_{min} = 0.2$ and 1. It also computes how many jobs will need each service, again subdividing based on a random uniform distribution. The simulator loads (or generates) the configuration file and defines the neighbor map for each device. After initializing the monitoring and simulation environment, it begins creating and initializing each device.

Each device performs a number of start-up tasks. First, it chooses an ID, given sequentially so no device has the same. It then randomly determines which services it will support, based on a uniform distribution and compared to the previously defined distribution of services. It initializes the monitoring tasks that will record the values of specified variables over the life of the simulation, including total processing time, job queue length, reputation, and number of jobs of each service type. Finally, it generates a routing table using a breadth-first-search. In addition to each MFD device, there is a corresponding external user, which has the same ID and no additional initialization necessary.

After all devices and external users are initialized, the simulation begins running. Figure 6.3 depicts the flow diagram for the external user process. Figure 6.4 outlines the flow diagram representing the actions and messages taken by the multi-function devices, where the external user supplies the jobs.

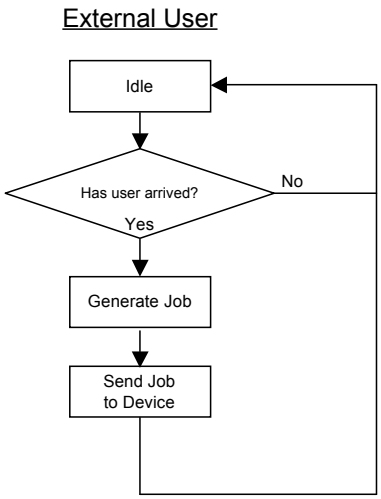

Figure 6.3: Flow Diagram of External User Process Behavior

Each external user generates jobs for its device based on λ. Once a job exists, the external user defines whether it is a single or multi job and the service type. If a multi job, the external user then defines the number of sub-jobs. Finally, the external user places the job in the device job queue and waits until the next job arrives.

A device idles until a job arrives in its queue. Upon receiving a job, it checks whether it is: a single job, a new multi job, a sub-job for another device, or a return response from a neighbor finished with a sub-job for an old multi job. The device processes each for a number of intervals according to the Pareto distribution. If it is a single job, the device first checks if it supports the service type. If it does, the device processes it and returns to idle. Otherwise, it treats it as a new multi job consisting of one sub-job and proceeds with the actions for a new multi job. If it is a new multi job, the device generates the list of neighbors supporting the service type and within the hop count of the closest capable device, ranks the list by reputation, and then sends sub-jobs to all the devices (including itself). The device then finished that processing interval; it will sleep if it assigned itself no sub-jobs (in the case where it does not

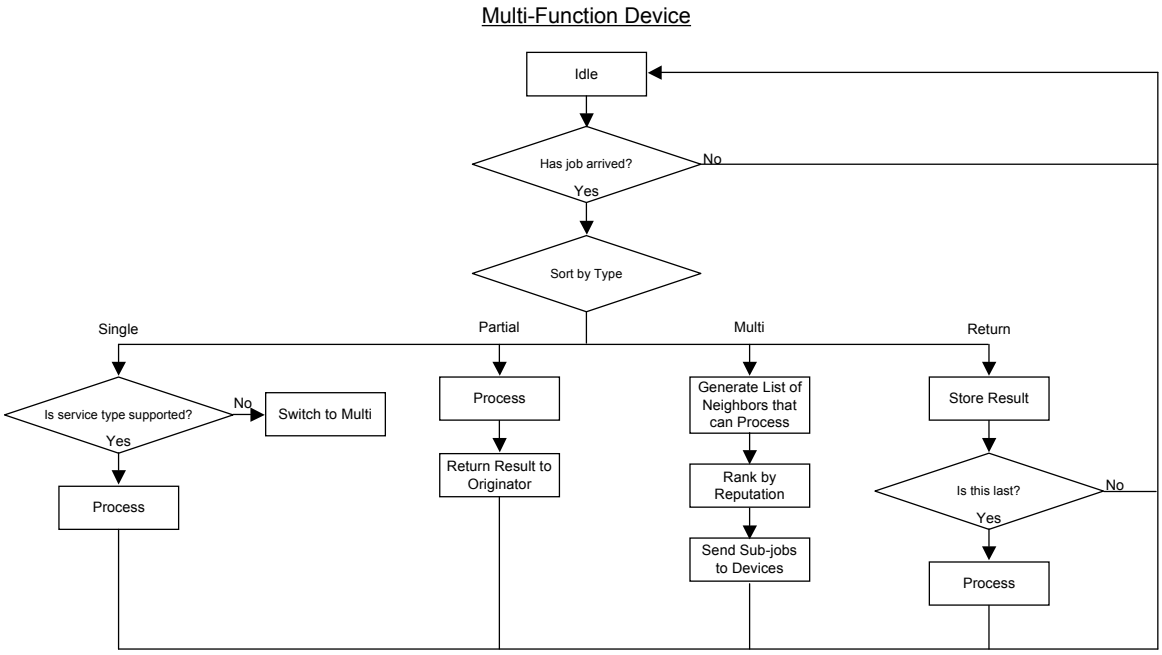

Figure 6.4: Flow Diagram of MFD Process Behavior

support the service type) or will awake in the next interval to process the sub-job. For a sub-job, the device processes it and sends a response to the initiating device once processing completes. Finally, for a return response from a sub-job, the device first marks that portion of the sub-job complete and then checks if it is the last response expected. If it is the last response, the device adds a finalize job to its job queue; this job will require processing intervals according to the Pareto, while the other response steps take no more than the one interval. These behaviors describe all the actions taken by each device and devices only interact through the job queue, mimicking the notion of messages through a network. Overall this provides an accurate simulation of the expected network behavior.

6.2 Prediction Model

In this section, we define congestion for a MFD network and describe a congestion prediction model.

6.2.1 Congestion Definition

We need to define what congestion means for a MFD network. As we ignore network considerations, we are really focusing on what congestion means for a multi-function

device. Within a single device, congestion is the inability to perform a unit of work within a specified time, T_C. A device quantifies this as the wait time of a job in its processing queue, which it can simplify by measuring its queue length. Because congestion occurs over a time window and we do not want to capture momentary fluctuations in queue length, we average the queue length over time increments, T_{inc}. Once average queue length exceeds Q_c, comparable to T_C, the device decides it is congested and performs some recovery behavior. To predict congestion, a device models congestion some time units into the future, T_L, and detects a predicted level over Q_c.

6.2.2 MLR Congestion Prediction Model

As the primary function of the device network is to provide a QoS guarantee, we need a prediction model with limited computation and communication requirements. This insures minimal contention for the device processor and network resources. Additionally, we would like a model that can self-calibrate to handle the variability of the network usage and topology. To achieve all this, we use our multiple linear regression model from Chapter 3. These models can easily scale the number of variables used in prediction, re-calibrate, and, after calibration, require limited computation to predict the output.

We apply this model to predict congestion for each device within the network. In only modeling congestion for a single device in the network, our theory is that local detection and correction can alleviate global congestion. To predict congestion for a device, we use the average queue length, Q, which is a time series variable as necessary for our model. We also can include reputation and number of jobs of each service type, all variables locally measurable by the device.

Algorithm 6.1 displays the basic structure of this model. In it, Q is the past average queue length, N is the number of past values used within the equation (or the order), Y is the predicted future average queue length, and Q_c is the threshold for queue length leading to congestion.

Algorithm 6.1 Congestion Prediction Algorithm

Q : average queue length
N : # past queue length values used
Y : predicted queue length
T_T : training time window
T_L : prediction lead time
T_R : recalibration time window
▷ Compute initial coefficients and prediction
$Q \leftarrow [Q(1 : T_L - N), ..., Q(1 + N : T_L)]$
$X = ((Q * Q^T)^{-1} * Q^T) * Y(1 + T_L : T_L)$
$Y(1 + T_L : T_L) = Q * X$

for $t = T_T + 1$ to ... **do** ▷ Forecast
    if $(t\%T_R) == 0$ **then**
        $Q \leftarrow [Q(t - T_L : t - N), ..., Q(t - T_L + N : t)]$ ▷ Recalibrate coefficients
        $X = ((Q * Q^T)^{-1} * Q^T) * Y(t - T_T : t)$
    end if
    $Q \leftarrow [Q(t - N), ..., Q(t)]$
    $Y(t + T_L) = Q * X$
    if $Y(t + T_L) > Q_c$ **then**
        Congestion : *Avoidance actions necessary*
    end if
end for

6.3 Testing: Simulation and Implementation

We now verify the simulator in SimPy, validate the model in Matlab, and implement the model in the simulator.

6.3.1 Network Simulator

In this section, we validate the operation of the simulator and demonstrate the effects of several parameters.

We would like to understand the behavior of the MFD network through the simulator. The simulator should reflect enough real world operations to provide a valid testing environment, so we verify the operation of our simulator through a variety of configuration options and tests.

We added each behavior sequentially, ensuring the correct operation through hand-analysis of small tests with fixed configurations. This guaranteed that, as we developed the model, each new parameter and behavior was implemented as we specified.

Next, we generated a large number of configurations and tests. We decided to explore a range in the number of devices comprising the network. With no real world data except an idea of common non-networked MFD office configurations, we examined networks of 2, 7, 12, 17, and 22. This provided a reasonable number of devices for small to medium offices and remained manageable for analysis. For this range of network sizes, we developed a set of common fixed network topologies: ring (both limiting connections to right-hand neighbor and allowing connections to both neighbors), star, grid, and line. Assuming many offices do not actually setup their copiers in these configurations, we also generated random and hybrid configurations. Random configurations determine a set of neighbors for each device; to ensure repeatability, we pre-generate these random configurations and use them for all testing scenarios. Hybrid configurations randomly connect a set of star configurations (see Figure 6.2). These configurations are parameterized by minimum number of stars, maximum number of stars, minimum number of nodes per star, maximum number of nodes per star, and number of connections between stars. To generate, we randomly determine how many stars. The first star begins with node 0 as the hub and a random number of devices with sequential IDs, of which some number connect to nodes outside the star. The remaining stars repeat the process, using the next ID as the hub center. In choosing the connections between stars, we randomly choose the outside nodes, but also ensure the generated network is connected.

For each configuration, we generate a set of tests. We tested job ratios of $\alpha = 0.2$, 0.4, 0.6, and 0.8, and job sizes of $\theta = 5$, 10, and 15. The machine aging parameter, η, stayed fixed at 10 for these tests. The simulation length we also fixed at 5400; if we consider each interval equal to a second, the simulator runs for 1.5 hours, which provides a good snapshot of behavior for analysis. With all configurations and parameter variations, we have 611 tests.

As output of these tests, every interval we record several monitored variables: job queue length, number of single unit jobs, number of sub-jobs from other devices, number of multi unit jobs, reputation, number of service type 1 jobs in queue, number

Variables of Interest	Average Value
Average Job Queue Length	4.89
Number of Single Jobs	86.41
Number of Multi Jobs	49.22
Number of Partial Jobs	530.26
Number of Service 1 Type Jobs	5.69
Number of Service 2 Type Jobs	4.38
Number of Service 3 Type Jobs	6.21

Table 6.2: Average Behavior of Device 0 for All Tests

of service type 2 jobs in queue, and number of service type 3 jobs in queue. These reflect the primary variables of interest and those changing the most throughout the simulation. We log these variables for each device in each test, generating a large amount of data describing the various behaviors of the network.

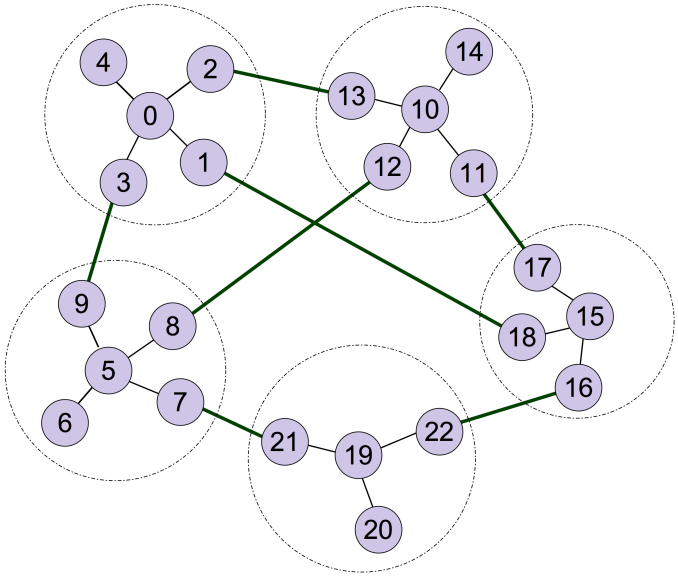

Figure 6.5: Hybrid Configuration Used in Parameter Analysis

Table 6.2 shows the average behavior of Device 0 in all tests. The average across the tests of the average job queue length indicates that congestion does occur and we have room to improve on the system behaviors. As we expect, the number of single and multi jobs are within the same order of magnitude, while partial jobs are an order of magnitude greater reflecting their creation by sub-dividing multi jobs. Jobs split about equally among the 3 service types, which reflects our expectations of this category and definition within the simulator.

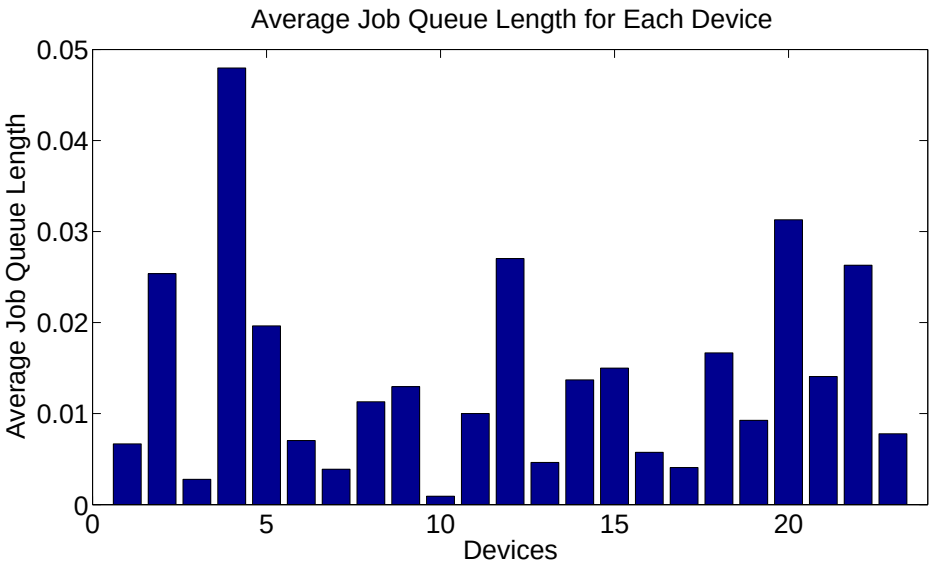

Figure 6.6: Average Job Queue Length

We choose one configuration (see Figure 6.5) and explore the behavior of devices in greater detail. Figure 6.6 shows the average job queue length for each device. No patterns exist in distribution of the average job queue. For example, the device with the greatest length for each sub-star is a connecting one for most but the hub device for one, the device with the shortest length for each sub-star is a connecting device but all connect to different sub-stars not to a specific sub-star, the sub-star with the device seeing the longest queue length also contains the device seeing the second smallest, and the list of conflicting patterns continues. This indicates that we cannot devise a congestion model based on the network topology, but need a model that learns the various non-intuitive behaviors occurring.

To understand the breakdown of the jobs, we examine Figures 6.7(a), 6.7(b), and 6.7(c). Here we see the partial jobs dominate the behavior with more jobs in general and a clear mimicry of the average job queue length across devices. This makes sense. Partial jobs exist as the sub-division of multi jobs, which the simulator generates relative to single jobs. We expect to see roughly the same order of magnitude then for multi and single jobs, but a larger number of partial jobs. This larger number of jobs influences the overall behavior of the system and the average job queue length seen by devices.

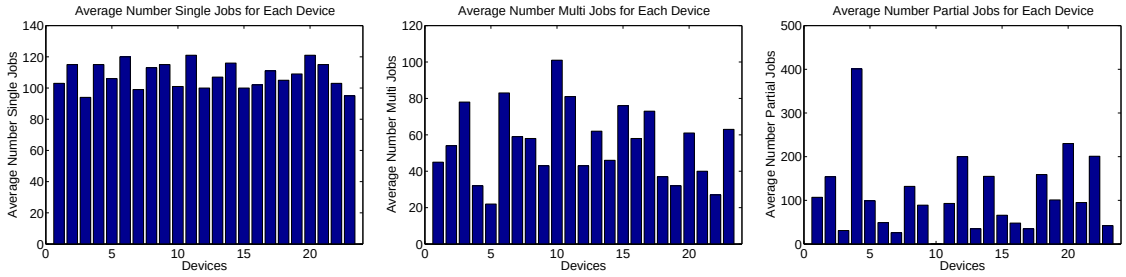

Figure 6.7: Average Number of Single, Multi, and Partial Jobs

We also examine the queue behavior over time as shown in Figure 6.8. This reflects the behaviors seen in the average job queue length, Figure 6.6. Additionally, we see no job balancing across devices. Some devices, such as 2 and 9, have almost no jobs while other devices, such as 3, 13, and 19, are quite busy with jobs throughout the time window. Controls based on job congestion can alleviate some of this, enabling a more balanced network.

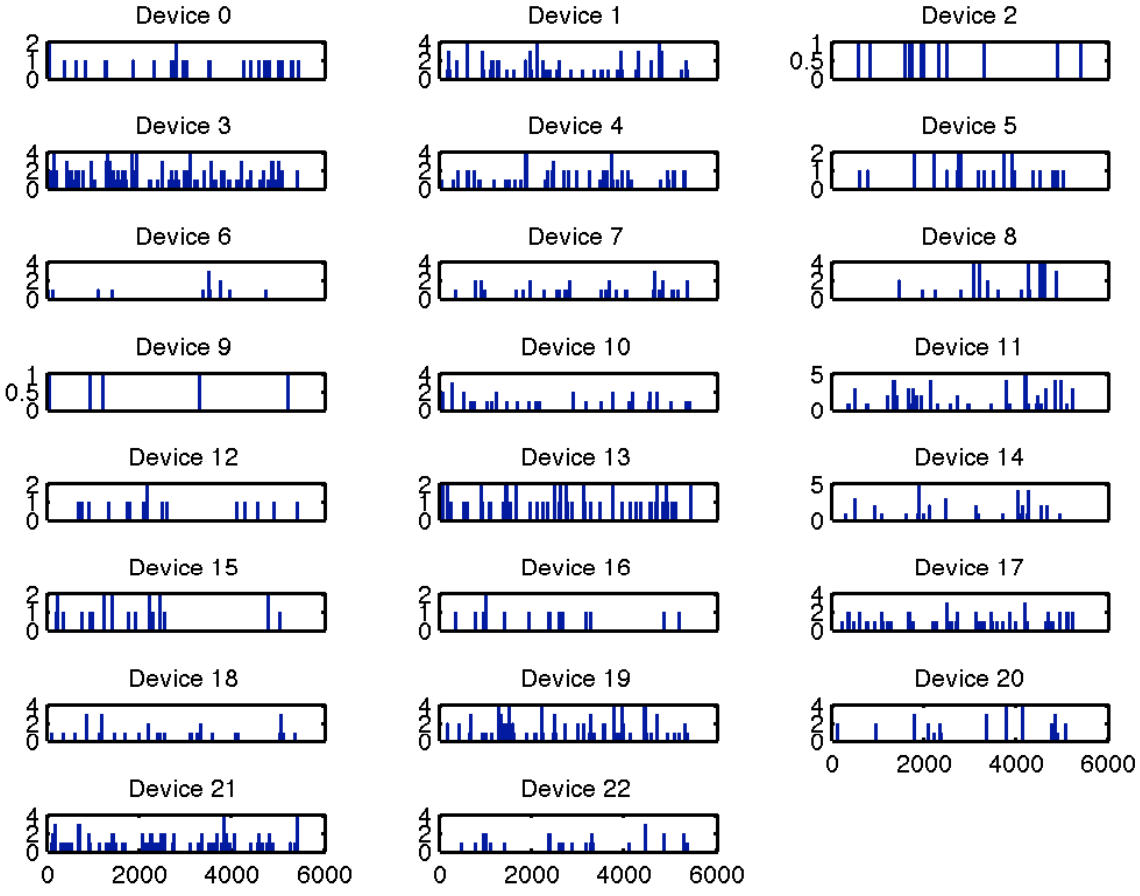

Figure 6.8: Job Queue Length Over Time for Each Device

Additionally, we defined several variables for which we do not have real world data to fix and would like to understand the effects of these variables, namely α, θ, and η. With the configuration defined above, we examine the effect of each parameter on job queue length, job breakdown between single and multi, and reputation.

Effect of α

To examine the effect of α, which defines the percentage of multi jobs generated, we test the network with the hybrid configuration of Figure 6.5 and $\theta = 5$. We nullify the effect of reputation by giving all devices a reputation of 1. We then skew α over $(0, 1)$ in increments of 0.1.

After running the 11 tests, we see an average job queue length across all devices as shown in Figure 6.9. As α increase, job queue length increases. This reflects the fact that an increase in α increases the number of multi jobs generated, which requires more processing that single jobs. This processing not only encompasses the sub-job processed by the devices, but the initial processing job and the final processing job.

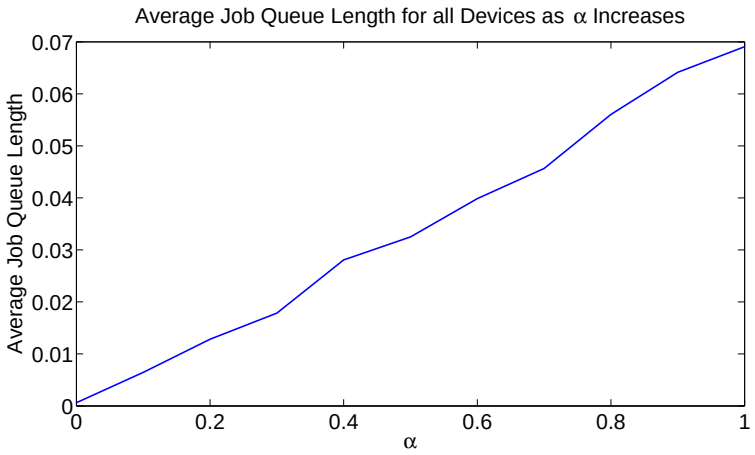

Figure 6.9: Average Job Queue Length as α Increases

We can best see this through examining the job breakdown in Figure 6.10. Single jobs linearly decrease as α increases and multi jobs linearly increase, matching the definition of α. Multi jobs do not start at 0, however, which reflects the service support. As a device receives a job of a service type it does not support, it redefines it as a multi job consisting of one sub-job which it then sends to a neighboring device

capable of supporting that job. This allows the device to use the same set of messages and processing framework for jobs it cannot support as with multi jobs, easing the processing and message requirements of the simulator (and probably of the physical devices as well). This also explains the non-zero, yet equal to multi jobs, number of partial jobs at $\alpha = 0$. Partial jobs also increase linearly, yet at a greater rate than multi due to the multiplicative factor of the relationship between partial and multi; every multi job generates 1 to 5 partial jobs.

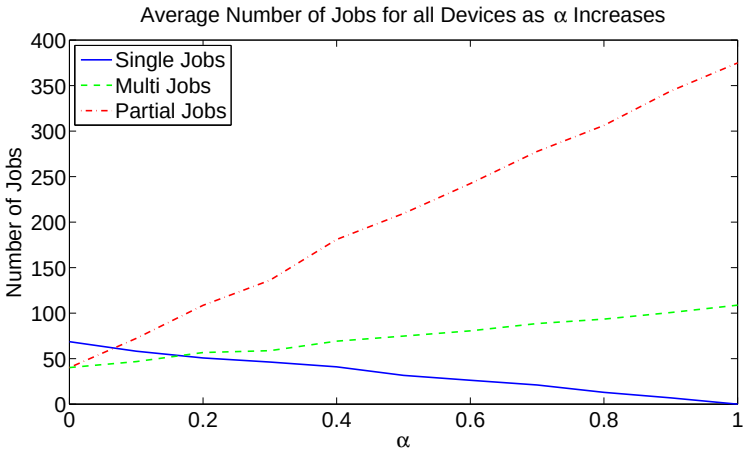

Figure 6.10: Average Number of Jobs as α Increases

Overall, the more interesting behaviors from a congestion standpoint occur after the number of multi jobs increases past the number of single jobs at approximately $\alpha = 0.2$. Yet increasing α to 1 will not reflect what we know of system behavior so we pick a value of 0.4 that best reflects both reality and interesting behaviors.

Effect of θ

We next explore the effect of θ, which defines the maximum number of sub-jobs allowed for a multi job. Again, we test the network with the hybrid configuration of Figure 6.5, $\alpha = 0.2$, and nullifying the effect of reputation by giving all devices a reputation of 1. We skew θ over $(2, 15)$ in increments of 1 (our definition of multi jobs does not allow for external users initiating jobs of only 1; these jobs can only be generated from devices as part of the service support strategy).

Figure 6.11 shows the average job queue length as θ increases. The job queue length mostly increases linearly, except for a slight down tick at $\theta = 15$. Looking at

a larger values of θ results in a continued linear increase of θ; this small decrease is most likely due to the randomness within the simulator for all of the variables.

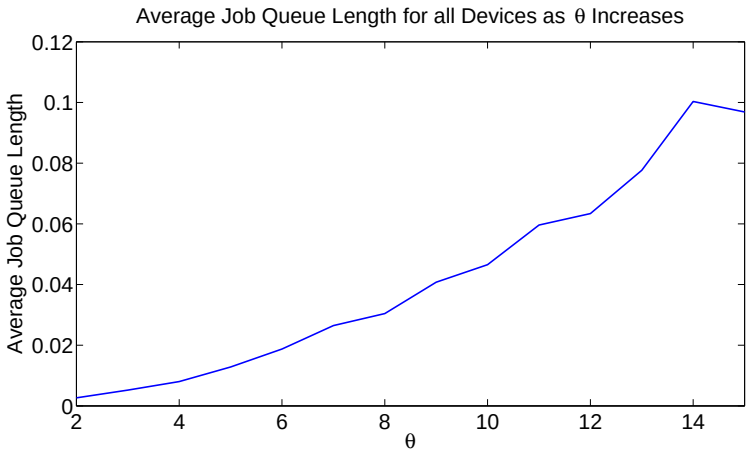

Figure 6.11: Average Job Queue Length as θ Increases

Figure 6.12 outlines the job breakdown. As we expect with no variation in α, single and multi jobs hold steady throughout with minor variations due to randomness. Partial jobs increase linearly with a slight down tick, matching the average job queue length, which comes as no surprise since partial jobs dominate.

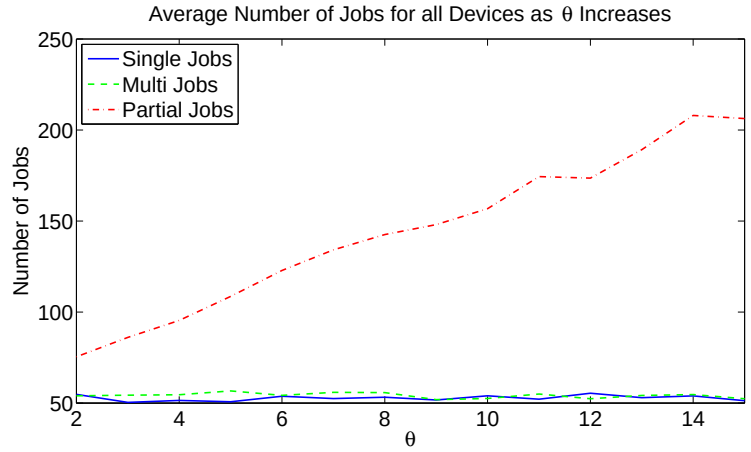

Figure 6.12: Average Number of Jobs as θ Increases

With these results, we see that our choice of θ will directly affect the number of partial jobs generated. We then want to choose a reasonable value that generates enough jobs to provide interesting behaviors so we pick $\theta = 10$ for our congestion tests.

Effect of η

Our final parameter to examine is η which reflects the aging of devices and affects the reputation of devices. A characteristic of this parameter is that as η increases, the K parameter of the Pareto distribution grows, which decreases the variance, decreases the range of processing times, and effectively youthens the device. Thus, the device with the lowest η is the oldest model and the device with the highest η is the newest.

We test the network with the hybrid configuration of Figure 6.5, $\alpha = 0.2$, and $\theta = 5$. To attempt to isolate the effect of η, we confine the aging to Device 0. For Device 0, we skew η over $(2, 32)$ in increments of 2.

We first examine the average queue length for all devices in Figure 6.13 and the average number of jobs for all devices in Figure 6.14. The graphs have three discrete behavior sets from $\eta = (2, 20)$, $\eta = (22, 26)$, and $\eta = (28, 32)$. Both show rather odd behaviors as they increase for $\eta = (22, 26)$, but then decrease for $\eta = (28, 32)$. We can understand both the discretization and odd behavior patterns by closer examination of Device 0.

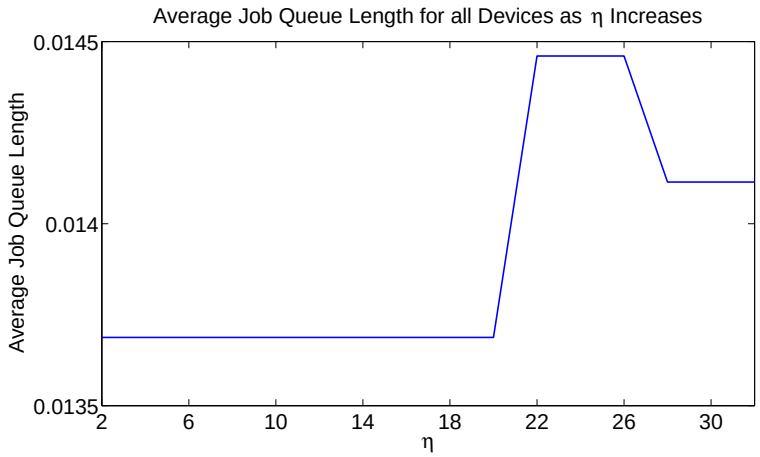

Figure 6.13: Average Job Queue Length as η Increases

As Figure 6.15 demonstrates, the average job queue length of Device 0 dominates the average over all devices of the average job queue length (seen in 6.13), which makes sense as the other devices have the same parameters for all tests and, therefore, should have the same behavior for all tests. The average reputation directly inverts this pattern, as shown in Figure 6.16.

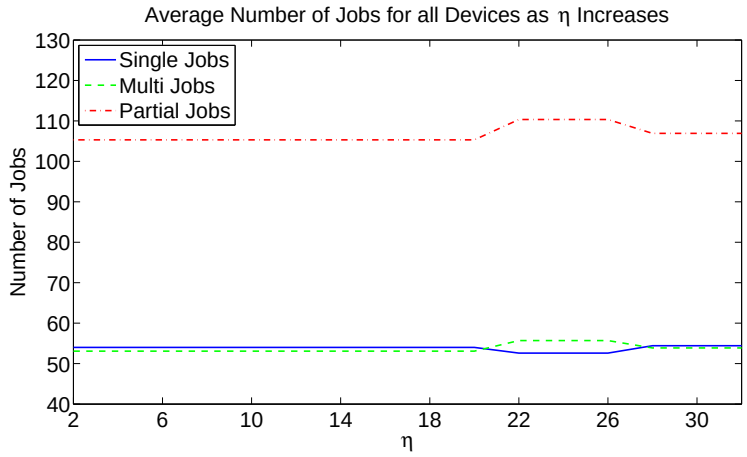

Figure 6.14: Average Number of Jobs as η Increases

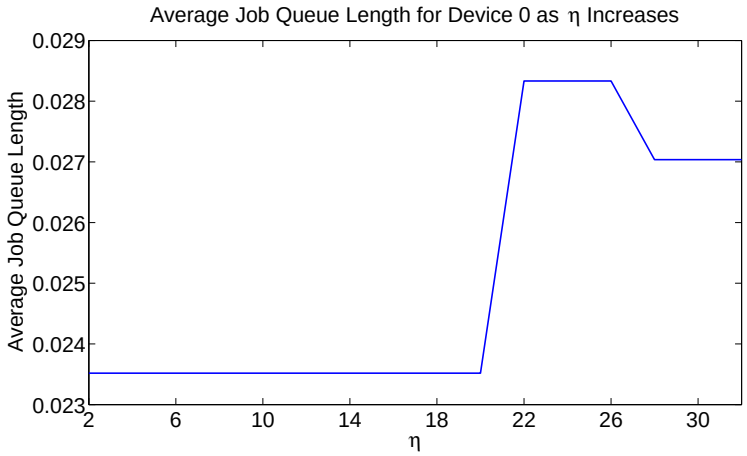

Figure 6.15: Average Job Queue Length of Device 0 as η Increases

The overall discretization of the behavior we can attribute to the processing distribution and random effects. Modifying η modifies the processing time, which we model as a Pareto, using the internal Python distribution. Because this is a random variable, our modification changes the variance, but not the mean. This leads to changes in the processing time, but not necessarily dramatic changes. Gradual changes in processing time will affect reputation, which will modify the number of jobs sent to Device 0. This also is a gradual change and the combination of the two leads to the discretization we see in the figures.

Looking closer at the transition between $\eta = (2, 20)$ and $\eta = (22, 26)$, we see these effects in greater detail. Figure 6.17 shows the reputation over time for Device 0 when $\eta = 0$ and $\eta = 24$. As reputation relates to processing time, a lower reputation is

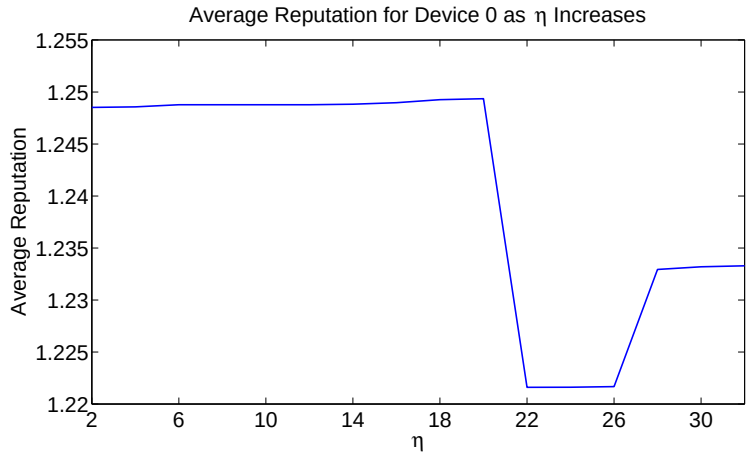

Figure 6.16: Average Reputation of Device 0 as η Increases

better. In the graph, we see the reputations match initially, then diverge once the processing effects begin to multiply, and remains roughly lower for $\eta = 24$ over the remainder of the simulation. This decrease reflects the decrease in processing time caused by the increase in η, explaining the decrease in average reputation.

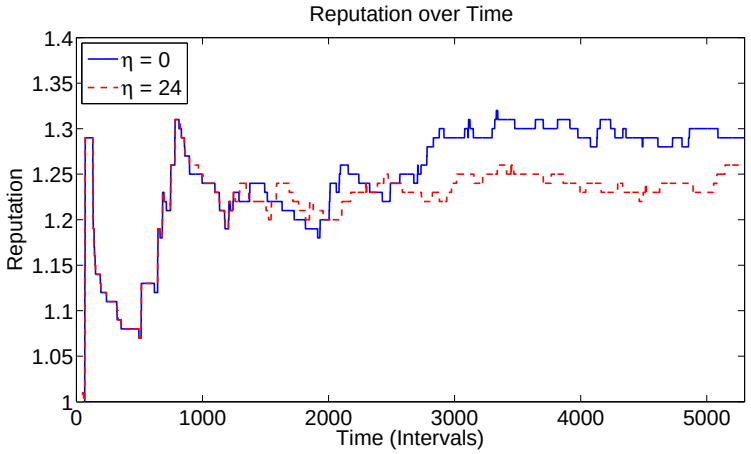

Figure 6.17: Reputation of Device 0 for $\eta = 0$ and $\eta = 24$

To understand the specifics of the increase in queue length, we need to examine the number of jobs over time. Figures 6.18, 6.19, and 6.20 display, respectively, the number of single, multi, and partial jobs for $\eta = 0$ and $\eta = 24$ over the time of the test. Both single and multi jobs relate to the external user and creation of them relies on the exponential arrival process, not the Pareto. However, the external user wakes the device when creating jobs so a faster processing time leads to quicker emptying

of the queue, which is the time when the simulator counts the number of jobs (at processing not at creation). This means the number of jobs can increase slightly as the device processes the queue faster, which we see in these graphs as η increases from 0 to 24. Partial jobs, on the other hand, directly relate to reputation and processing time. As the reputation of Device 0 decreases, the other devices become more likely to send their sub-jobs to Device 0, leading to the constant increase in number of partial jobs. Receiving more jobs increases the average job queue length for Device 0, explaining the increases seen in both the average job queue graph for Device 0 and the average job queue graph for all devices (Figure 6.15 and Figure 6.13).

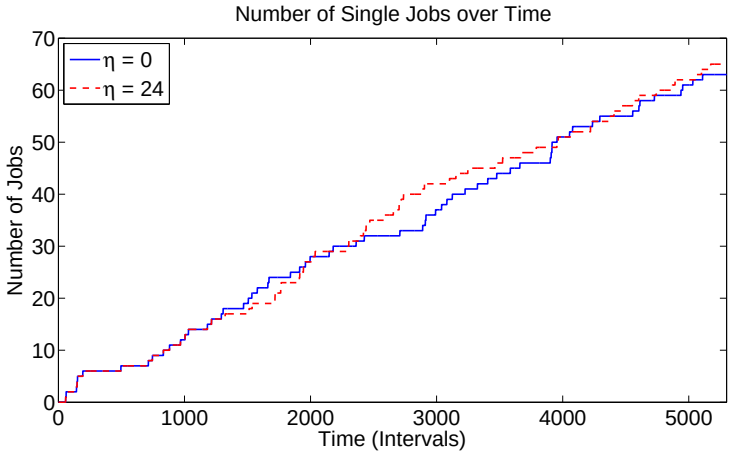

Figure 6.18: Number of Single Jobs for Device 0 for $\eta = 0$ and $\eta = 24$

The decrease in job queue length at $\eta = (28, 32)$ shows a point where devices send too many jobs to Device 0, increasing the processing time and increasing the reputation. Possibly we would see it decrease if the simulation ran longer, but most likely this reflects a transition point where the faster devices get swamped because they are faster. This argues that congestion does occur and we need job balancing to avoid it.

Understanding the transitions in the average job numbers of Figure 6.14 requires considering the global system behavior in response to the changes in Device 0's behavior. As devices send more partial jobs to Device 0, Device 0's queue lengthens and the response time of Device 0 decreases. These devices then see a slowdown in their ability to process jobs as they wait to complete the multi jobs pending the

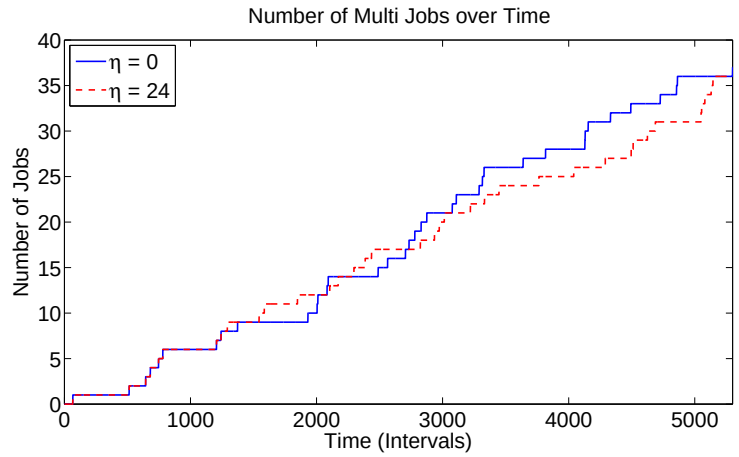

Figure 6.19: Number of Multi Jobs for Device 0 for $\eta = 0$ and $\eta = 24$

response of Device 0. This affects the reputation of those devices in turn affecting how other devices distribute their jobs. Ultimately, the systems sees a trickle effect, changing the job distribution, job queue length, reputations, and job creation. This latter results from jobs not being processed at the same times and the reliance on random variables for all parts; a shift in processing times also affects the order of random variables generated by external users. The overall effect of these changes causes unpredictable behaviors such as those we see in the average job numbers.

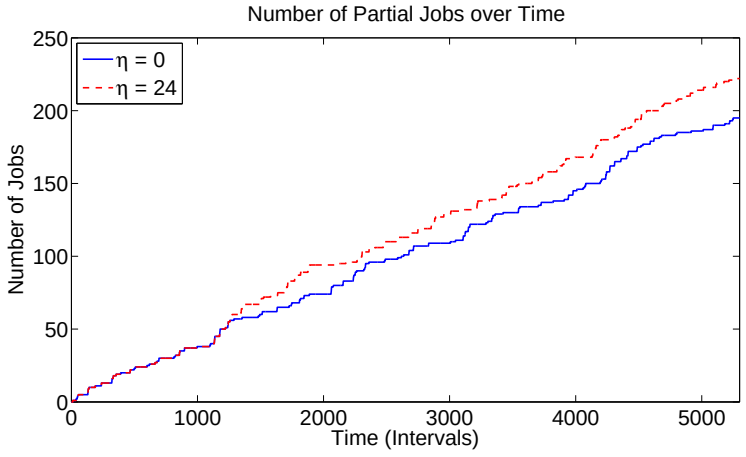

Figure 6.20: Number of Partial Jobs for Device 0 for $\eta = 0$ and $\eta = 24$

Summary

We verified the simulator's operation of our MFD network model and explored the effects of α, θ, and η. Moving forward we define a set of parameters to run all configurations through for congestion simulation. Based on our analysis, we choose $\alpha = 0.4$, $\theta = 10$, and $\eta = 24$ with randomizing from $(0, 24)$. This will provide enough variation to ensure congestion occurs while mimicking reasonable network behavior.

6.3.2 Congestion Simulation

We want to verify the predictability of congestion by regression models. To start, we outline our test setup, followed by our model calibration process, and finally our results.

Test Data and Setup

To test this model, we use the network simulator monitoring results for all network configurations with $\alpha = 0.4$, $\theta = 10$, and $\eta = 24$. By fixing the parameters, we reduce our test number from 612 to 50, still a reasonably large number of tests. Table 6.3 outlines the network topology and number of devices for each test number.

For each device, the network simulator records queue length, number of jobs of each type (single, multi, or partial), reputation, and the number of jobs for each of the 3 service types. We process this data, averaging all variables over time intervals of 1, 5, and 10. Averaging smooths out potential non-linear behaviors, highlighting the key congestion areas. This processing generates a time series record of the behavior of each device and provides the necessary input for the congestion prediction model.

The model predicts future job queue length for Device 0, using the data gathered by the simulator monitor as inputs for Device 0's prediction. Our testing computes predictions for time periods of 1, 5, and 10 intervals in the future. As a comparison to our model, we also compute predictions for these intervals using a naive approach of persistence (or assuming that the queue length will stay at its currently observed value).

We define one measurement criteria for determining the quality of our algorithms: the root mean square error (RMSE). For this metric, the lower the value the better

Network Topology	Number of Devices					
	1	**2**	**7**	**12**	**17**	**22**
Random			1	2	3	4
Single	5					
Line		6	7	8	9	10
Ring: One Direction		11	12	13	14	15
Ring: Both Directions		16	17	18	19	20
Star		21	22	23	24	25
Grid		26	27	28	29	30
Network Topology	**Number of Devices**					**Test Number**
Hybrid 0	13					31
Hybrid 1	15					32
Hybrid 2	23					33
Hybrid 3	23					34
Hybrid 4	21					35
Hybrid 5	23					36
Hybrid 6	31					37
Hybrid 7	18					38
Hybrid 8	16					39
Hybrid 9	15					40
Hybrid 10	22					41
Hybrid 11	25					42
Hybrid 12	21					43
Hybrid 13	20					44
Hybrid 14	13					45
Hybrid 15	19					46
Hybrid 16	23					47
Hybrid 17	16					48
Hybrid 18	19					49
Hybrid 19	13					50

Table 6.3: Test Number for Each Network Topology and Number of Devices

the prediction. This allows for a comparison of the observed queue length time series to our predicted time series and reflects the quality of our overall prediction.

Model Calibration

We implemented the model as described in Section 6.2 in Matlab, starting with defining the training window and recalibration. We define a training window, T_T, equal to 0.1 of the total intervals for the average queue length data. Either we do not recalibrate the coefficients after the initial training or we recalibrate after we observe a new training time window half the length of the original, or $T_T/2$.

Given these two parameter definitions, we analyzed the remaining parameters describing the models to determine optimal values. To pick the best values, we sweep the order (the number of past values used) for the average queue length as well as other input variables with and without including the error of the past prediction. We find the optimal configuration for our metric; the one with the lowest root mean square error.

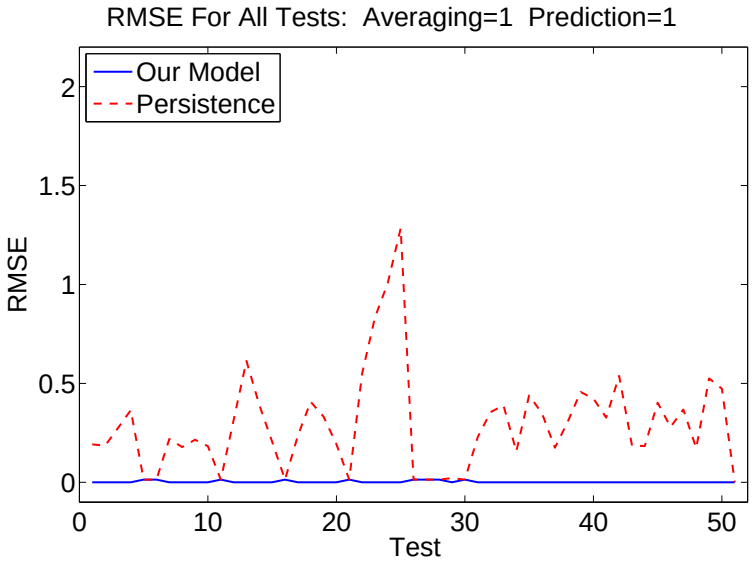

Figure 6.21: Lowest RMSE for Averaging=1 and Prediction=1

Model Results

We compute the results for all tests. As we have 3 averaging parameters and 3 prediction values, we have 9 different sets of results. For each set, we plot the optimal configurations for each test and lowest RMSE seen for each test.

Figures 6.21 and 6.22 display the results for averaging every interval and predicting 1 interval in advance. We plot both the results for our model and the results for persistence in Figure 6.21 outlining the RMSE achieved for each test. In this case, our model clearly performs better than persistence for the majority of tests with persistence providing the best results for a few tests. To understand the test configurations, we examine Figure 6.22, which displays the configuration of parameters providing the

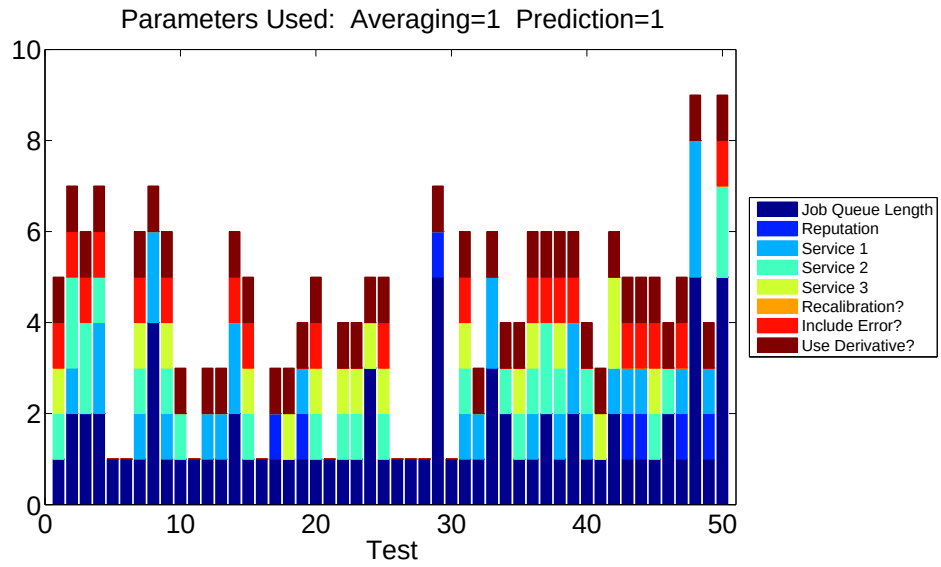

Figure 6.22: Optimal Configurations for Averaging=1 and Prediction=1

lowest RMSE for each test. The configuration options consist of: job queue length, reputation, number of service 1 jobs, number of service 2 jobs, number of service 3 jobs, if we recalibrate, if we include the prediction error seen so far, and if we use the derivative of the job queue length data (giving the rate of change of that variable which can indicate future trends). For each test, we indicate which of these options were used through color bars, where lack of a variable's color indicates that variable is not used in the configuration. The time series variables also allow the model to use more past time values ($t, t-1, t-2$, etc.), which the graph indicates by larger blocks of that color. For example, Test 2 observes the best result using the past 2 measurements of the job queue length, 1 past measurement of the number of service 1 jobs, 2 past measurements of the number of service 2 jobs, the error of the prediction for the last observed job queue length, and the last measurement of the derivative of the job queue length. We can see in this graph that, while the majority require complicated configurations provided by our model, 9 tests had the best results using the persistence configuration of only the past measurement of the job queue.

Figures 6.23(a) through 6.25(f) display the two different figures for the remaining 8 configurations. By construction, our model always has the lowest RMSE although in a couple cases our model configuration matches persistence. These cases always account for less than 25% of all cases. As we rarely construct or maintain fixed net-

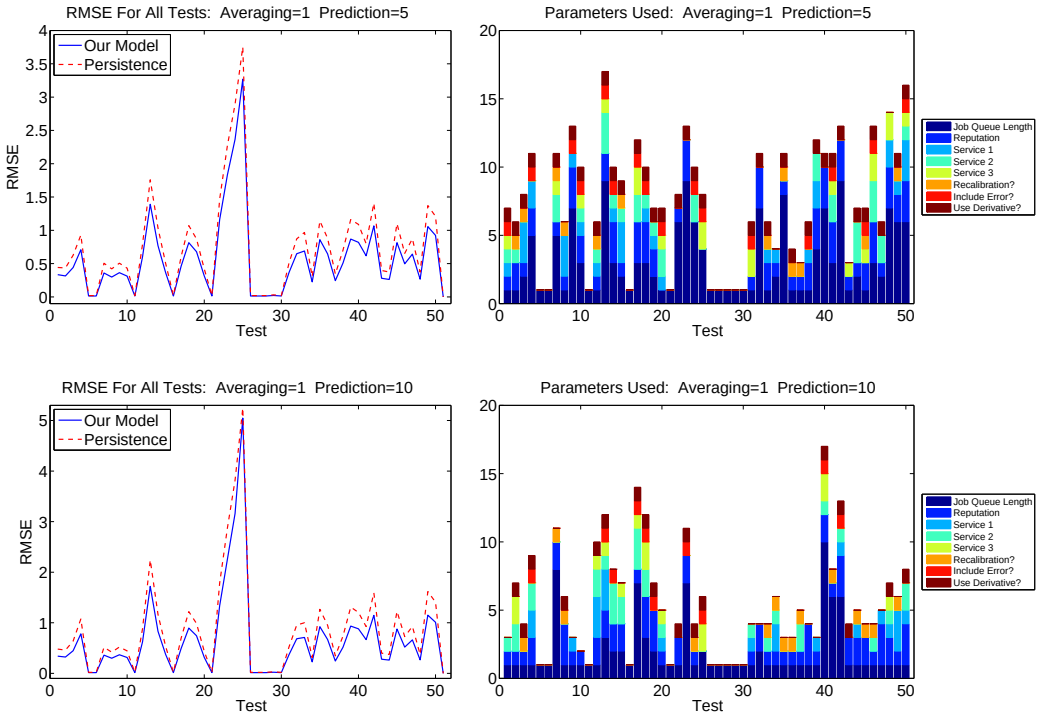

Figure 6.23: Lowest RMSE and Optimal Configurations for Averaging=1 and Prediction={5,10}

work configurations, this argues that our model can provide good predictions despite changes to the network and in spite of the network parameters regarding the number of devices, connections between devices, services supported, distribution of services supported, and many other considerations.

Examining the figures for insights, we see that our model performs best for the shortest prediction interval. This is unique in our experience as usually persistence can capture the closest prediction windows best. In these shorter prediction windows, though, the nonlinearities of the system emerge with job queue length changing erratically. Our regression model captures this better because it includes so many other variables, which we can see by examining the parameter breakdown for the tests. In all three averaging cases, the prediction equals 1 scenario utilizes the full spectrum of variables, always including the derivative in models performing better than persistence and often using the prediction error as well. As the prediction window increases, we see a transition to more job queue length measurements included in the model. When the prediction equals 5, most models include more than 5 past job queue length measurements with many including up to 10. Increasing the prediction to 10 shows

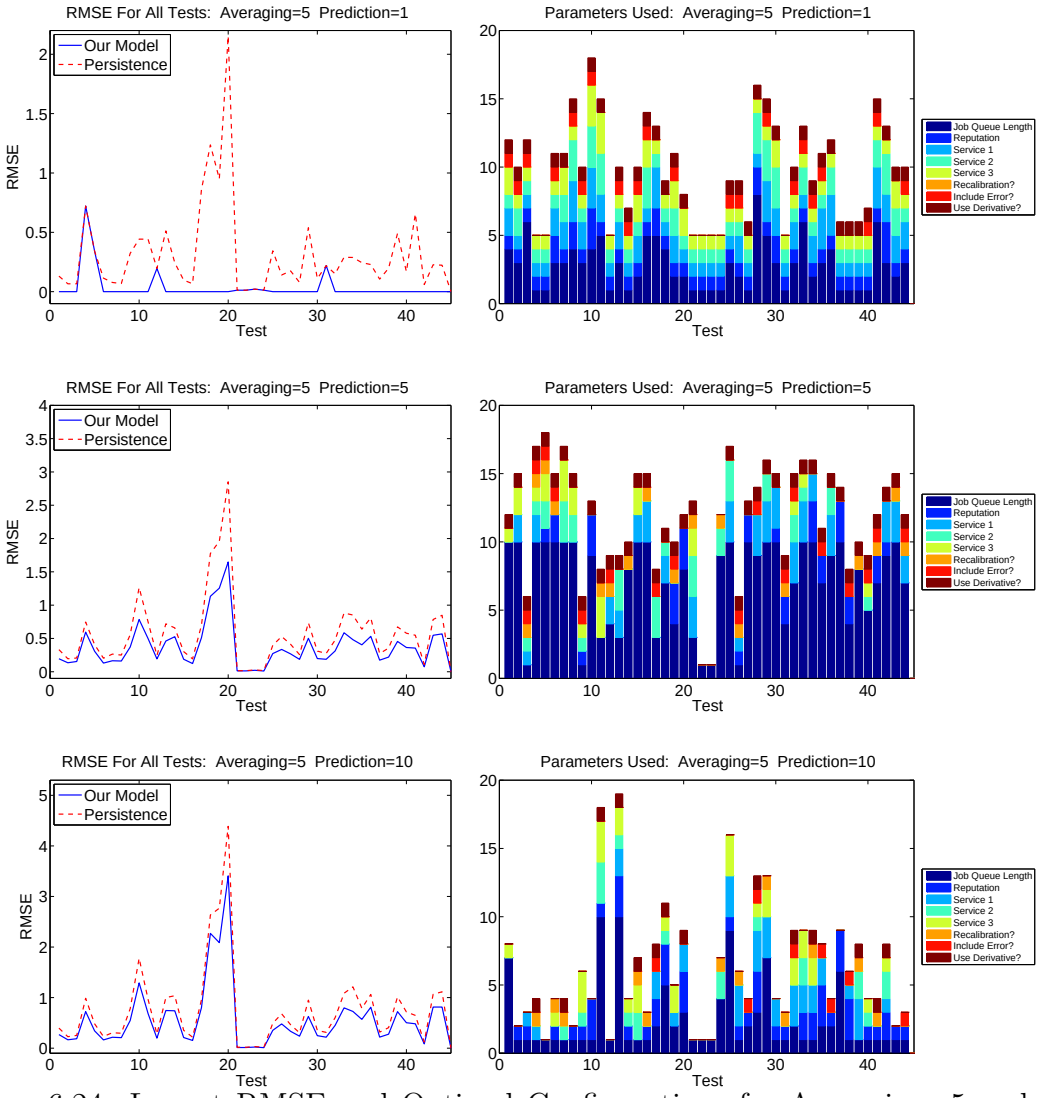

Figure 6.24: Lowest RMSE and Optimal Configurations for Averaging=5 and Prediction={1,5,10}

a slight decrease in this trend with a slight increase in the reliance on the service type parameters. Overall, this explains why our model improves on persistence and reinforces our need for dynamic forms of multiple linear regression models.

6.3.3 Congestion Prediction on Network Simulation

We implement our congestion prediction in the MFD network simulator. Doing so requires using the NumPy [67] package as Python does not support matrix operations. Other than that change, we only need to add the prediction and calibration algorithms.

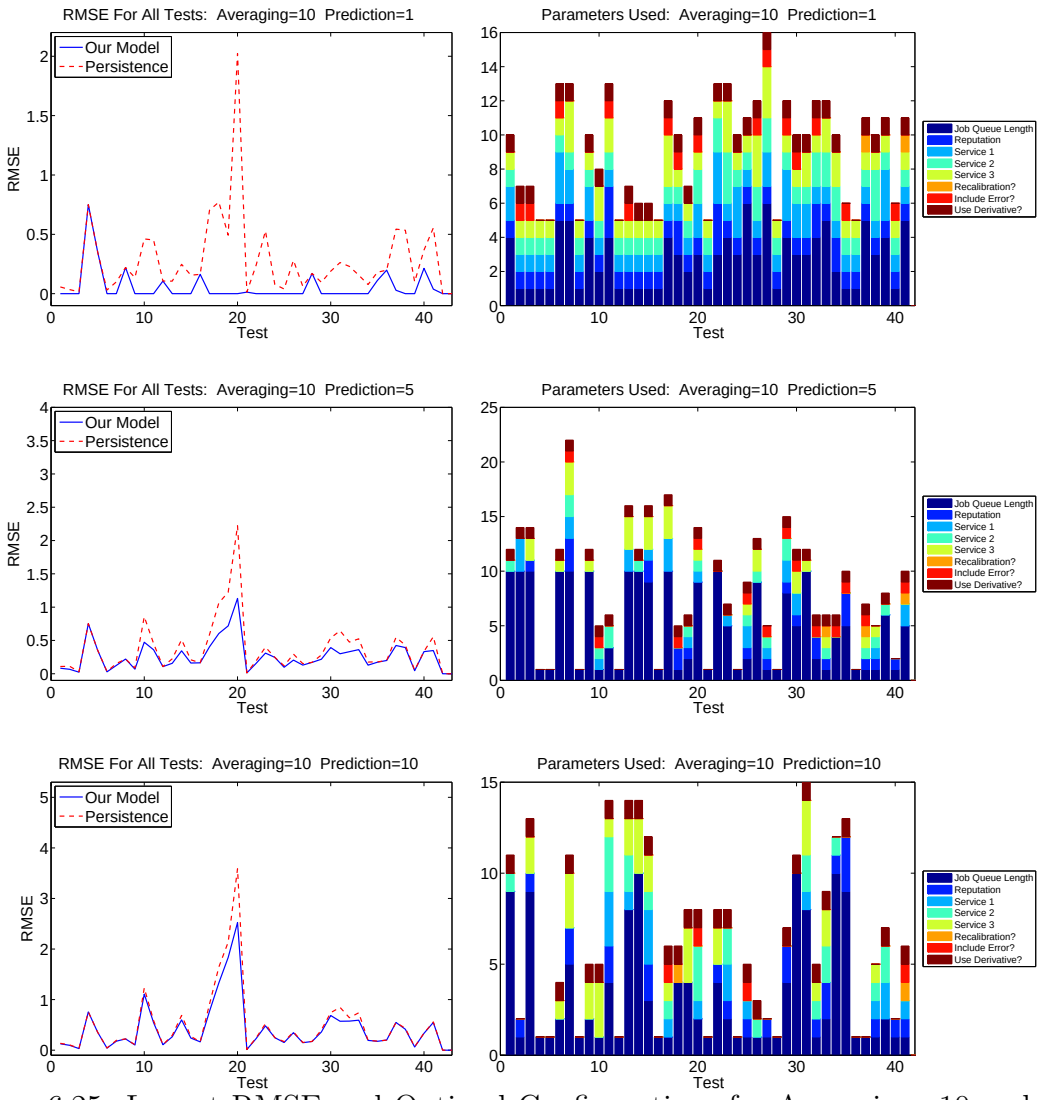

Figure 6.25: Lowest RMSE and Optimal Configurations for Averaging=10 and Prediction={1,5,10}

Our implementation includes the same parameters as used in simulation: job queue length, reputation, number of service 1 jobs, number of service 2 jobs, and number of service 3 jobs.

Each device already maintains a record of all devices' reputations and needs to add monitoring the number of service jobs. While the simulator environment makes this especially easy, we do not think communicating these extra parameters will strain the system and any real implementation will include update messages of these parameters. A device attempting to predict its future job congestion maintains a list of the matrix parameters including the calibration window (defining the number of

rows), the variables and number of each (defining the number of columns), and the prediction window. The structure of this allows for future addition of algorithms to dynamically redefine these parameters; for the purpose of this test we use our simulation results. The device then enlists its one-hop neighbors in computing the model, dynamically determining who they are and partitioning matrix variables amongst them equally. Unlike our previous implementation, we allow unequal partitioning of the columns; devices equally share the variables with each device maintaining all columns associated with that variable. As each column of a given variable includes only one more past value, this actually eases storage requirements as the device only needs to store a number of past values equal to the calibration time plus the number of columns.

Once the matrix is partitioned, the predicting device initiates calibration using the same message queue as for jobs. This allows jobs to take priority over the model and interleaves model calibration with job processing so that computing the model does not affect the quality of service. Each device will first receive a message indicating the beginning of calibration. Following that, calibration occurs according to the model and distributed algorithms. Once calibration concludes, the predicting device communicates the coefficients and distributed prediction begins. The device recalibrates the model according to a fixed calibration window.

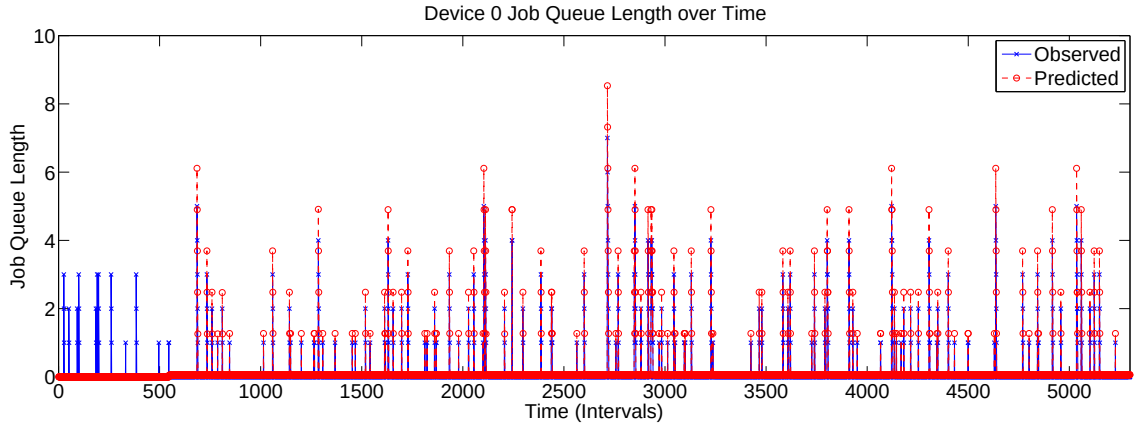

Figure 6.26: Results for Device 0 with Congestion Prediction In Network Simulator

We ran the model using the hybrid configuration shown in 6.5 for the parameters outlined by our simulator analysis: simulation time of 5400 intervals, $\alpha = 0.4$, $\theta = 10$,

and $\eta = 24$. We perform no averaging and predict 1 intervals into the future, which allows time to avert the congestion. Figure 6.26 displays our results for Device 0. The device first performs calibration at approximately 530, hence the lack of predicted values until then. Device 0 in this configuration had 4 neighbors so each maintained the variables and number of columns as outlined in Table 6.4.

Device Number	**Variable Stored**	**Number of Columns**
0	Job Queue	2
1	Reputation	1
2	Number of Service 1 Type Jobs	1
3	Number of Service 2 Type Jobs	1
4	Number of Service 3 Type Jobs	2

Table 6.4: Variables and Columns Maintained by Each Device in Hybrid Configuration

Within the test, we define a congestion level and decide when our prediction indicates future congestion. In these cases, as a initial pass to congestion control and job balancing, we simply forward the next few packets to neighboring nodes. Our test shows that this simple policy reduces the load on the device by 9.68% and removes some of the congestion before it can occur. Figure 6.27 shows the improvement in the job queue over time as we add in our simple congestion control. One simplification of this policy is that we only redirect partial jobs (others require a physical artifact so need more information to redirect). Also, implementing this policy for all nodes might simply move the problem around the network and not balance the load; full analysis of this policy and other policy options we leave for future work and discuss in Chapter 7.

With this test, we complete the full system implementation. Our congestion model successfully works within the MFD network simulator and provides an initial control behavior that reduces congestion.

6.4 Conclusions

In this chapter, we outlined the problem of job congestion on multi function device networks. We defined MFD networks and developed a simulator based on our description. Given this network, we then defined congestion and created a model for

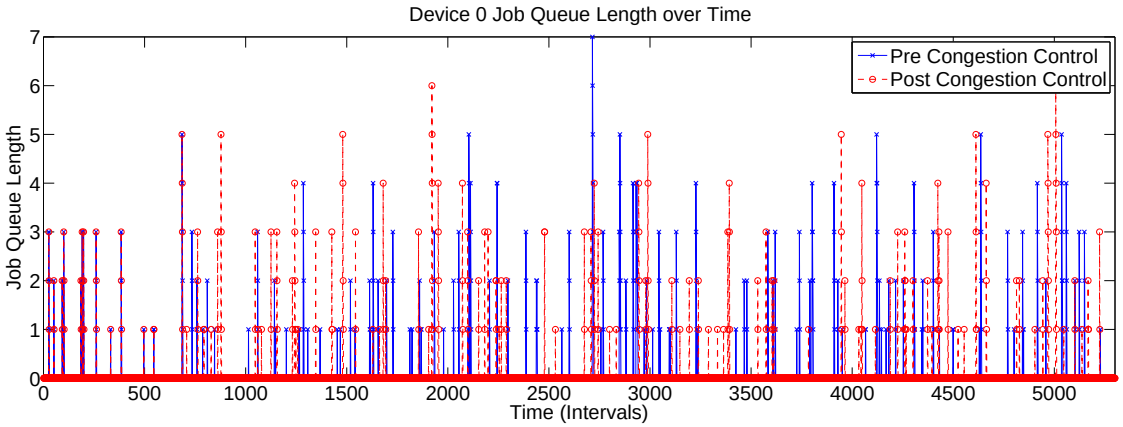

Figure 6.27: Results of Congestion Policy: Device 0 Queue Length Before and After

congestion based on job queue lengths. In creating this congestion model, we used abstract, internally measured variables, a very different approach from our other applications. Our model allows this definition, which demonstrates its flexibility and indicates a wider set of problems for which we can use this model than previously considered.

We tested all components. For our simulator, we analyzed the overall behavior and the effects of certain parameters, defining a specific set that provided interesting yet reasonable behavior. Utilizing these parameters, we verified the congestion model in Matlab and showed that our model performs best in the majority of scenarios, especially due to our flexibility in including a variety of variables. We then implemented the congestion model within the MFD network simulator. Our implementation introduces several new possibilities, including dynamic repartitioning of columns among neighbors and unequal partitioning of columns. This implementation correctly predicts future congestion and performs a simple local response to prevent this congestion.

Chapter 7

Conclusions and Future Work

In this thesis, we presented distributed algorithms for computing, on a sensor network, a matrix pseudoinverse and the related multiple linear regression model. We used these algorithms to solve 3 very different prediction problems: (1) river flooding for disaster mitigation, (2) solar current for energy management, and (3) job congestion for multi function device networks. Each of these share requirements for limited computation, limited communication, and self-calibrating models. Each also provided a different challenge. River flooding required design and implementation of a sensor network capable of autonomously operating in Honduras (or more generally, rural and developing regions), covering a large geographic area with a small number of sensor nodes, and surviving a natural disaster. Solar current required transferring the algorithms to a different sensor network platform and beginning to explore the requirements for dynamic models. Job congestion required defining and developing a MFD network simulator, and ensuring the model could work with abstract, internally measured variables.

We successfully achieved each. In doing so, we learned several lessons regarding sensor network deployments and international development projects; we feel these may be of interest and use to the wider community so share them here. Additionally, we determined many areas for future work, in all application areas as well as in the broader contexts of predicting events on sensor networks and of environmental monitoring sensor networks.

7.1 Lessons Learned

Here we would like to share lessons regarding: theory and algorithms, sensor network platform design, and experiments.

7.1.1 Theory and Algorithms

We learned some lessons regarding the algorithms and simulation that we share here.

Regression models have a nice simplicity to them regarding what data they can use and what they can predict. All that the model requires is that we can describe our event of interest using a time series. Any related (or unrelated) time series can be included in the model as can transformed versions of the event time series (such as differentiation). The methods prefer that the columns of the matrix are linearly independent; one benefit of utilizing real measurements as we do is that the noise naturally existing in the measurements ensures this for us. These models can solve a wide variety of problems. However, they are limited to those problems where sufficient data exists to generate a time series. Binning or averaging the data can help when the problem is only due to a couple missing measurements, but if large holes exist in the data, regression models cannot help.

Additionally, trickiness does arise in these models due to indexing. Where and what data goes in the calibration matrix defines the model and many parameters define where and what data. The number of rows within the matrix depends on the calibration window; however we cannot calibrate the matrix until sufficient time passes to gather the appropriate measurements. This time not only depends on the calibration window but the prediction window as we also have to observe the related predictions associated with each row in the calibration matrix. As several columns are time-shifted variations on the same variable, we also have to observe the appropriate number of those past values which will extend back in time farther than our calibration window. Since each variable could have a differing number of time-shifted columns, we need to ensure each is loaded corresponding to the correct t_{now}. If we are using the prediction error or derivatives, we need to include those

values within our time calculations and matrix indexing. Overall, regression models require careful record keeping: what past data values of each variable go where, what additional computations are necessary and where they go in the matrix, and what is the total observed time window.

Finally, related to our other "theory" area, we had mixed experiences using Python for network simulation. SimPy and Python made implementing the device behaviors easy. We had no problems determining the interactions between the processes, controlling the simulation behavior, and monitoring the internals of the simulator. However, mathematical operations in Python are painful. NumPy provides a nice package for many operations, but we still had difficulties implementing complicated matrix operations. We would recommend SimPy for behavioral analysis of sensor network operations, but recommend finding a different simulator for application implementation and analysis.

7.1.2 Sensor Network Platform Design

We can say little about the Fleck platform design; however, we designed a sensor network for our flood prediction system and learned several lessons from the experience. In the beginning of our platform work, several people collaborated on a base platform (both hardware and software) for an easily reconfigurable sensor network architecture. This hardware formed the base for our sensor network platform of Chapter 4 (used also in two other applications to date: underwater coral reef monitoring [96] and virtual fencing for cattle [79]). Collectively, all of us designing and using this system learned several lessons generally applicable to platform design. We outline here these larger lessons in the context of the more general multi-application sensor network platform.

The startup costs of designing our multi-functional system have been high both on the hardware and the software sides. With such a variety of needs, we found it difficult to initially design each part with the necessary flexibility, often requiring development first for one project style and later modification for the other. However, by sharing the same base hardware and software, debugging is very fast; usually the

other project successfully breaks the new addition in minutes. From this, we suggest multi-application platforms use the same base code, separating out only that code specific to the application, but ensuring both many layers of abstraction and access to functions on all abstraction layers. It is nearly impossible to foresee all usage models of any aspect of the software so access to the various layers ensures the base code remains the base code instead of fragmenting into different software projects.

The predominance of serial peripherals ensures that no processor exists that provides enough serial connections. Having some form of external serial multiplexer is necessary, whether it is a simple serial multiplexer, a SPI-to-UART converter, or a more complicated CPLD/FPGA as we use. This allows for simultaneous use of several communication methods and sensors, a situation that has arisen in all three of our applications.

The communication abstraction infrastructure makes adding and using a new communication device fast and simple. By creating such a complex serial routing structure and utilizing an abstraction encompassing all different bus protocols, we have no problems routing messages through different radios, different expansion boards, and different devices. Given that each application uses a different messaging structure and has different operational behaviors, having this abstraction becomes a necessity and enables future application modalities.

Our system enabled easy prototyping of each application but at higher cost and more complexity than would occur in an application-specific design. Should we want a long-term production system, it would be more optimal from cost, hardware and software standpoints to design separate systems for each application.

7.1.3 Experiments and Deployments

In attempting this work, we attempted, and sometimes achieved, 15 field deployments of sensor networks. Only one of these occurred on a campus network, 3 occurred in the Charles River upstream of Boston in its more suburban location, and 11 occurred in Honduras. A few lessons apply to all experiments, many apply to those in off-campus settings, and some apply only to international development; we will address

those we feel are the most useful and perhaps less obvious below. Some of what we learned in working in Honduras we initially discussed in [7]; we reiterate those lessons here and add new ones.

Power

Power issues plagued all of our deployments in varying degrees. These problems fell into two major categories: solar panel output or power design issues. Solar panel issues occurred with the Fleck and Dover deployments, which all occurred during winter. These deployments saw problems with placing the panels such that they could receive enough solar exposure to offset the power used. With the Fleck deployment, the solution was to move the boxes to another part of campus, easy enough as we had plenty of room to move them. With the Dover deployments, we were constrained in placing the sensor nodes; the river is only accessible in a 2-3 kilometer stretch with very narrow banks and forest on both sides. We solved this by only installing in winter, after enough leaves fell and our panels had solar exposure. This solution did have the drawback of the weaker solar exposure of winter, but that provided more solar exposure than under the leaf-laden trees (and spring was best for Honduras deployments).

The second issue of power system design affected our Dover and Honduras deployments. Obviously, ensuring our sensor networks had a continuous power supply was a key design issue we addressed in our system development. We calculated the expected power draw based on the manufacturer specifications for the major system components. When we had the physical boards, we performed lab tests to understand the power usage and verify our calculations. Based on this, we designed the solar panel and batteries necessary for ensuring one week of operation without any solar current. Yet, in deploying our nodes outside, our calculations did not save our nodes from failing due to lack of power. The larger scale operations, the effects of the environmental conditions on the nodes (heat and high humidity in Honduras, freezing cold and low humidity in Massachusetts), battery aging (leading to capacity reduction), and limitations on solar panel placement all affected the overall system power and reduced the lifetime.

Several approaches helped us deal with these problems. First, we added intelligent battery monitoring to the lithium-polymer batteries, providing the charge current and estimate of remaining capacity. With this information and the solar panel charge current, we created more intelligent policies that took into account time of day and likelihood of receiving more solar power in the near future. We also became overly conservative in our choice of battery capacity, upgrading the system to lithium-polymers with larger-than-necessary capacity and lead-acid batteries at our nodes.

Ultimately, we learned three lessons. First, strongly consider solar exposure when defining a deployment location (if a choice exists), taking daily and seasonal effects into account. Second, obsessively measure all aspects of the power in-situ as it is the only way to truly understand how the node behaves in the environment and how the power operations change. Finally, power system design requires overly conservative estimates, even slightly beyond what one might think reasonable; nothing will ruin a deployment more than lack of power.

Aerocomm AC4790 Radios

Choosing the Aerocomm AC4790 radio was a bad design decision we paid for during field experiments. We picked them after comparative testing of possible radios at MIT using the development boards for all candidates. In those tests, the AC4790 subjectively performed the best (we did not actually measure distances, but considered tests relatively) and promised the longest range, range being our most important consideration at the time. Sadly, they did not live up to their promise.

First, we never saw the promised range. In multiple field trials around MIT with a variety of antennas using both our board and the Aerocomm development board, we achieved at most 2km. Moving our systems to our remote field location where we would expect better performance, we achieved at most 2km. This created significant limitations in placing nodes, especially in the urban areas of Honduras, and led to a few bad placements that led to stolen nodes. We could find no solutions to increase the range and received limited help from Aerocomm in solving this.

Second, we could not use functionality such as "Ready-To-Send" (RTS) signals. RTS allows the host to tell the radio it needs to stop sending data as the host buffer is full. To use this signal on the AC4790, the host microcontroller must switch to something called "Auto Destination" mode. This mode automatically sends all future packets to the last destination from which it received a packet, using the point-to-point protocol. We use the radios in broadcast mode so, in order to use RTS, we must switch in to auto destination mode while receiving packets from the AC4790, but switch out of it to send packets. As sending and receiving are often interleaved within the radio such that the microcontroller does not really know what state the radio is in, performing this little auto destination dance is impossible and meant we could not use the useful RTS signal.

Finally, the AC4790 uses quite a bit of power, more than suggested in the data sheet. This added to our frustrations with power issues during deployment. On start-up (either initial power-on or waking from sleep mode), we also saw it pull significant current, dropping the power plane to levels at which the rest of the board did not function correctly. This caused SD card corruption issues during our system integration test in Honduras. Fundamentally the AC4790 is not to blame for this, what it taught/reminded us is the importance of separate power regulators for the various board segments. Additionally, it points out the additional large power draws a communication device may have, in addition to known current draws during transmit.

Overall, we learned to not use the AC4790 in future designs. We would have switched had we discovered these problems earlier in our test process; unfortunately many of them emerged only during deployments and after we had significant development in the platform based on utilizing the AC4790. In the future, we plan to exhaustively test our communication devices in the most stressful situations we can devise to avoid the pain we experienced using the Aerocomm AC4790.

Testing

Every system needs testing at many different levels-most people agree on the obviousness of that statement. However, in our experience, a large-scale system such as

ours that heavily relies on in-country infrastructure usually follows a test strategy whereby component testing occurs in the lab and complete system testing occurs in the field as an installation of the system in its planned location. In regards to our flood prediction work, this strategy has repeatedly failed us.

On one hand, we needed the components in the US for component improvements and debugging. On the other hand, we needed the entire system in Honduras for complete testing and we could not be in Honduras all the time. This resulted in a combination of approaches. Initially, we tested the components in lab to some level, traveled to Honduras, installed the system as it existed, ran tests for a week or so to find system problems, left some components for longer testing, and returned home with the remaining components for further development. The status of the components left, from a technical point of view, awaited our next trip to Honduras as our in-country partner could tell us if they still existed and if they appeared to work, but not any specifics on how they were working or what may have failed without the remainder of the system. Thus we only discovered long-term problems when we returned to Honduras, where we could perform some debugging but needed to return to the lab for further debugging and most development work. This was not a good system.

Ideally we would have a comparable system within the US, but arranging for a 10,000 km^2 river basin where we could install antenna towers was not feasible. Instead, we discovered local small-scale system testing. By this, we do not mean installing the complete system in the lab or just outside the office; in our experience, this does not show certain fundamental system problems and installation issues. Instead, we mean to find a roughly similar location nearby to the lab that introduces some of the more difficult features (such as measuring river level and outside deployments). This we should have obviously figured out sooner as it would have saved money, time, and sanities. However, we now share this blindness to an obvious fact in hopes others will not repeat it.

What we did was talk to the government organizations in Massachusetts responsible for measuring the river and asked for their help. This resulted in our Dover

test site, which supplied access to the cement structures they use for their antennas, computers, and sensors. Because we could use this existing site and infrastructure, we could install a miniature version of the system (sans antenna towers and government offices) in a local US river a short 40 minute drive away instead of a 6 hour flight. This sped up our development work considerably. In all future development, we plan to start with finding a comparable local location and test our system extensively there first.

International Collaborations

We were very fortunate to collaborate with CSIRO, Brisbane on our solar current project. This collaboration has gone very well over the last year and we look forward to it continuing.

Specifically regarding lessons for deployments, we successfully collaborated on a deployment in Brisbane, testing our solar current prediction, while we were at MIT. Web access to the deployment data in near real-time was the key to the success of our deployment for which all credit goes to CSIRO. CSIRO has a great online interface to all their sensor network deployments, linking to the database storing their data results and providing all the data for all the sensors in graphical online form as well as downloadable for analysis.

For our deployment, we did see some early problems with the location and communication by examining the system remotely, which we could then communicate to CSIRO and have them reposition the nodes. Without an online interface, we would have waited until CSIRO downloaded the data from the nodes and forwarded it to us some time into the deployment, at which point the test would have failed as the nodes ran out of power. Instead, the interface ensured a great deployment and successful test.

Our work focuses on enabling autonomous deployments so has not yet examined such an interface; however, we can unequivocally state that the existence of one is necessary to remote monitoring and international collaborations.

International Development Partnerships

The partnership we created with CTSAR in developing our flood prediction system worked very well and the structure of the collaboration introduced us to one of the more successful frameworks for international development projects. Key to everything was their identification of the problem, request for our help, and securing initial funding. Having them initiate the project ensured their commitment to it through all of the setbacks and design changes such projects endure. They initiated it because of community feedback, guaranteeing the support of the community in implementing and maintaining the system. Had we introduced the project, given its need for community input and lack of potential income, our eventual withdrawal would most likely begin the slow decay of the system, as the force behind the project would disappear.

Within the partnership, we divided the work nearly equally between both partners and divided the responsibility for providing resources. Allowing CTSAR to insulate our work from the community ensured the progress of the work and cooperation of all parties. We could not travel to Honduras all of the time and did not have a continuous connection with the communities, but did have better access to technical resources when in the US. CTSAR had both access to and a history of work with the communities, creating a relationship that fosters cooperation, but they did not have the technical skills nor access to materials. This equal division of tasks and shared commitment through resource buy-in, while seemingly obvious, does not exist in many projects and the lack of a committed partner often causes the failure of that project.

Security

Our Dover installations did not see security issues (although we always feared them occurring); however, in Honduras, we dealt with many security issues throughout our project. While our in-country partnership aided in providing community support, the scale of our work region limited the number of people directly aware of the reasoning and goals of our work. This led to thefts and confusion.

Several early sensor prototypes were stolen from their installation near the river because, while the sensor was in the river, the electronics and cable were placed at the top of bridges or near the bank for water protection and easy retrieval. In one instance, someone stole over 180 meters of cable running along the bridge connecting the electronics to the sensor. We had connected it to electricity cables running along the bridge in order to disguise it and the person, in taking those cables, took ours as well. Children climbed on our towers because they were there and, after providing more security, shot marbles at boxes on top of the towers. One marble punctured a plastic box, creating a small hole in which rainwater entered, damaging our battery and radio stored there because we had not expected marble damage (a story also applicable to the general theme that field deployments never turn out how one expects and certain errors are never predictable).

Overall, our lesson has repeatedly been that there is no such thing as too much security and if we find accessing our equipment difficult for routine work, then it presents too much work for thieves. In response to these problems, we placed all our cable within PVC pipes that we bolted to the bridge so that people could not cut or easily steal them. We later upgraded this to a fully wireless design (which then had problems due to the Aerocomm radios, see the above sections for more information).

We upgraded our tower security to include razor wire, padlocks too small for chain cutters, and metal panels. Our towers can only be climbed with a ladder (see Figure 7.1) and we upgraded our boxes to metal after the marble incident.

We did figure out some methods requiring less construction as well. Our very white rainfall sensors now wear camouflage, cut from girls' pants and providing a cheap, effective method of hiding these nodes (see Figure 7.2). For our sensors, especially the rainfall and temperature sensors, we discovered a couple of friendly families within our urban site to host the sensors thereby protecting them from thieves and minimizing the infrastructure needed. This solution was not without problems; several people did not understand the nature of our system and threatened the families into giving them the boxes or hiding the boxes. Unsurprisingly, the sensor network did not survive either behavior. In future work, we need to address the social side

Figure 7.1: 5 Meter Antenna Tower with Security

in addition to the technical, holding town meetings to introduce the project to those communities where we install sensor nodes and providing them with enough information to understand the system.

Community Knowledge

Lacking historical data in which to develop flood prediction algorithms or train models, our initial plan consisted of installing sensors and gathering the data for ourselves. The realization that we could not gather all the data necessary in a reasonable timeframe led us to consider other methods than those commonly used by the hydrology community. We discovered that we could achieve the results for which we wanted the historical data, at some level, from the memories of the community members who view the river every day along with the historical effects of floods. This allowed us to leapfrog the many years necessary to gather enough data for current models and,

Figure 7.2: Camouflaged Rainfall Sensor

using a much simpler and cheaper method, generate a table of what constitutes a flood. We also plan to utilize this insight in providing a check on the system by allowing users to input information on the current state of the river and details on past flooding, which the system can then use to refine the models and verify sensor activity.

Permission

We also dealt with issues of permission throughout our deployments, some less obvious than others. In Dover, we needed permission for land use. We learned of the site through the USGS and received permission from them to use their small location. However, to deploy the system in a useful and interesting way, we needed to expand beyond their location. It appeared that the land across the river was a public park and refuge so we placed nodes on that side as well. In a classic case of not making assumptions, we then discovered that it was actually private property. Fortunately, the owners kindly let us keep our sensors on their land and our deployment successfully concluded. For future deployments, the lesson is to identify the full area before deployment and make sure permission exists for all of it.

In Honduras, we had much different experiences. One benefit of working there is the lack of concern about permission. We asked the government about radio frequencies to use and they did not care what frequency we used as long as we did not monopolize it. For land use, we worked with local officials to identify where we could install our towers, again seeing a lack of care as we only wanted unoccupied space near the town dump (a fun place for a deployment). After that, when we needed to

install cable along the bridge, we just did it and no one cared about permission (which would not occur in the US). However, the opposite side emerged when we did need permission for using existing structures, as we did once we started having security issues. Then it became difficult to talk to the appropriate people and required several trips to remote cities attempting to meet with the right people. For one location where we wanted to place an antenna and box on an existing tower, despite calling in advance to ensure the right people were available, they never were once we arrived and we never managed to get permission, but had to reconsider our installation plans.

Overall, we cannot sum these experiences into a single pithy lesson, but share them in order to emphasize the details that arise in deploying systems in the "real-world" and the considerations we need to take before doing so.

The Unexpected

Finally, the phrase "expect the unexpected" holds true for deployments. Something will occur during a field deployment that could not be planned for or expected; once a person has performed enough deployments this becomes clear. For those just beginning to deploy systems, we outline a few of our stories to illuminate this concept.

First, something always had broken by the time we arrived at our deployment site. With Dover, as we traveled by car, this usually was a couple disconnected cables, which we easily fixed. For Honduras, since our equipment first flew and then jostled along a truck ride, this usually was more serious; we always had an office day repairing unhappy nodes harassed by the TSA and travel. Often we ended up replacing boards to ensure we could deploy the number of nodes we wanted; as we always brought spares, this was usually sufficient. Once, however, a box of spare boards disappeared between MIT and Honduras, requiring a FedEx delivery (no other postal service exists) that took a week and delayed everything.

Our marble story (described earlier in the context of security) also provides a great example of unexpected situations. We installed our node with its 144 MHz radio and car battery inside a plastic box on top of a tower, assuming the plastic would protect everything from rain and exposure. However, we did not count on children deciding

our box would make a great target for marble shooting. One hit the box which was brittle due to the sun and made a little hole in the box. Rain then entered the little hole and ruined the radio. We had no reason to expect this problem and design around it until it occurred.

Another issue we faced with our installation occurred after this event. Our installation procedure switched to using plastic Otter boxes to house anything damageable by water. We installed this on top of a tower and everything worked well at first, but we soon saw some issues with the radio transmissions failing. After debugging the issue, it turned out that the structure became too hot after sitting in the sun and the mechanical power relays stopped working. We had designed the system such that it could withstand these temperatures, but the temperatures were too close to the margins for the relays, apparently. We had to place a small fan within the box, add ventilation, and provide a roof for the tower to avoid this problem, which became a large amount of engineering in Honduras and could not have been predicted.

With the security issues surrounding our cables in Honduras, we also ran into unexpected problems. After driving out to the site with no expectation that our cable would be missing, we had to change all plans to figure out how to repair over 180 m of missing cable. When we installed the cable initially, we had soldered all the connectors and spools together in the lab (it is hard to find a 180 m spool available in Honduras and even harder to fly with that much cable). Now we were faced with performing this same task in the field as well as running all that cable through PVC pipes to reduce future occurrences. We managed it, but it was an adventure. In addition to the obvious problems, we also faced issues with keeping the connectivity correct for RS485 signals down to the sensor node at the bottom of the bridge (do not decide to use 2 2-conductor cables instead of 1 4-conductor cable or take less-than-idiot-proof notes!).

What also complicated much of our work was the fact that we were installing either in a river or on a bridge above a river, and the river is big with a strong current. Most of this required planning in advance, but occasionally we had to improvise solutions to install in the river when it was too high for standing or too strong for swimming.

The best installation story was installing the cable to run from the sensor box up to the top of the bridge. The cable has to be bolted to the bridge at least part of the way up to avoid snagging on passing debris. We eventually had to place a ladder inside a boat to reach high enough to place the bolts - not an installation strategy one can plan in Boston!

This sub-set of our stories highlights the random things that arise during field deployments for which one rarely can prepare. The only thing we recommend to alleviate this is packing an assortment of supplies to enable addressing the issue in the field. We suggest: duct tape, electrical tape, ziploc bags, zipties of assorted sizes, a collection of resistors and capacitors (in addition to the spare parts kit we assume you would already bring), nylon rope, a knife, an inverter, a car battery, a soldering kit, spare batteries and fuses for your multimeter, tools (not just a multimeter, but pliers, a wrench, and assorted screwdrivers, to name a few items), and a first aid kit. Hopefully, most of it will be unnecessary (especially the first aid kit), but, in our experience, you are sure to need them if you do not take them!

7.2 Future Work

With so many ideas of extensions and additions for our projects, we divide our future work into three areas: (1) specific extensions of our application areas, (2) general extensions of event prediction on sensor networks, and (3) general extensions in sensor networks for environmental monitoring.

7.2.1 Application Areas

Within each of our three application areas, we have several ideas to extend the work for that application area.

River Flooding

Our initial extension of this work is to fix the minor issues affecting our platform performance, namely the AC4790 transmission range and power issues that corrupt the SD cards. We should fix these easily with only the complexity of a board rebuild to address. In addition to fixing these issues, we would also like to explore the use of cell

phone modules to provide an alternative communication method to 144 MHz. The coverage in Honduras has advanced significantly and, with three carriers in the same area, significant competition exists to ensure reliable networks. This extension would allow for installations in regions with poor line-of-sight for 144 MHz communication and provide a possible method for transmitting the data from the remote office to the capital.

With these fixes and changes, we would like to deploy the system for longer than a month. This will also require addressing the social and security issues we have seen, which we now have a better grasp on and our community partner should have the capacity to help. A longer deployment will allow us to verify the validity of the system for solving the disaster monitoring problem; we could then consider what facets still need generalization for disseminating the project to other areas. We believe that the base system is general enough, but the other aspects will need work, making it more a project for other branches of engineering with whom we would be interested in collaborating.

We also have interesting, unexplored questions regarding how to communicate the information to the communities and wider audience. Significant issues arise with the variety of education levels and issues of mispredicts which we will need to explore. These questions will entail exploring user interfaces as well as other hardware methods of alerting such as sirens or lights.

Finally, we would like to include our solar current prediction work in this system to enable better overall power management. As the system already supports the mathematical operations, this should be an easy extension with good benefits for the system lifetime.

Solar Current

For solar current, we want to include the prediction on an environmental monitoring sensor network, such as our flood prediction system. This would allow us to explore any issues arising with this prediction and power management conflicting with regular operation. If we also place it in a long-term deployment, we can explore the long-term benefits of running this prediction on the sensor network system.

Job Congestion

We would like to continue our work exploring control policies to balance jobs and relieve congestion within the MFD network. Instead of just shifting the message to any neighbor we could try analyzing which neighbors see less traffic due to the network topology or based on their predictions of congestion. Additionally, issues arise with the physical artifacts needed by the user of the device, but perhaps possibilities exist to use other devices that are physically near the device.

Finally, we hope to test everything on a real system at some point. The simulator matches the small, basic test system, but how it compares to a larger system still needs verification. We also want to ensure the model runs with sufficiently low overhead on the system to not interfere with normal operating behavior.

7.2.2 Event Prediction on Sensor Networks

We have many ideas to improve and extend our event prediction on sensor networks.

First, we need a more dynamic model extending the ideas introduced in Section 5.3. The system needs to autonomously determine an appropriate structure; to achieve this we must first have a reasonable centralized method and then optimize the computation to work on a sensor network. This should include determining reasonable time windows in addition to matrix structure. With a dynamic model, we remove any reliance on an existing data set and human expertise to determine a reasonable structure.

Next, the model requires fault tolerance. We need to include some form of data replication across the nodes, which will also require memory management. We also need a method to choose who participates in the calibration; perhaps a voting mechanism or a priority-based scheme that will allow nodes to self-identify if they are "in" the model computation or not. This should include a method for transferring ownership of the first column so that the system tolerates failure of that node in cases where the prediction is used globally for the system.

We also want to implement the fully distributed version on a sensor network and see what issues may arise. This version applies best to events with significant data

needs in terms of number of variables used in the matrix, or number of columns, as this relates to the ability of a node to maintain centralized versions of the SVD matrices. Additionally, it requires reliable communication; the calibration can fail gracefully already when communication fails, but it is hardly useful to have a model that constantly fails due to messaging issues. This may limit the platform options (we could not install it on our platform until we replace the AC4790).

Finally, we want to explore other modeling uses for the pseudoinverse and other model possibilities for prediction. Other models do exist that use the pseudoinverse including the wide variety of other regression models and logistic growth models; these are other options that may better address certain prediction problems. Nonlinear models that do not use the pseudoinverse could provide another set of possible methods to predict events with significant nonlinearities. This would also introduce questions of the system autonomously switching between model forms to find the best one as environmental conditions change over the long-term.

7.2.3 Sensor Networks for Environmental Monitoring

In the broader area of sensor networks for environmental monitoring, we have several ideas of future areas of work. Expanding to different application areas offers new problems in measuring, detecting, and predicting events. One is our project from Chapter 1 that of monitoring endangered species, specifically the Manzano Mountain Cottontail. Disaster mitigation also provides a good use case for sensor networks with interesting behavior patterns and reliability issues. Here, landslide detection and forest fire detection provide two examples of problems that environmental monitoring sensor networks could aid. Finally, agriculture, especially irrigation, could use this technology to provide more efficiency, especially in the area of water use where the inefficiencies in the face of increasing water shortages will affect the economics and the environment.

Outside of application specific problems, two problems hinder widespread adoption of sensor networks: reliability and robustness. Many innovations are needed to solve this; initially, we need more fault tolerance within the software and better overall

power management. Improving the ability of the sensor network to recover from failures (from system-wide issues to individual node resilience) and improving the lifetime of the system though power management will enable a more stable platform, leading to more general usage of sensor networks outside the research community.

7.3 Summary

Overall, we provide a set of tools, applications, and experiences in this thesis. Hopefully through our explanations and lessons learned, we enable others to include more event predictions within sensor networks. As we described in this chapter, many interesting problems still remain and we look forward to solving them.

Bibliography

[1] Aerocomm. *AC4790 900 MHz OEM Transceivers User Manual*, 1.3 edition.

[2] Newsha K. Ajami, Hoshin Gupta, Thorsten Wagener, and Soroosh Sorooshian. Calibration of a semi-distributed hydrologic model for streamflow estimation along a river system. *Journal of Hydrology*, 298:112–135, October 2004.

[3] ALERT Systems Organization. Alert history. `http://www.alertsystems.org`.

[4] Torsha Banerjee, Kaushik R. Chowdhury, and Dharma P. Agrawal. Using polynomial regression for data representation in wireless sensor networks. *International Journal of Communication Systems*, 20(7):829–856, 2007.

[5] Elizabeth Basha. Interview with COPECO officials in La Masica, Honduras, January 2004.

[6] Elizabeth Basha, Sai Ravela, and Daniela Rus. Model-based monitoring for early warning flood detection. In *Proceedings of the International Conference on Embedded Networked Sensor Systems (SenSys)*, Raleigh, NC, USA, November 5-7 2008.

[7] Elizabeth Basha and Daniela Rus. Design of early warning flood detection systems for developing countries. In *Proceedings of the Conference on Information and Communication Technologies and Development (ICTD)*, Bangalore, India, December 2007.

[8] M. W Benson and P. O Frederickson. Fast parallel algorithms for the Moore-Penrose pseudo-inverse. In *Proceedings of the Conference on Hypercube Multiprocessors*, Knoxville, TN, USA, September 1986. SIAM.

[9] Jan Beutel, Stephan Gruber, Andreas Hasler, Roman Lim, Andreas Meier, Christian Plessl, Igor Talzi, Lothar Thiele, Christian Tschudin, Matthias Woehrle, and Mustafa Yuecel. PermaDAQ: a scientific instrument for precision sensing and data recovery in environmental extremes. In *Proceedings of the International Symposium on Information Processing in Sensor Networks (IPSN)*, pages 265–276, San Francisco, CA, USA, 2009. IEEE Computer Society.

[10] Keith Beven, Renata Romanowitz, Florian Pappenberger, Peter Young, and Micha Werner. The uncertainty cascade in flood forecasting. In *International*

conference on innovation advances and implementation of flood forecasting technology, 2005.

[11] A. M. Bianco, M. García Ben, E. J. Martínez, and V. J. Yohai. Outlier detection in regression models with ARIMA errors using robust estimates. *Journal of Forecasting*, 20(8):565–579, 2001.

[12] Christian H. Bischof. A parallel ordering for the block Jacobi method on a hypercube architecture. In *Proceedings of the Conference on Hypercube Multiprocessors*, pages 612–618, Knoxville, TN, USA, September 1986. SIAM.

[13] Christian H. Bischof. Computing the singular value decomposition on a distributed system of vector processors. *Parallel Computing*, 11(2):171–186, August 1989.

[14] George E. P. Box and Gwilym M. Jenkins. *Time series analysis: forecasting and control*. Holden-Day, San Francisco, CA, USA, 1976.

[15] Rafael Bras and Ignacio Rodriguez-Iturbe. *Random Functions and Hydrology*. Dover Publications, Inc, Mineola, NY, USA, 1993.

[16] A. Brath, A. Montanari, and E. Toth. Neural networks and non-parametric methods for improving real-time flood forecasting through conceptual hydrological models. *Hydrology and Earth System Sciences*, 6(4):627–639, 2002.

[17] Richard Brent, Franklin Luk, and Charles Van Loan. Computation of the singular value decomposition using mesh-connected processors. *Journal of VLSI and Computer Systems*, 1(3):242–270, 1985.

[18] Zack Butler, Peter Corke, Ron Peterson, and Daniela Rus. From robots to animals: Virtual fences for controlling cattle. *International Journal of Robotics Research*, 25(5-6):485–508, 2006.

[19] Mauricio Castillo-Effen, Daniel H. Quintela, Ramiro Jordan, Wayne Westhoff, and Wilfrido Moreno. Wireless sensor networks for flash-flood alerting. In *Proceedings of the International Caracas Conference on Devices, Circuits and Systems*, pages 142–146, Dominican Republic, Nov 2004. IEEE.

[20] Center for Hydrometeorology & Remote Sensing, University of California, Irvine. Hydrologic predictions - on-going activities. `http://chrs.web.uci.edu/research/hydrologic_predictions/activities07.html`.

[21] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Computing Surveys*, 41(3):1–58, 2009.

[22] Youngjin Choi. New form of block matrix inversion. In *International Conference on Advanced Intelligent Mechatronics*, Singapore, July 2009.

[23] Rashed Chowdhury. Consensus seasonal flood forecasts and warning response system (FFWRS): an alternate for nonstructural flood management in Bangladesh. *Environmental Management*, 35:716–725, May 2005.

[24] Peter Corke and Pavan Sikka. FOS — a new operating system for sensor networks. In *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*, Bologna, Italy, January 2008.

[25] Peter Corke, Philip Valencia, Pavan Sikka, Tim Wark, and Les Overs. Long-duration solar-powered wireless sensor networks. In *Proceedings of the Workshop on Embedded Networked Sensors*, pages 33–37, Cork, Ireland, 2007. ACM.

[26] Guiomar Corral, Agustín Zaballos, Joan Camps, and Josep Maria Garrell. Prediction and control of short-term congestion in ATM networks using artificial intelligence techniques. In *Proceedings of the International Conference on Networking*, pages 648–657. Springer-Verlag, 2001.

[27] Geoff Coulson, Danny Hughes, Gordon Blair, and Paul Grace. The evolution of the GridStix wireless sensor network platform. In *International Workshop on Sensor Network Engineering*, Santorini, Greece, June 2008.

[28] Vronique Delouille, Ramesh Neelamani, and Richard Baraniuk. Robust distributed estimation in sensor networks using the embedded polygons algorithm. In *Proceedings of the International Symposium on Information Processing in Sensor Networks (IPSN)*, pages 405–413, Berkeley, CA, USA, 2004. ACM.

[29] Mark DeMaria and John Kaplan. An updated statistical hurricane intensity prediction scheme (SHIPS) for the atlantic and eastern north pacific basins. *Weather and Forecasting*, 14(3):326–337, 1999.

[30] Petros Drineas, Michael W. Mahoney, S. Muthukrishnan, and Tamas Sarlos. Faster least squares approximation. *ArXiv e-prints*, abs/0710.1435, 2007.

[31] Deborah Estrin, Ramesh Govindan, John Heidemann, and Satish Kumar. Next century challenges: scalable coordination in sensor networks. In *Proceedings of the International Conference on Mobile Computing and Networking (MobiCom)*, pages 263–270, Seattle, WA, USA, 1999. ACM.

[32] L. Magnus Ewerbring and Franklin T. Luk. Computing the singular value decomposition on the connection machine. *IEEE Transactions on Computers*, 39(1):152–155, 1990.

[33] Kai-Wei Fan, Zizhan Zheng, and Prasun Sinha. Steady and fair rate allocation for rechargeable sensors in perpetual sensor networks. In *Proceedings of the International Conference on Embedded Network Sensor Systems (SenSys)*, pages 239–252, Raleigh, NC, USA, 2008. ACM.

[34] Federal Communications Commission: Public Safety and Homeland Security Bureau. Emergency alert system. `http://www.fcc.gov/pshs/eas/`.

[35] Bryce D. Finnerty, Michael B. Smith, Dong-Jun Seo, Victor Koren, and Glenn E. Moglen. Space-time scale sensitivity of the Sacramento model to radar-gage precipitation inputs. *Journal of Hydrology*, 203:21–38, December 1997.

[36] Pedro Galeano, Daniel Pea, and Ruey S Tsay. Outlier detection in multivariate time series by projection pursuit. *Journal of the American Statistical Association*, 101(474):654–669, 2006.

[37] Konstantine P. Georgakakos. Analytical results for operational flash flood guidance. *Journal of Hydrology*, 317:81–103, February 2006.

[38] N. Gnanasambandam, N. Sharma, S. R.T Kumara, and Hua Liu. Collaborative Self-Organization by devices providing document services - a Multi-Agent perspective. In *IEEE International Conference on Autonomic Computing*, pages 305 – 308, June 2006.

[39] Gene H Golub and Charles F Van Loan. *Matrix Computations*. Johns Hopkins studies in the mathematical sciences. Johns Hopkins University Press, Baltimore, 1st edition, 1983.

[40] Gene H Golub and Charles F Van Loan. *Matrix Computations*. Johns Hopkins studies in the mathematical sciences. Johns Hopkins University Press, Baltimore, 3rd edition, 1996.

[41] Carlos Guestrin, Peter Bodik, Romain Thibaux, Mark Paskin, and Samuel Madden. Distributed regression: an efficient framework for modeling sensor network data. In *Proceedings of the International Symposium on Information Processing in Sensor Networks (IPSN)*, pages 1–10, Berkeley, CA, USA, 2004. ACM.

[42] Richard Guy, Ben Greenstein, John Hicks, Rahul Kapur, Nithya Ramanathan, Tom Schoellhammer, Thanos Stathapoulos, Karen Weeks, Kevin Chang, Lew Girod, and Deborah Estrin. Experiences with the extensible sensing system ESS. In *Proceedings of CENS Technical Report #60*. CENS, March 2006.

[43] William W. Hager. Updating the inverse of a matrix. *SIAM Review*, 31(2):221–239, June 1989.

[44] T. M. Hopson and P. J. Webster. Operational short-term flood forecasting for Bangladesh: application of ECMWF ensemble precipitation forecasts. *Geophysical Research Abstracts*, 8, 2006.

[45] Faisal Hossain, Nitin Katiyar, Yang Hong, and Aaron Wolf. The emerging role of satellite rainfall data in improving the hydro-political situation of flood monitoring in the under-developed regions of the world. *Journal of Natural Hazards*, 43:199–210, March 9 2007.

[46] Jason Hsu, Sadaf Zahedi, Aman Kansal, Mani Srivastava, and Vijay Raghunathan. Adaptive duty cycling for energy harvesting systems. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 180–185, Tegernsee, Bavaria, Germany, 2006. ACM.

[47] Wen Hu, Van Nghia Tran, Nirupama Bulusu, Chun Tung Chou, Sanjay Jha, and Andrew Taylor. The design and evaluation of a hybrid sensor network for cane-toad monitoring. In *Proceedings of the International Symposium on Information Processing in Sensor Networks (IPSN)*, pages 503–508, Los Angeles, CA, USA, April 2005. IEEE.

[48] Danny Hughes, Nelly Bencomo, Gordon Blair, Geoff Coulson, Paul Grace, and Barry Porter. Exploiting extreme heterogeneity in a flood warning scenario using the gridkit middleware. In *Proceedings of the Middleware Conference Companion*, pages 54–57, Leuven, Belgium, 2008. ACM.

[49] Danny Hughes, Mickaël Daudé, Geoff Coulson, Gordon Blair, Paul Smith, Keith Beven, and Wlodek Tych. Managing heterogeneous data flows in wireless sensor networks using a "Split personality' mote platform. In *Proceedings of the International Symposium on Applications and the Internet (SAINT)*, pages 145–148, 2008.

[50] Danny Hughes, Phil Greenwood, Gordon Blair, Geoff Coulson, Paul Grace, Florian Pappenberger, Paul Smith, and Keith Beven. An experiment with reflective middleware to support grid-based flood monitoring. *Concurrency and Computation: Practice & Experience*, 20(11):1303–1316, 2008.

[51] Danny Hughes, Phil Greenwood, Gordon Blair, Geoff Coulson, Florian Pappenberger, Paul Smith, and Kevin Beven. An intelligent and adaptable grid-based flood monitoring and warning system. In *Proceedings of the UK eScience All Hands Meeting*, 2006.

[52] Danny Hughes, Phil Greenwood, Geoff Coulson, and Gordon Blair. GridStix: supporting flood prediction using embedded hardware and next generation grid middleware. In *Proceedings of the International Workshop on Mobile Distributed Computing*, 2006.

[53] IUCN, Conservation International, Arizona State University, Texas A&M University, University of Rome, University of Virginia, and Zoological Society London. An analysis of mammals on the 2008 IUCN red list. `http://www.iucnredlist.org/mammals`, 2008. Downloaded on 1 February 2010.

[54] Valeriy Y. Ivanov, Enrique R. Vivoni, Rafael L. Bras, and Dara Entekhabi. Preserving high-resolution surface and rainfall data in operational-scale basin hydrology: a fully-distributed physically-based approach. *Journal of Hydrology*, 298:80–111, October 2004.

[55] Xiaofan Jiang, Joseph Polastre, and David Culler. Perpetual environmentally powered sensor networks. In *Proceedings of the International Symposium on Information Processing in Sensor Networks (IPSN)*, page 65, Los Angeles, CA, USA, 2005. IEEE Press.

[56] Gregers H. Jørgensen and Jacob Høst-Madsen. Development of a flood forecasting system in Bangladesh. In *Operational Water Management Conference*, 1997.

[57] Philo Juang, Hidekazu Oki, Yong Wang, Margaret Martonosi, Li Shiuan Peh, and Daniel Rubenstein. Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with ZebraNet. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 96–107, San Jose, CA, USA, 2002. ACM Press.

[58] Aman Kansal, Jason Hsu, Mani Srivastava, and Vijay Raghunathan. Harvesting aware power management for sensor networks. In *Proceedings of the Design Automation Conference*, pages 651–656, San Francisco, CA, USA, 2006. ACM.

[59] Aman Kansal, Jason Hsu, Sadaf Zahedi, and Mani B. Srivastava. Power management in energy harvesting sensor networks. *ACM Transactions in Embedded Computing Systems*, 6(4):32, 2007.

[60] T. N. Krishnamurti, C. M. Kishtawal, Zhan Zhang, Timothy LaRow, David Bachiochi, Eric Williford, Sulochana Gadgil, and Sajani Surendran. Multimodel ensemble forecasts for weather and seasonal climate. *Journal of Climate*, 13(23):4196–4216, 2000.

[61] Shaobo Liu, Qing Wu, and Qinru Qiu. An adaptive scheduling and voltage/frequency selection algorithm for real-time energy harvesting systems. In *Proceedings of the Design Automation Conference*, pages 782–787, San Francisco, CA, USA, 2009. ACM.

[62] Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk, and John Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the International Workshop on Wireless Sensor Networks and Applications (WSNA)*, pages 88–97, Atlanta, GA, USA, 2002. ACM Press.

[63] Sabine M McConnell and David B Skillicorn. A distributed approach for prediction in sensor networks. In *Proceedings of the Workshop on Sensor Networks, SIAM International Conference on Data Mining*, 2005.

[64] I. Z. Milovanovic, E. I. Milovanovic, and M. K. Stojcev. Matrix inversion algorithm for linear array processor. *Mathematical and Computer Modeling*, 16(12):133–141, December 1992.

[65] Clemens Moser, Jian-Jia Chen, and Lothar Thiele. Power management in energy harvesting embedded systems with discrete service levels. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 413–418, San Fancisco, CA, USA, 2009. ACM.

[66] Clemens Moser, Lothar Thiele, Davide Brunelli, and Luca Benini. Robust and low complexity rate control for solar powered sensors. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 230–235, Munich, Germany, 2008. ACM.

[67] NumPy Developers. Numpy homepage. `http://numpy.scipy.org/`.

[68] National Oceanic and Atmospheric Administration's National Weather Service. Distributed model intercomparison project. `http://www.nws.noaa.gov/oh/hrl/dmip/`.

[69] National Oceanic and Atmospheric Administration's National Weather Service. HL distributed modeling research. `http://www.nws.noaa.gov/oh/hrl/distmodel/abstracts.htm#abstract_7`.

[70] Victor Pan. Parallel least-squares solution of general and toeplitz systems. In *Proceedings of the International Symposium on Parallel Algorithms and Architectures*, pages 244–253, Crete, Greece, 1990. ACM.

[71] Phillips. *LPC241x User Manual*, 2 edition, July 2006.

[72] Joel B. Predd, Sanjeev R. Kulkarni, and H. Vincent Poor. Regression in sensor networks: training distributively with alternating projections. In *Proceedings of Advanced Signal Processing Algorithms, Architectures, and Implementations*, San Diego, CA, USA, 2005.

[73] Joel B. Predd, Sanjeev R. Kulkarni, and H. Vincent Poor. A collaborative training algorithm for distributed learning. *IEEE Transactions on Information Theory*, 55(4):1856–1871, 2009.

[74] Andres Quiroz, Nathan Gnanasambandam, Manish Parashar, and Naveen Sharma. Robust clustering analysis for the management of self-monitoring distributed systems. *Cluster Computing*, 12(1):73–85, 2009.

[75] Andres Quiroz, Manish Parashar, Nathan Gnanasambandam, and Naveen Sharma. Clustering analysis for the management of Self-Monitoring device networks. In *Proceedings of the International Conference on Autonomic Computing*, pages 55–64, Chicago, IL, USA, 2008.

[76] Nithya Ramanathan, Laura Balzano, Deborah Estrin, Mark Hansen, Thomas Harmon, Jenny Jay, William Kaiser, and Gaurav Sukhatme. Designing wireless sensor networks as a shared resource for sustainable development. In *Proceedings of the International Conference on Information and Communication Technologies and Development (ICTD)*, pages 256–265, May 2006.

[77] Seann Reed, Victor Koren, Michael Smith, Ziya Zhang, Fekadu Moreda, Dong-Jun Seo, and DMIP Participants. Overall distributed model intercomparison project results. *Journal of Hydrology*, 298:27–60, October 2004.

[78] Renata J. Romanowitz, Peter C. Young, and Keith J. Beven. Data assimilation in the identification of flood inundation models: derivation of on-line multi-step ahead predictions of flows. In *International Conference: Hydrology, Science and Practice for the 21st century*, volume 1, pages 348–353. 2004.

[79] Mac Schwager, Carrick Detweiler, Iuliu Vasilescu, Dean M. Anderson, and Daniela Rus. Data-driven identification of group dynamics for motion prediction and control. *Journal of Field Robotics*, 25(6-7):305–324, 2008.

[80] Leo Selavo, Anthony Wood, Qing Cao, Tamim Sookoor, Hengchang Liu, Aravind Srinivasan, Yateng Wu, Woochul Kang, John Stankovic, D. Young, and J. Porter. Luster: wireless sensor network for environmental research. In *Proceedings of the International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 103–116, Sydney, Australia, 2007. ACM.

[81] R. R. Shrestha and Franze Nestmann. River water level prediction using physically based and data driven models. In *Proceedings of the International Congress on Modelling and Simulation (MODSIM)*, pages 1894–1900. Modelling and Simulation Society of Australia and New Zealand, December 2005.

[82] Pavan Sikka, Peter Corke, Philip Valencia, Christopher Crossman, Dave Swain, and Greg Bishop-Hurley. Wireless adhoc sensor and actuator networks on the farm. In *Proceedings of the 5th International Symposium on Information Processing in Sensor Networks (IPSN)*, pages 492–499, Nashville, TN, USA, 2006. ACM Press.

[83] SimPy Development Team. Simpy homepage. `http://simpy.sourceforge.net/`.

[84] A.T. Smith and A.F. Boyer. Sylvilagus cognatus. In *IUCN 2010. IUCN Red List of Threatened Species. Version 2010.1.*, 2008. Downloaded on 1 February 2010.

[85] Michael B. Smith, Dong-Jun Seo, Victor I. Koren, Seann M. Reed, Ziya Zhang, Qingyun Duan, Fekadu Moreda, and Shuzheng Cong. The distributed model intercomparison project (DMIP): motivation and experiment design. *Journal of Hydrology*, 298:4–26, October 2004.

[86] Dimitri P. Solomatine, Mahesh Maskey, and Durga Lal Shrestha. Instance-based learning compared to other data-driven methods in hydrological forecasting. *Hydrological Processes*, 22:275–287, 2008.

[87] Dimitri P. Solomatine and Michael B. Siek. Modular learning models in forecasting natural phenomena. *Neural Networks*, 19(2):215–224, 2006.

[88] Dimitri P. Solomatine and Yunpeng Xue. M5 model trees and neural networks: Application to flood forecasting in the upper reach of the Huai River in China. *Journal of Hydrologic Engineering*, 9(6):491–501, November/December 2004.

[89] Mohammad A. Talaat, Magdi A. Koutb, and Hoda S. Sorour. A survey on unicast congestion control protocols for media traffic. *International Journal of Computer Science and Network Security*, 9(3):254–261, March 2009.

[90] Jay Taneja, Jaein Jeong, and David Culler. Design, modeling, and capacity planning for micro-solar power sensor networks. In *Proceedings of the International Symposium on Information Processing in Sensor Networks (IPSN)*, pages 407–418. IEEE Computer Society, 2008.

[91] Bjørn Thorstensen, Tore Syversen, Trond-Are Bjørnvold, and Tron Walseth. Electronic shepherd - a low-cost, low-bandwidth, wireless network system. In *Proceedings of the International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 245–255, Boston, MA, USA, 2004. ACM Press.

[92] Gilman Tolle, Joseph Polastre, Robert Szewczyk, David Culler, Neil Turner, Kevin Tu, Stephen Burgess, Todd Dawson, Phil Buonadonna, David Gay, and Wei Hong. A macroscope in the redwoods. In *Proceedings of the International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 51–63, San Diego, CA, USA, 2005. ACM Press.

[93] tRIBS Development Team. tRIBS HydroMet data. `http://www.ees.nmt.edu/vivoni/tribs/weather.html`.

[94] United Nations International Strategy for Disaster Reduction. Disaster statistics 1991-2005. `http://www.unisdr.org/disaster-statistics/introduction.htm`.

[95] Charles Van Loan. The block Jacobi method for computing the singular value decomposition. In *Computational and Combinatorial Methods in Systems Theory*, pages 245–256. Elsevier Science Publishers B.V. (North-Holland), 1986.

[96] Iuliu Vasilescu, Carrick Detweiler, and Daniela Rus. AquaNodes: an underwater sensor network. In *Proceedings of the Workshop on Underwater Networks*, pages 85–88, Montreal, Quebec, Canada, 2007. ACM.

[97] Christopher M. Vigorito, Deepak Ganesan, and Andrew G. Barto. Adaptive control of duty cycling in energy-harvesting wireless sensor networks. In *Proceedings of the IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, pages 21–30, 2007.

[98] Jasper A. Vrugt, Breanndán Ó. Nualláin, Bruce A. Robinson, Willem Bouten, Stefan C. Dekker, and Peter M.A. Sloot. Application of parallel computing to stochastic parameter estimation in environmental models. *Computers and Geosciences*, 32:1139–1155, October 2006.

[99] Peter J. Webster and Robert Grossman. Forecasting river discharge into Bangladesh on short, medium and long time scales. *Climate Forecasting Applications in Bangladesh*, January 2003. Online at `http://cfab.eas.gatech.edu/cfab/Documents/InfoSheets/CFAB_forecast.pdf`.

[100] Geoff Werner-Allen, Konrad Lorincz, Jeff Johnson, Jonathan Lees, and Matt Welsh. Fidelity and yield in a volcano monitoring sensor network. In *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, pages 381–396, Seattle, WA, USA, 2006. USENIX Association.

[101] Pei Zhang, Christopher M. Sadler, Stephen A. Lyon, and Margaret Martonosi. Hardware design experiences in ZebraNet. In *Proceedings of the International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 227–238, Baltimore, MD, USA, 2004. ACM Press.

[102] Jing Zhou and David De Roure. FloodNet: coupling adaptive sampling with energy aware routing in a flood warning system. *Journal of Computer Science and Technology*, 22(1):121–130, 2007.