UNIVERSITY OF CALIFORNIA, SAN DIEGO

Enabling Rich Applications and Reliable Data Collection in Embedded Wireless Networks with Low-Footprint Devices

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Electrical Engineering (Computer Engineering)

by

Shoubhik Mukhopadhyay

Committee in charge:

Professor Sujit Dey, Co-Chair
Professor Curt Schurgers, Co-Chair
Professor William Hodgkiss
Professor Joseph Pasquale
Professor Mohan Trivedi

2009

The dissertation of Shoubhik Mukhopadhyay is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____
Co-Chair

_____
Co-Chair

University of California, San Diego

2009

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

There have been a large number of people whose support and guidance throughout my years at UCSD made this dissertation possible. I take this opportunity to express my sincere gratitude for all of them. Though I cannot hope to enumerate, let alone repay, all of whom I am indebted to, I still insist on naming a few in print.

The first people I would like to thank are the two whose vision guided my research and shaped this dissertation. I am grateful to my advisor, Prof. Sujit Dey, who has been a constant source of encouragement and support over these years. I thank him for being extremely patient with me, for teaching me how to conduct research and present ideas, and for being enthusiastic in venturing into new topics of research. I am also greatly indebted to my co-advisor, Prof. Curt Schurgers, for his endless hours of advice and guidance. His involvement in every aspect of my work, from discussing algorithms to shaping my writing, was a defining factor in ensuring the quality of the final manuscript.

In addition, I would like to thank Debashis Panigrahi, my collaborator and friend, for his contributions, insights and feedback, especially for the work presented in Chapter 2. I also thank the members of my thesis committee, Professors William Hodgkiss, Joseph Pasquale and Mohan Trivedi for providing valuable suggestions and feedback. This work has also improved in quality thanks to detailed feedback provided by the anonymous reviewers of the related conference and journal papers. I would like to thank Salil Pradhan and Malena Mesarina of HP Labs, and Kyle Jamieson and Hari Balakrishnan at MIT CSAIL, for contributing datasets for my experiments. I gratefully acknowledge the financial support provided by the Jacobs Engineering Fellowship, the UCSD Center for Wireless Communications and UC Discovery that made my doctoral studies possible.

Throughout my graduate studies, I have been fortunate to have the chance to work with a wonderful group of co-workers in the MESDAT lab at UCSD. I am very grateful to all the past and present members of our lab for providing an interesting and encouraging working experience. Our interactions and discussions, be they technical, political or personal, made the day-to-day graduate student life enjoyable. I also wish

to thank my fellow students in WISL, with whom I shared an office for a few months, for a great work environment.

I owe a special debt of gratitude to all my friends whose encouragement and moral support kept me sane for all these years and helped me bear the various ups and downs of graduate life. I take this opportunity to thank all the new friends who made me feel at home in a foreign land, and old ones who, often from across continents, made me think that I had never left home. The list of all who deserve mention is far too long to include here, but special thanks go to Niloy Mitra, Naomi Ramos, Anindya Patthak and Amit Saha for being incredibly patient listeners and providing valuable feedback on everything from research, writing and life. Finally, I cannot begin to properly thank Paulami Banerjee, who was there as a continual source of joy and steadfast support through the final stages of graduate school.

I dedicate this dissertation to my parents. They are the ones who taught me all the really important things in life, ensured that I got a good education against considerable odds, and were the first to believe in me. I would not have been able to dream of embarking upon this journey if it were not for their encouragement and sacrifices. And without their love and unfathomable patience, I would never have been able to complete it.

The text of the following chapters, in part or in full, is based on material that has been published in conference proceedings or journals, or is pending publication in journals, or is in review. Chapter 2 is based on material that has been published in the IEEE Global Communications Conference(Globecom) (S. Mukhopadhyay, C. Schurgers, S. Dey, "Joint Computation and Communication Scheduling to Enable Rich Mobile Applications", *Multimedia Communications, Software and Services Symposium at the IEEE Global Communications Conference*, Washington D.C., November 2007) and material submitted to the ACM Mobile Computing and Communications Review (S. Mukhopadhyay, C. Schurgers, S. Dey, "Enabling Rich Mobile Applications: Joint Computation and Communication Scheduling", *ACM Mobile Computing and Communications Review*). Chapter 3 is based on material that has been published in the IEEE Wireless Communications and Networking Conference in 2004 (S. Mukhopadhyay, D. Panigrahi, and S. Dey, "Data Aware, Low Cost Error Correction for Wireless Sensor Networks", *Proceedings of IEEE Wireless Communications and Networking Conference* , Atlanta, USA, March 2004), the IEEE Sensor and Ad Hoc Communications and Networks Conference(SECON), (S. Mukhopadhyay, D. Panigrahi, S. Dey, "Model Based Error Correction for Wireless Sensor Networks", *Proceedings of IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks*, Santa Clara, October 2004), and accepted for publication in the IEEE Transactions on Mobile Computing (S. Mukhopadhyay, C. Schurgers, D. Panigrahi, S. Dey, "Model Based Techniques for Data Reliability in Wireless Sensor Networks", *IEEE Transactions on Mobile Computing*). I was the primary researcher and author of each of the above publications, and the co-authors listed in these publications collaborated on, or supervised the research which forms the basis for these chapters.

VITA

1999	B.Tech., Electronics and Electrical Communications Engineering, Indian Institute of Technology, Kharagpur
2001–2008	Research Assistant, Dept. of Electrical and Computer Engineering, University of California, San Diego
2002	Summer Research Intern, HP Labs, Palo Alto, California
2004	M.S., Electrical Engineering (Computer Engineering), University of California, San Diego
2009	Ph.D., Electrical Engineering (Computer Engineering), University of California, San Diego

PUBLICATIONS

S. Mukhopadhyay, C. Schurgers, S. Dey, "Enabling Rich Mobile Applications: Joint Computation and Communication Scheduling", *ACM Mobile Computing and Communications Review* (in review).

S. Mukhopadhyay, C. Schurgers, D. Panigrahi, S. Dey, "Model Based Techniques for Data Reliability in Wireless Sensor Networks", accepted for publication in IEEE Transactions on Mobile Computing.

S. Mukhopadhyay, C. Schurgers, S. Dey, "Joint Computation and Communication Scheduling to Enable Rich Mobile Applications", in the Multimedia Communications, Software and Services Symposium at the IEEE Global Communications Conference (Globecom'07), Washington D.C., Nov. 2007 (*Best Paper Award*).

S. Mukhopadhyay, D. Panigrahi, S. Dey, "Model Based Error Correction for Wireless Sensor Networks", in The First IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON '04), pp. 575-584, Santa Clara, Oct. 2004.

S. Mukhopadhyay, D. Panigrahi, S. Dey, "Data Aware, Low Cost Error Correction for Wireless Sensor Networks", in IEEE Wireless Communications and Networking Conference (WCNC'04), pp. 2492-2497, Atlanta, March 2004.

ABSTRACT OF THE DISSERTATION

Enabling Rich Applications and Reliable Data Collection in Embedded Wireless Networks with Low-Footprint Devices

by

Shoubhik Mukhopadhyay
Doctor of Philosophy in Electrical Engineering (Computer Engineering)

University of California San Diego, 2009

Professor Sujit Dey, Co-Chair
Professor Curt Schurgers, Co-Chair

Following the remarkable developments in computing and wireless communication technologies, there has been a rapid proliferation of mobile and embedded computing systems and applications that are becoming ubiquitous in all aspects of modern life, including the enterprise world, the entertainment world as well as in common household appliances. However, designers of new generations of these systems have to address a twofold demand for increasing application functionality as well as a reduction in device footprints.

This dissertation proposes a design principle that can allow mobile computing applications to transcend the limitations on the end-devices, by splitting up the functionality of the application end points between the low-footprint wireless end-nodes and shared fixed nodes inside the network. It presents two instantiations of this idea in different types of wireless networks, identifying and solving some of the key challenges faced when applying the proposed principle. In the first part, the focus is on wireless access networks, where shared processing resources within the network can be used to support high-end applications on thin handheld clients. A major challenge here is to determine how to schedule the wireless communication and computing resources together to support a large set of clients. The second part of the dissertation demonstrates how the same design principle can be applied to wireless sensor net-

works to enable the use of simple, ultra-low-power sensor nodes. The resource limitations on the sensor nodes make it challenging to ensure the reliability of the sensor data, and a novel solution is proposed that leverages the correlation properties of the data to do so.

The experimental results presented demonstrate the viability of the proposed architectural principle in systems that vary markedly in terms of application goals and device capabilities. It is shown that the technique proposed in the first part can achieve very efficient scheduling performance with minimal processing overheads. This enables the expansion of the application functionality without increasing the footprint of the end-devices. Similarly, in sensor networks, the error correction method demonstrated the feasibility of achieving the primary functionality, i.e., reliable data collection even when the footprints of the sensor nodes are reduced too far to implement traditional reliability measures. The techniques described will facilitate the adoption of infrastructure-based approach to system design, leading to high-end applications using low-footprint devices.

Chapter 1

Introduction

In the recent decades, remarkable and steady developments in the fields of computing and communication technologies have ushered in an age of information technology, which has drastically altered the way we live, work, conduct business and communicate with each other. Many of the devices and applications that have made this change possible are results of a convergence between data processing and wireless communication capabilities. Examples of such emerging wireless applications and systems are abundant in both enterprise and consumer worlds, covering such diverse areas as personal communication, consumer electronics, medical technologies, navigation systems.

However, while it is the technological changes of the past that have enabled the present developments, it becomes increasingly hard for each successive generation of technology to continue the existing rates of growth, because the requirements for such growth impose conflicting requirements on the underlying systems. We believe that many of the future improvements will depend on innovations in the system architectures and application designs that make use of the technological achievements in new ways. In our work, we look at one such architectural idea and its potential to improve the capabilities of different types of wireless data processing and communication systems.

In this chapter, we first discuss the history of technological growth and convergence of computation and communication systems, and present a background on how

technology trends and application requirements have shaped the present generation of wireless computing systems. Next, we describe how an architectural approach based on modifying the design of the end-nodes and the applications can lead to substantial improvements in capabilities. Finally, we outline the contributions made by this thesis, in applying this proposed architectural approach to two types of wireless systems. We conclude with an overview of the remaining chapters.

1.1 Convergence of computing and communication technologies

The embedded wireless computing systems of the present day are a result of a convergence of computing and communication systems, which developed separately for much of their existence. We note that the convergence comes at the end of a long history of remarkable developments for both types of systems over the recent decades. Fig. 1.1 shows some of the milestones along this path of growth, and illustrates how they have gotten closer over time, leading to the close overlap of the present day.

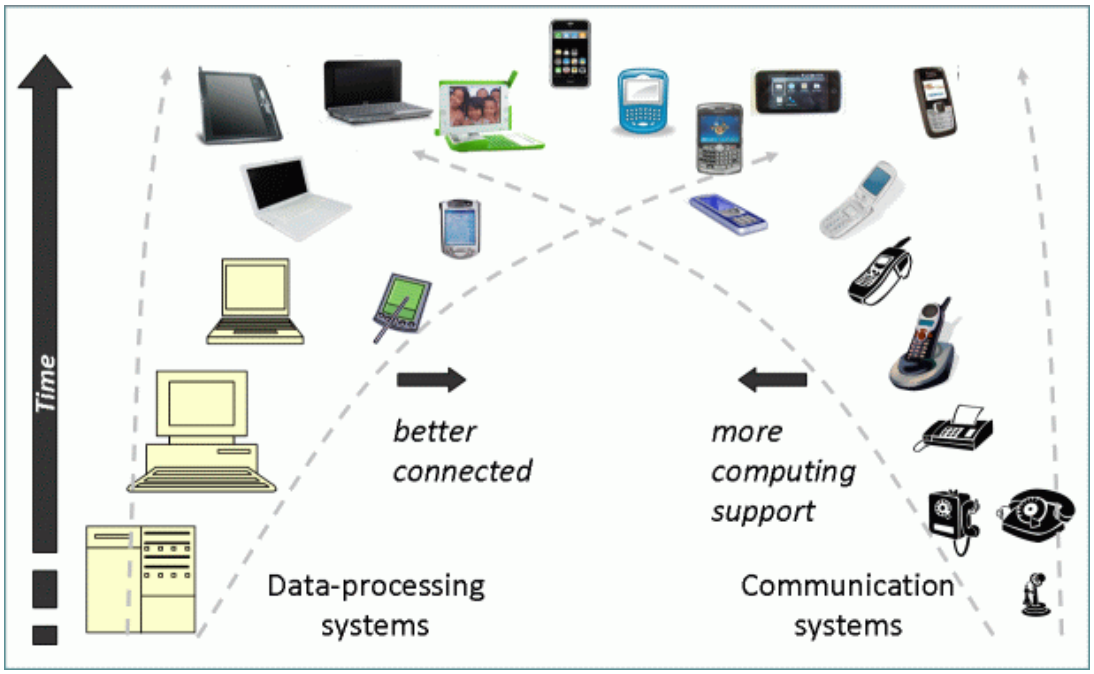


Figure 1.1: Convergence of data-processing and communication systems over time

Computing or data processing systems started off as centralized mainframes of early years, which were followed by desktop personal computers and workstations, and eventually by laptop computers that constitute the primary computing systems for most present day users. Along these stages of development, advancements in hardware design and fabrication technologies have enabled concentrating more and more computational power in smaller packages. This has produced an exponential growth in the density of computational power, as described famously by Moore's Law. Since the hardware was able to compress computing and storage into smaller spaces, the computing systems, *i.e.*, the data processing devices have also become progressively smaller and portable , even while they became more and more powerful. This is evident in the way that portable laptop computers have become capable of handling most common computing tasks over the last decade. Moreover, laptops have been followed by even smaller and lighter devices like Blackberry and PDAs, which are capable of many common computing tasks like email and basic web browsing.

Over the same period of time, there has been a parallel development in telecommunication systems and applications, starting from wired telephony, followed by analog cellular phones, and by three generations of digital cellphone technologies thereafter. During this development, the function of the communication system has evolved from basic setup and routing of voice calls, to narrowband data communication, and eventually to rich multimedia applications like video streaming over mobile broadband technologies. Along with the increasingly complex functionality and expanding bandwidth, the devices used to access these systems have also grown more and more complex and capable. For example, the computational capabilities of the CPUs in Smartphones of the present day have become comparable with the processors from the earlier generations of desktop computers. In summary, just as the computing devices became more portable with the development of computing systems, the end devices in communication systems have grown more complex.

As illustrated in Figure 1.1, both these trends have eventually led to an overlapping of roles between computational and communication devices in contemporary data processing and communication systems. The lines between these types of systems are getting blurred, *e.g.*, by the widespread use of computing devices for commu-

nication tasks (e.g. IM, VoIP and video chat), or by cellular phones and Smartphones that are used for data processing tasks like email or web browsing. This merging of roles is most conspicuous in the proliferation of multi-purpose portable devices that are equipped with both computing and communication capabilities to varying degrees. These types of crossover devices are now being used for functions as diverse as personal communication, media playback, navigational aids, as well as common data processing tasks.

The ubiquitous presence of these devices have also led to a corresponding growth in supporting network infrastructure, leading to higher availability and capacity of network connectivity. In the case of data processing systems, this has been in the form of widely available Wi-Fi hotspots, while for communication systems this has been in the wide and increasingly affordable support for cellular data services like EDGE,EVDO,etc. The availability of the network infrastructure has, in turn, encouraged the proliferation of such devices, resulting in a self-sustaining cycle of growth. The growing infrastructure has also encouraged many new applications of technology that take advantage of this convergence by embedding computation and communication capabilities in new applications and appliances, and inventing new models of usage. For example, digital cameras now often support wireless LAN connectivity, so that the user can share photos with anyone in the world immediately after taking them. Similarly, it is becoming feasible to supplant printed magazines with portable electronic book readers that can automatically synchronize with current newspaper and magazine issues over cellular data connections.

Even though phenomenal advancements have been made in embedded wireless networks, the cycle of growth is predicted to continue, as new applications get integrated into this types of devices, and the economy of scale continues to make them cheaper. To use an analogy with automobiles, the development of cars did not come to a standstill once they became capable of the basic functionality of transportation, and commoditized to the point where most people could own one. Instead, even after everyone had one, the demand for new functionalities and features, like safety, power and energy efficiency, has continued to grow. Similarly, as portable devices with wireless communication and computational capabilities become common, we

expect to see a demand for new features from these devices that continue to stretch their capabilities.

1.2 Directions of Development of Mobile Devices

The wireless computing and communication systems described above have emerged from the convergence of two different paths of technological development. In order to support and continue the phenomenal rates of growth as projected, successive generations of these systems will have to make progress along the design goals that were a characteristic of both types of systems in the past. One of these goals is the lowering of the footprint of the end devices, which consists of reducing their form factor and energy consumptions. The other is for more complex application features, which need higher computing power or high data rates in communication. However, these goals place contradictory requirements on the system, since the higher functionality also typically requires higher processing power on the end nodes. The only way to achieve both at the same time is through significant technological changes. For example, a change in a circuit design or semiconductor technology that can dramatically reduce the energy needed to perform a certain type of video compression could potentially enable real time video compression in our wristwatches. However, with each successive generation such technological changes become harder and more expensive achieve. Instead, the typical development of these applications and devices must explore the trade-off between requirements, the final choice depending on the relative importance of each goal.

The effect of this course of development is illustrated in Fig. 1.2, which shows how some examples of these devices compare in terms complexity and footprints, and are affected by improvements in technology. Each curve marked on the figure denotes each successive generation of technology, and different points on each curve represent the continuum of possible trade-offs in two dimensions. Typically, a new generation of technology or a new design enables more capabilities within the same footprint (*e.g.*, more powerful laptops) or a similar functionality within a reduced footprint (*e.g.*, sensor Motes [XBO] becoming smaller in each generation). To sustain the phe-

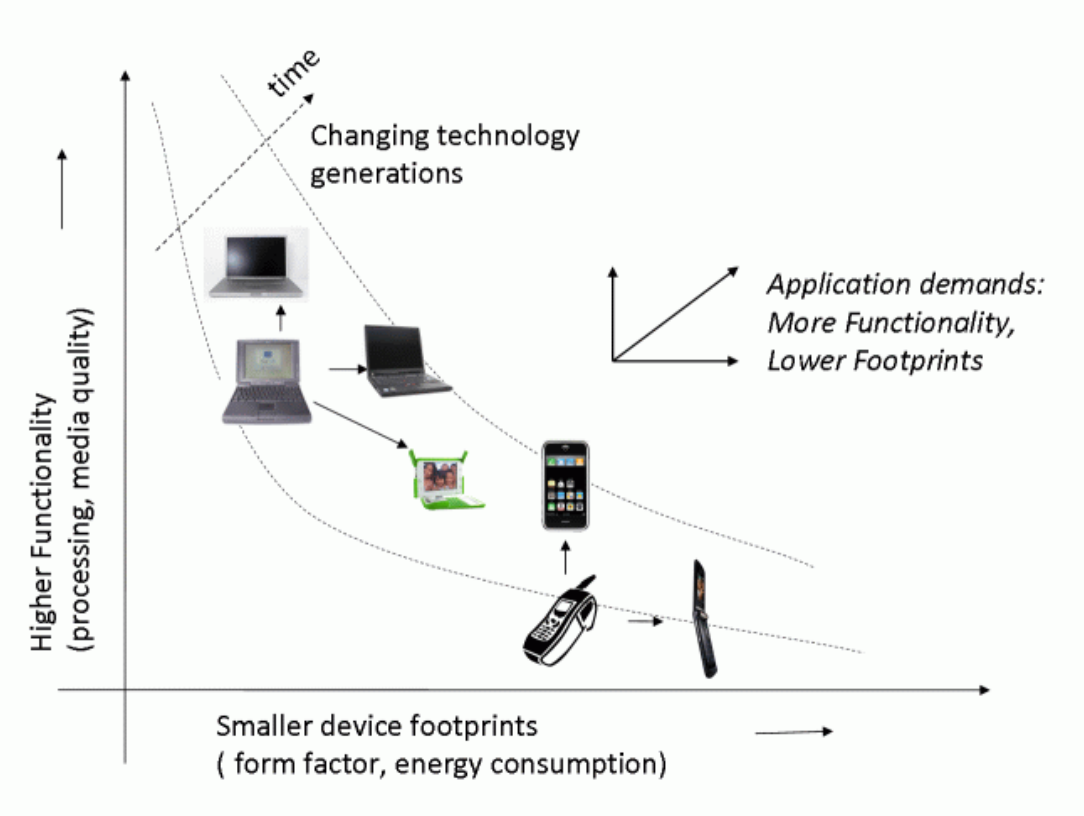


Figure 1.2: Development of mobile devices vs application demands

nomenal levels of growth promised by the convergence, it will be necessary for future systems to push the boundaries along both axes, for which it is not enough to depend solely on technological changes. Instead, we believe that it is possible to transcend the range of this trade-off through fundamental changes in network architecture and application design. Below, we look at this architectural idea in more detail.

1.3 Proposed Application Design Principle : Splitting End Nodes

In this dissertation, we propose an approach to effect such change through a revision of application design and network architecture. In broad terms, it consists of a redesign of the way the end nodes in a distributed wireless networked system implement the functionality of the user-related parts of the application. Before de-

scribing our approach, we first look at the corresponding approach used in traditional computing systems and communication systems.

Historically, applications for wireless networks have broadly conformed to the architecture depicted in Fig.1.3(a). The central characteristic of this architecture is that the functionality specific to the client, or the user, is concentrated at the end nodes, which can take the form of various types of devices. This architecture has been universally used in data processing systems as well as communication networks, differing only in types of devices used for the end nodes. For data processing applications, the end node typically consists of a laptop computer connecting to the network over some type of 802.11 wireless link. In this case, the applications would be identical to those designed for desktop computers connected to wired networks, with the end nodes being completely responsible for handling all the user-specific and application-specific processing. The role of the network is only to provide connectivity for data transfers with relatively simple semantics. Even communication functionalities specific to the user instance, like ensuring reliability, maintaining sessions or controlling data flows, are left to the end points. On the other hand, in wireless telecommunication systems, the end nodes are simpler devices, and the connection and application states are maintained by the network itself (Fig. 1.3(b)). Even with technological evolutions that have affected the processing capabilities as well as the communication bandwidths, the architectures for such systems have essentially remained same. For example as laptops became more and more portable, approaches like mobile IP were used to handle the connectivity issues within the network, changing nothing in the client architecture. Similarly, while the cellular data connections now support high bandwidth communication capable of handling many desktop applications, the application architecture has largely remained the same as before. So networked applications created for desktop computers have to be scaled down to match the reduced capabilities of the clients, which include CPU, battery as well as display size limitations.

The existing architecture has worked well, and will continue to do so even with the changing characteristics of network applications. However, the architecture also offers only a limited scope of a trade-off between the client functionality and the device capabilities of the end node. In this work, we present a modification of

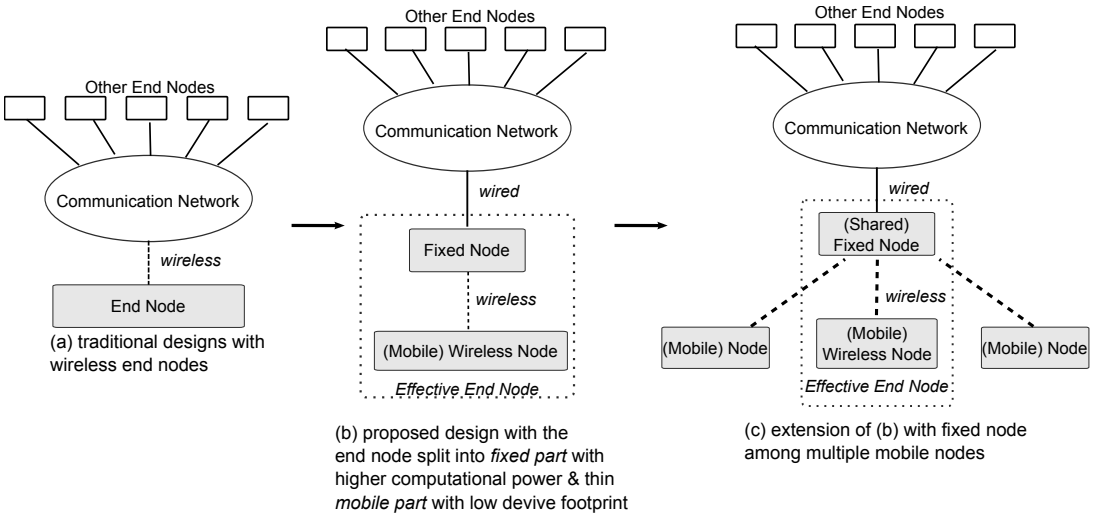


Figure 1.3: Proposed application architecture

this architecture as an way to handle new applications that helps us keep pushing the envelopes of higher functionality and lower device footprints. Our approach for handling both these demands together consists of restructuring the part of the applications that reside on the end node. As shown in Fig.1.3(b), the responsibilities of the end node are split among two different devices, marked here as the fixed node and mobile node. Now, the fixed node is decoupled from the footprint restrictions on the end node, and can support higher functionality. Similarly, the footprint of the mobile node can be further reduced, because the part of the application requiring higher processing at the client can be performed at the fixed node. While these two devices can remain tightly coupled, with a pair of such devices exclusively representing an end node, their physical location can be adapted to the network architecture and system requirements. For example, the mobile part can often benefit from being moved away from the network, only retaining a point-to-point link with the fixed node. Depending upon the application, it can be placed in close association with an user (in mobile access networks), or with a physical environment (for a sensor network). Another possible extension of this architecture is to push the fixed node deeper into the network, having it connect to multiple mobile nodes. Since the fixed node is relatively free from footprint restrictions, in some cases this may allow merging a few of the fixed nodes and having a number of mobile nodes share them. This approach is shown in Fig.1.3(c).

Since it ensures that the number of fixed nodes are not tied very closely to the number of mobile nodes, it enables a greater flexibility in the system design. We also note that sometimes there some such shared infrastructure is already present in a network, playing similar roles as the fixed node. This infrastructure may be extended to implement the fixed node, *e.g.*, in a Wireless LAN environment, it would be a simple solution to integrate the fixed node into the access points by co-locating general-purpose computers with them. Similarly, in a hierarchical sensor network, the fixed nodes can be be an extension of a clusterhead node that acts as a network gateway for multiple sensor nodes.

In our work, we show how the proposed architecture can allow diverse types of wireless computing and communication systems to increase application functionality and reduce device footprints at the same time. Various wireless applications, ranging from sensor networks to desktop-like rich applications like multimedia, have used the architecture of Figure 1.3(a). Since they put all the processing for the client on the end nodes, the client functionality is limited by the amount of processing that can be accommodated within the device constraints of the clients. Below, we show how splitting the end node allows this limitation to be bypassed in two applications with very different requirements and goals.

Consider the system of wireless LAN APs mentioned above, where the fixed nodes consist of powerful processors co-located with wireless access points, while the end nodes are low-footprint, mobile devices that act as network clients. In the traditional architecture of Figure 1.3(a), all the client-side processing in the application will be performed by the mobile nodes, so that the processing power in the clients would limit the richness and quality of the application. In the proposed architecture, the 'enhanced' access points can perform as application-layer gateways that provide processing support in addition to network access to the clients, opening up the scope of the system to enable support for rich multimedia mobile services. For example, this could allow the fixed nodes to render complex 3-dimensional graphics and forward the result to thin mobile handsets as video streams. In general, this architectural modification can allow a significant flexibility in balancing processing and communication resources against application requirements when designing mobile applications. Even

when the end nodes are capable of performing these tasks, this design allows the option of trading off bandwidth usage with battery life by allowing processor-intensive tasks to be offloaded to the access points.

In another type of network, for example, in certain types of sensor networks, this design would mean splitting up the functions of the sensor nodes into two parts. Common tasks in sensor nodes include reading data from sensors, and storing, processing and reporting the data, and many common architectures assign these diverse tasks to one type of nodes. However, using the architecture shown in Fig. 1.3(b), the sensing and data processing tasks can be partitioned among dedicated sensor nodes and other network nodes designed for the specific type of data processing. This would allow each of the sensor nodes to specialize in sensing certain types of data sources, so that they can be designed very efficiently, with very little software content. On the other hand, the fixed nodes can consist of complex devices with higher processing capabilities. These nodes can perform various data processing or routing tasks that can be implemented in software, allowing these devices to be reconfigurable and reusable across multiple applications.

In the following chapters, we look at both these types of applications and show how they benefit from the proposed restructuring of end nodes. We also address additional challenges that arise from adopting this architecture. In the case of enhanced wireless access points, this is the problem of sharing the fixed nodes among the end nodes, whereas in sensor networks, it is the problem of ensuring the reliability of the data when the sensor nodes are too thin to support powerful protection measures against errors. In both cases, we demonstrate how the result of addressing these challenges, can expand the boundaries of functionality and footprint. In the next section, we describe these two problems and present an overview of the remaining chapters, summarizing the contributions that this dissertation makes in addressing them.

1.4 Contributions and Overview

The main contributions made by this thesis are threefold. First, we present a network architecture for wireless networks that enables the expansion of application

functionality further than that permissible by end nodes. Second, for this architecture we have solved the problem of scheduling two resources, computation and communication, with complex interdependencies. Third, for wireless sensor networks using this architecture, we solve the problem of correcting non-linear errors in the data using a data-dependent framework.

For the scheduling problem, the novelty of our work is to take into consideration the complex effects of wireless links on the communication constraints when scheduling computation resources in a network. Wireless links have certain unique properties, e.g., location dependence and interference due to the use of shared channels. As a result, the effect of allocating one resource to a given user on other users is dependent on the positions of this user, as well as others in its neighborhood. Thus the scheduling problem we look at has unique spatio-temporal aspects, which have not been taken into account by existing work on processor scheduling, as we discuss later in related work.

In case of the reliability problem, there are two aspects to the novelty of this work: it addresses non-linear errors and does so in an adaptive manner. Our method performs error correction exclusively through post-processing, but it can handle random-bit-flipping errors that have non-linear effects, which cannot be handled by typical post-processing approaches (e.g. smoothing) that depend on linear filtering. Moreover, our approach includes run-time model adaptation, which allows the error correction system to keep up with variations in the data source by choosing different AR models as appropriate to the current data properties.

The main impact of the work on scheduling is that it merges the power of desktop computing with the portability and location-awareness of handheld devices. By decoupling the application functionality from the footprint limitations, it can enable high-end social networking and enterprise applications for mobile users. Moreover, by encouraging the development of applications that perform some of the processing within the infrastructure, it also makes the case for development of more wireless infrastructure side by side with the development of handsets.

For wireless sensor networks, the impact of this dissertation is to enable the use of ultra-miniaturized, simple sensor nodes for designing reliable applications.

This can in turn lead to many new applications that can be created from cheap off-the-shelf components instead of waiting for custom design of sensor nodes. In addition, the proposed architecture can use generic designs for the aggregator nodes irrespective of sensing applications, which would lead to a cost reduction through economy of scale.

In the remainder of this dissertation, we take a detailed look at two instances of the proposed architecture presented in Fig. 1.3 in two types of wireless networking applications. In the first part, we consider mobile networks with thin clients, and show that our architecture allows them to support rich applications that would otherwise require more powerful clients like laptops. In the second part, we focus on wireless sensor networks, where the primary application is reliable data collection. For such systems, we show that our approach allows the footprints of the sensor nodes to be substantially reduced without affecting the reliability of the data collection.

We present the first work in Chapter 2, where we focus on rich mobile applications that would normally require substantial data processing on the end nodes. We apply the idea for redesigning the application end-points to this context by having thin wireless handsets offload complex data processing tasks to fixed nodes. We propose an architecture where the fixed processing nodes are shared among multiple clients (as shown in Fig. 1.3(c)) in order to support a large and flexible number of clients. This sharing leads to the main challenge of such a network: to allocate the computing and communication resources in such a system effectively among a large number of heterogeneous clients. Our main contribution is to solve this problem of sharing, which we represent as a joint scheduling problem. We present efficient techniques for solving these problem with minimal system overheads.

In Chapter 3, we present our second work which focuses on data collection applications in wireless sensor networks. Here we look at an example of how the functionality of the system can be preserved or improved, while drastically reducing the footprint of the end nodes. Ensuring reliability of sensor data is one of the primary functions of a sensor network, and we show that reliable data collection is possible even with ultra-light, low-energy sensor nodes, by splitting some of the functionality among dedicated devices that either perform sensing or data aggregation. We present

a method that can ensure reliability by using the properties of sensor data, while performing all the processing necessary for this on the fixed aggregator nodes.

Chapter 2

Enabling Rich Mobile Applications: Joint Computation and Communication Scheduling

In the previous chapter, we presented an overview of the architectural principle of splitting the functionality of the end-nodes in wireless computing and communication systems. In this chapter, we demonstrate an application of this principle in increasing the functionality of mobile applications without requiring an increase in the footprint of the end nodes. We explore the joint resource scheduling problem that results from this approach, and present methods for ensuring efficient utilization of computing and communication resources in the system.

2.1 Introduction

The work presented in this chapter addresses the demand for increasing functionality in mobile applications, which has been a result of their rising popularity and widespread adoption. In today's world, mobile applications are desired to provide functionality that was previously possible only on fixed systems like desktop computers, gaming consoles, etc. However, we also want the client devices to simultaneously get thinner and lighter and have higher battery life. As discussed in Chapter 1,

these two goals are hard to pursue together, which leads to a gap between the desired functionality and the capabilities of the mobile devices to support them as achievable with the current technology. The result is a trade-off between these conflicting requirements. The alternatives consist of using low-footprint devices like PDAs, which would require scaling down the application functionality, or using laptops as clients, which limits the flexibility of the user. We propose that splitting the functionality of the end nodes, following the principle introduced in Chapter 1, presents an opportunity to implement complex application functionality while retaining the advantages of low-footprint end nodes. In other words, this attempts to effectively provide the power of a laptop in a smaller device.

To implement our approach, we also take into account the growth of fixed computing infrastructure, which has become more affordable and more widely available in the physical environment. We envision that the ubiquitous availability of this computing infrastructure can be leveraged to support the heavy computational tasks for the mobile applications, and use a thin-client design for the low-footprint part of the end nodes. In our work, we consider one implementation for this architecture where the computing support is built into the wireless access points, which could provide both computation and communication support for mobile applications, as illustrated in Figure 2.1.

This figure shows the use of this architecture for running rich mobile applications in various contexts. The physical environment is instrumented with 802.11-type Wireless LAN access points (APs), each of which is enhanced by a powerful general-purpose processor that may be physically integrated or co-located with the AP. Such a system can enable flexible and powerful mobile services, *e.g.*, video-conferencing systems that support streaming high-definition video from handheld devices. Using currently available architectures, this would require the terminals to support computationally intensive tasks like video compression and streaming. But in the proposed architecture, the clients could be thin devices like cellphones, which transfer these tasks to the enhanced access points over short range, high capacity wireless links. Such an approach can also be useful in network gaming. Here, high definition 3-D graphics could be brought to portable displays by performing certain rendering tasks

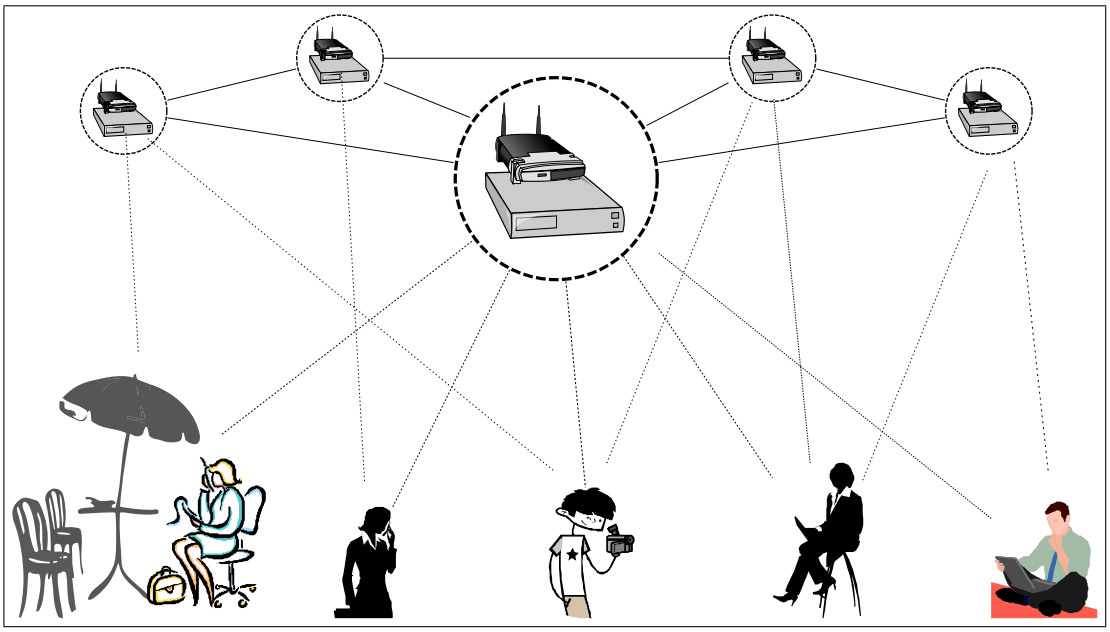


Figure 2.1: Network of enhanced wireless LAN with additional processing elements co-located with the access points.

at the access points.

Implementing the fixed part of the end node inside the network also enables it to be shared across multiple users. While this provides obvious advantages in terms of cost and scalability, it also introduces the complexity of managing the sharing of this resource among the users. Consider the network in Figure 2.1, where each mobile client may be in the coverage area of multiple access points and have multiple choices available for scheduling the application tasks. The processing capacity at each access point is a system resource that has to be shared among the client tasks. Similarly, the communication channels between the clients and the access points are also shared. When a client has a task with certain computational and communication resource requirements, the choice of which access point to use will have to be made taking into account both these resource requirements. This gives rise to a joint computation and communication scheduling problem, which forms the focus of this chapter.

Joint Scheduling Problem

The objective of our joint scheduling problem is to find the best way of sharing the computation and communication resources in the network. This translates to finding the best choice of access point (AP) for each client. For the overall system, the best choice can be defined as that which minimizes fragmentation of the resources, and allows usage of the system resources by the maximum number of clients. So the best assignment for each client will depend on the positions of the other APs and clients, and their choice of assignments as well. For example, consider a handheld client C that needs to off-load a processing task to a nearby AP. Suppose it sends out a broadcast query and finds two APs, A and B, that offer adequate data rates and processing capacity for this task, with A being nearer and supporting a higher data rate. Now, from the point of view of C, the best choice of AP for C could be A due to the higher data rate. But from the system's point of view, B might turn out to be a better choice if there are more clients contending for the process resources at A. On the other hand, if C has to communicate with B at a lower data rate, it will end up holding the communication channel longer to transmit the same amount of data. So depending on the presence and requirements of other clients, the best choice for the system could differ.

In a real application network, when the scheduler is computing the best set of assignments for the system, the clients will be waiting to begin executing the tasks. So a practical requirement of the scheduling is that it has to be completed within a very short time, on the order of milliseconds. Moreover, since the scheduler shares the processing resources on the APs, its computation requirements also need to be limited. On the other hand, finding the best overall assignment is a complex problem, as will be discussed in Section 2.5. There could be multiple feasible assignments, which have to be evaluated and compared to pick the best one. But finding the optimal solution by evaluating all options can be very slow. This is illustrated in Fig. 2.2 for a simplified version of the problem that ignores interference constraints. The figure shows how the running time of the optimal scheduler varies with the number of clients for three different sizes of the system that have 6, 8 and 10 access points respectively.

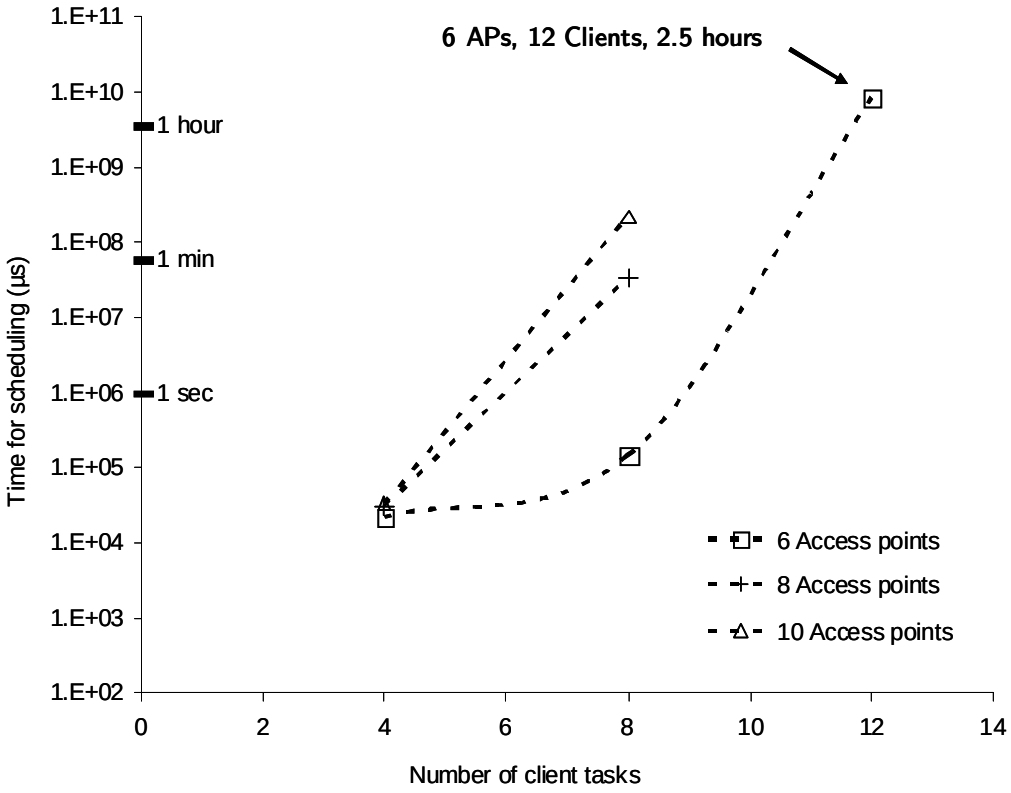


Figure 2.2: Running times for Optimal scheduler

It can be observed that the optimal scheduler can take hours to run even on relatively small loads of 10-12 clients on a network of 6 APs. This plot will be discussed in more detail with comparisons to our proposed scheduler in Section 2.7.4.

Our goal is to find a practical solution that satisfies the running time requirements as well as schedule the most number of clients possible. There are two main contributions to our work presented in in this paper: identifying the joint computation and communication scheduling problem, and developing a heuristic-based solution that is practical and efficient. In the rest of the paper, we present the problem of scheduling a set of tasks with specific resource requirements in the context of the system shown in Figure 2.1. Using simple per-node schedulability tests, we formulate the task allocation problem in an optimization framework, and develop a polynomial-time heuristic for an approximate solution. We analyze the performance of our ap-

proach and demonstrate through computational models that our approximate solution performs better than two other approaches. We then validate our models through packet-level simulation, and also show that the scheduling performance approaches the optimal, while managing to execute three or more orders of magnitude faster.

2.2 Related Work

In this chapter, we present a solution for the problem of scheduling computation tasks on a set of processing devices connected through wireless communication links. Previous work in scheduling of computation tasks on multiple processor systems has primarily been focused only on the computation resource constraint. While these works have taken into account communication overheads due to inter-processor communication, they have primarily used wired interconnects. On the other hand, wireless communication scheduling has been an area of extensive research, which has focused on the efficient sharing of communication resources among multiple users under various constraints and objectives like throughput and energy requirements. The communication scheduling techniques cannot be applied to scheduling of computation tasks. Below, we first present a comparison with the existing research on processor scheduling, followed by that on wireless communication scheduling.

The problem of scheduling a set of processing tasks over multiple resources has been studied in detail in the context of multi-processor scheduling in distributed systems, grid computing as well as in computer architecture [RSZ89][CM01][LS98]. In these problems, the primary focus is on minimizing the average delay and developing approximate algorithms that approach the optimal solutions with bounded error. However, these formulations are different from our problem because they use simplified models for the communication links that do not account for their variable quality or location dependence. On the other hand, the problem examined in this dissertation makes use of more realistic communication resource models that considers the variability of wireless links, taking into account location-dependence and interference effects. One related group of work in processor scheduling is on designing schedulability tests, where it has been shown that complexity of the test depends on the

homogeneity of the tasks [MB05][Bak05]. We refer to some of these tests as in our scheduling model to check schedulability locally on each processing node.

Similarly, managing shared access of nodes to the communication channel is one of the primary problems in wireless networks. Therefore, communication scheduling has been one of the main areas of research, particularly in the design of MAC protocols. Various wireless schedulers have been designed to optimize for different goals like capacity, fairness and stability under various channel models [VDGB05] [LK03][TS02] [JCOB02]. Moreover, various types of joint scheduling problems have been studied in wireless networks [EE02][CS03], which have included joint optimization of different resources and criteria, like power, congestion control as well as routing. However, this set of solutions is orthogonal to our problem, since none of them include processor scheduling among the primary goal.

Finally, there is one work that has studied the problem of processor scheduling in an ad-hoc network with wireless communication links and mobile client nodes [ACG+06]. Here, the authors consider the effect of node positions on the connectivity between the nodes. However, they model the communication links using a simple binary model based on the transmission range. This model does not fully capture the location-dependence characteristics of real wireless links, which adapt their modulation schemes and transmission data rates with signal strength, leading to a complex dependence of link throughput with distance. The throughput is also affected by interference from other nodes transmitting on same channels within a particular range. These properties have complex effects on how the allocation of a particular resource to a node affects the remaining resources. In our work, we take into account these realistic properties of wireless links by using throughput vs. distance measurements made in 802.11 networks and modeling the channel-access time as a first order constraint.

In summary, the novelty of our work lies in the ability to account for the properties of wireless communication links in a realistic joint-scheduling formulation, which also considers the interdependencies resulting from the interaction of processing constraints with communication resource constraints.

2.3 Problem Formulation

In this section, we develop a formal definition of the joint computation and communication scheduling problem. We first present our system model and assumptions, and then present a formal definition of the problem based on this model.

2.3.1 System Model

Our system consists of a network of enhanced wireless access points (APs), which provide connectivity and computational support to a set of mobile clients for running networked applications. The APs are stationary, identical in processing capacities, each operating on a fixed single frequency channel. High capacity wired links like Ethernet connect the APs to a backbone and the Internet. This network of APs needs to support a set of clients attempting to run application tasks that require network connectivity as well as computational support from the APs. The clients are mobile and can enter or leave the network at random times, which results in a changing set of task requirements.

Upon entering the network, each client sends requests for resources to all access points in its vicinity. Each request includes a processing capacity requirement, which represents the processing resources necessary to run the task on the chosen AP. Every request also includes a communication requirement, which is the bandwidth needed for this task between the client and the AP. The resource requests from all the clients are collected by a scheduler program that computes a new schedule, attempting to assign each client task to one of the APs. For each of the tasks that can be scheduled, both the client and AP are notified. The remaining tasks are either added to a list to be processed later, or discarded if their maximum waiting periods are exceeded. The scheduler is run periodically, to account for the changes in the load due to mobile clients entering and leaving the system. The frequency of repeated runs depends on the scheduling time for each run, as we will see later. For simplicity, we only consider clients trying to schedule one task each. However, multiple independent tasks per client can be easily handled by replacing ‘client’ with ‘task’ in the following discussion.

We also assume that it is not possible to split the offloaded processing for a task among multiple APs. Certain applications may contain function blocks which can be executed in parallel. While it may be feasible to split such applications, allowing it does not reduce the complexity of the problem unless an arbitrarily large number of splits were possible, as we show later. Therefore this assumption can be included without sacrificing generality.

The overall goal of the scheduler is to maximize the total number of clients supported by the system under two main constraints: (a) processing resources at each AP and (b) shared communication channel for the clients connected to each AP. In our model, the processing resources on all the APs are identical, and the processing tasks are periodic with identical time periods. In this case, simple additive schedulability tests [Bak05] can be used to test the processing resource constraint for each AP. However, equivalent tests exist for loads with unequal periods, and can be easily included.

On the other hand, the communication resource constraint depends on the link quality, which is different for each wireless link between a client and an AP node. The link quality for each pair of nodes depends not only on the distance between them, but is also affected by interference from other transmitting sources on the same channel that are located close enough. Therefore, evaluating this constraint is more complex due to the interdependence among APs. In our system, a large number of APs operate on a limited number of orthogonal channels, so for any AP, there are likely to be others transmitting on the same channel who are in a position to interfere.

While estimating the effect of interference is a complex problem requiring detailed knowledge of the signal strengths, our scheduler uses a simplified model based on interference range to identify potentially interfering sets of nodes. For each group of interfering nodes, the scheduler attempts to ensure that enough time is available on the communication channel to accommodate all potentially interfering sources, if the MAC can avoid them transmitting simultaneously. Whether or not collisions actually occur depends on the collision avoidance mechanism of the MAC, and in Section 2.7, we will use simulations to verify how well our scheduler can work with the non-ideal collision-avoidance mechanism in 802.11. Our computation of the interference

range (I) is based on the common model $I = 2R$, where R represents the maximum transmission range. In our system, while the locations of the APs are known by the scheduler, we assume that it does not have any information about the positions of the clients. So, the scheduler uses a worst-case estimate, assuming that the set of interfering sources within the interference range includes all nodes connected to APs that share the same channel.

To account for the effect of multiple users sharing the resources, we model the communication resource constraint as the *channel occupancy time* for each AP. This is a fairly common model that takes into account path-loss, channel sharing and interference effects. Among the total channel occupancy time of 100 %, each client consumes a time proportional to the ratio of its bandwidth requirement to its link capacity. By capacity, we mean maximum throughput for the purposes of this section.

2.3.2 Formal Definition

Based on the model of the user requirements and system resources described above, we now present a formal definition of the joint scheduling problem. We consider a set of independent application tasks that need to be scheduled on a given network of stationary enhanced access points. The network includes m APs and n clients, which are organized into ordered sets M and N respectively. The available processing capacities of the APs are represented as a m-dimensional vector $\mathbf{c}$, and the position and operating channel for each are known. A matrix F is used to represent an interference graph computed on the basis of the positions and channel assignment information for the APs. The graph contains one node for each AP, and includes edge (i, k), *i.e.* $f_{ik} = 1$, if and only if APs i and k are within interfering distance of each other $(i, k \in M)$ and are on the same channel.

The processing and communication requirements for the tasks are known at the time of placing the scheduling requests, and are represented as n-dimensional vectors $\mathbf{p}$ and $\mathbf{r}$. The path-loss bound link capacities are represented by a vector $\mathbf{l}$ of size mn such that $l_{i+m(j-1)}$ represents the capacity (*i.e.*, maximum throughput) of the link from client i to AP j, $\forall i \in M, j \in N$. We define an $mn \times 1$ assignment vector $\mathbf{x}$

and related assignment function $\mathrm{x}^i(j) \equiv x_{i+m(j-1))}$, such that $\mathrm{x}^i(j)$ is 1 when client j is assigned to AP i and 0 otherwise. We represent the channel access time of the link between client j and AP i as $\tau_j^i = r_j/l_{i+m(j-1)}$. The various constraints can then be formalized as follows:

1. $\sum_{j=1}^{n} \mathrm{x}^i(j) p_j \leqslant c_i, \forall i \in M$, denotes the CPU capacity constraint for each of the APs.

2. $\sum_{j=1}^{n} \mathrm{x}^i(j)\tau_j^i + \sum_{k \neq i} f_{ik} \sum_{j=1}^{n} \mathrm{x}^k(j)\tau_j^k \leqslant 1, \forall i \in M$, represents the constraint on the channel access capacity of each AP in presence of interference.

3. $\sum_{i=1}^{m} \mathrm{x}^i(j) \leqslant 1, \forall j \in N$, denotes that each client can be assigned to a maximum of one AP.

4. $\mathrm{x}^i(j) \in \{0,1\}^{mn}, (\forall i, j)$ denotes that a task cannot be split across different APs, and can be either assigned to one or refused access.

The first three sets consist of $2m + n$ individual constraints, all in the 'less than' form, so they can be summarized into a single matrix inequality $\mathbf{Ax} \leqslant \mathbf{b}$, where the dimensions of $\mathbf{A}$ are $(2m + n) \times mn$. Under these constraints, the objective of the scheduler is to find an assignment that is closest to the total scheduling request for the complete set of clients. This is represented in the objective function: $\max_{\mathbf{x}} \sum_{j=1}^{n} (w_j \sum_{i=1}^{m} \mathrm{x}^i(j))$.

This function also defines the criteria for choosing a solution that satisfies only a partial set of requests. When all the clients are identical, the goal is to maximize the number of clients in the total. There is also a weight assigned to each client, $\mathbf{w} \in \Re^n$ representing the total weight vector. The weights can be used to represent some system-specific satisfaction score, in order to separate clients based on service classes.

2.4 Optimal Solutions

In Section 2.3.2, we presented the joint scheduling problem as an optimization problem with four sets of constraints. While the first three sets of constraints are linear, the assumption that tasks cannot be split across multiple APs gives rise to the integer constraints $x \in \{0,1\}^{mn}$. This results in the well known 0-1 Integer

Programming form, which is known to be NP-hard in the general case [Ber99]. Alternately, the problem of finding the optimal schedule can be cast as a K-constraint Multiple-Knapsack problem with K=2 (2-MKP) which is NP-hard in the strong sense [AC05]. It can be demonstrated that the stated problem is NP-hard as well, and cannot be efficiently solved or approximated. The approach most common in literature is of combinatorial optimization algorithms that make systematic searches throughout the whole solution space [Ber99]. In this problem, since each of the n clients can be assigned to one or none of the m APs, the number of possible solutions is $(m+1)^n$. An optimal solution is one that either includes all clients, or failing that, maximizes the objective function over a subset of them. Even for moderate sized systems, the solution space can thus grow to huge sizes that can take hours to search, *e.g.* with 20 clients and 6 APs the number of solutions is on the order of 10^{17}. If each test takes 3 instructions on a 3GHz CPU, this can take up to eleven days even with 100 searches running in parallel. For example, in the measurements shown earlier in Fig. 2.2, it took more than two hours to schedule 12 clients on 6 APs using a 3GHz CPU. However, in order to be useful in handling incoming clients, the scheduler will need to execute frequently, and find a quick solution within a few seconds. Therefore, optimal combinatorial algorithms cannot be applied here.

One possible approach could be based upon relaxation of the unsplittable tasks requirement. In some instances, it may be possible to split an application into independent tasks and run them in parallel. For such cases, the problem may be reduced to a simple linear programming(LP) form, which can be solved optimally using standard methods like Simplex [Ber99]. However, infinitely splittable tasks are an unrealistic assumption for most applications of interest. For example, in real-time applications, if the communication links from a client to different APs differ significantly, a large part of the available processing time for each unit of data will be consumed in the overheads of buffering and coordinating the splitting and merging of data. Moreover, in order for this approach to be successful, it is necessary to be able to split the application into arbitrary parts for allocation. Even for applications containing sections that can be executed in parallel, this is not feasible. With a limited number of separable parts, the relaxed problem still remains a combinatorial optimization problem of

similar difficulty as the original problem.

2.5 Approaches for Solution

As we discussed in the last section, it is infeasible to solve the joint scheduling problem optimally under the time constraints typical to our system. In this section, we propose a heuristic for a scheduler (FJRS) that efficiently generates an approximate solution under tight time constraints. Before describing our heuristic, we first look at another common approach for such problems, which is to relax the integer constraints, solve the resulting LP problem, and project the solution back to the original variable space. For this part, we used a simplified version of the problem that assumed the availability of enough non-overlapping communication channels to assign an unique channel to each AP. This assumption simplifies the communication constraint because interference can be ignored. However, as we show later in Section 2.7, this approach performs poorly compared to our heuristic even for the simplified case, so we did not extend it further for the complete problem.

2.5.1 LP Reduction

This approach is a heuristic based on the LP relaxation discussed in the last section. The solution of the relaxation is used as a starting point to guess an approximate solution, as illustrated in Figure 2.3. Beginning from the original formulation, the three linear constraints are collected into a $(2n + m) \times mn$ constraint matrix $\mathbf{A}$, and the integrality constraint on $\mathbf{x}$ is relaxed to generate the LP problem. This is solved using a standard method, and the resulting solution $\hat{\mathbf{x}}$ is analyzed and converted to an integer solution $\mathbf{x^{int}}$ in the following way: The assignment vector $\hat{\mathbf{x}}$ is mn-dimensional, consisting of n consecutive blocks of m elements each. Each such block denotes the assignment of a particular client. We consider each m-element block in the LP solution $\hat{\mathbf{x}}$, find the maximum non-zero element in it and set it to 1, while setting the other $m - 1$ elements to 0. The resulting $\mathbf{x^{int}}$ satisfies constraints 3 and 4 described in the formulation and is then tested for the other constraints. If it

fails any of the first two, the size of the solution is decremented by setting one element in $\mathbf{x}^{\mathbf{int}}$ to 0. This element is selected by looking at which constraints were violated, and to which APs they relate. Among the clients assigned to this subset of APs, we find the one whose corresponding element was the minimum in the LP solution $\hat{\mathbf{x}}$. The resulting solution is tested again, and this process iterated until either a solution is found to satisfy the constraint, or until $\mathbf{x}^{\mathbf{int}}$ reaches zero.

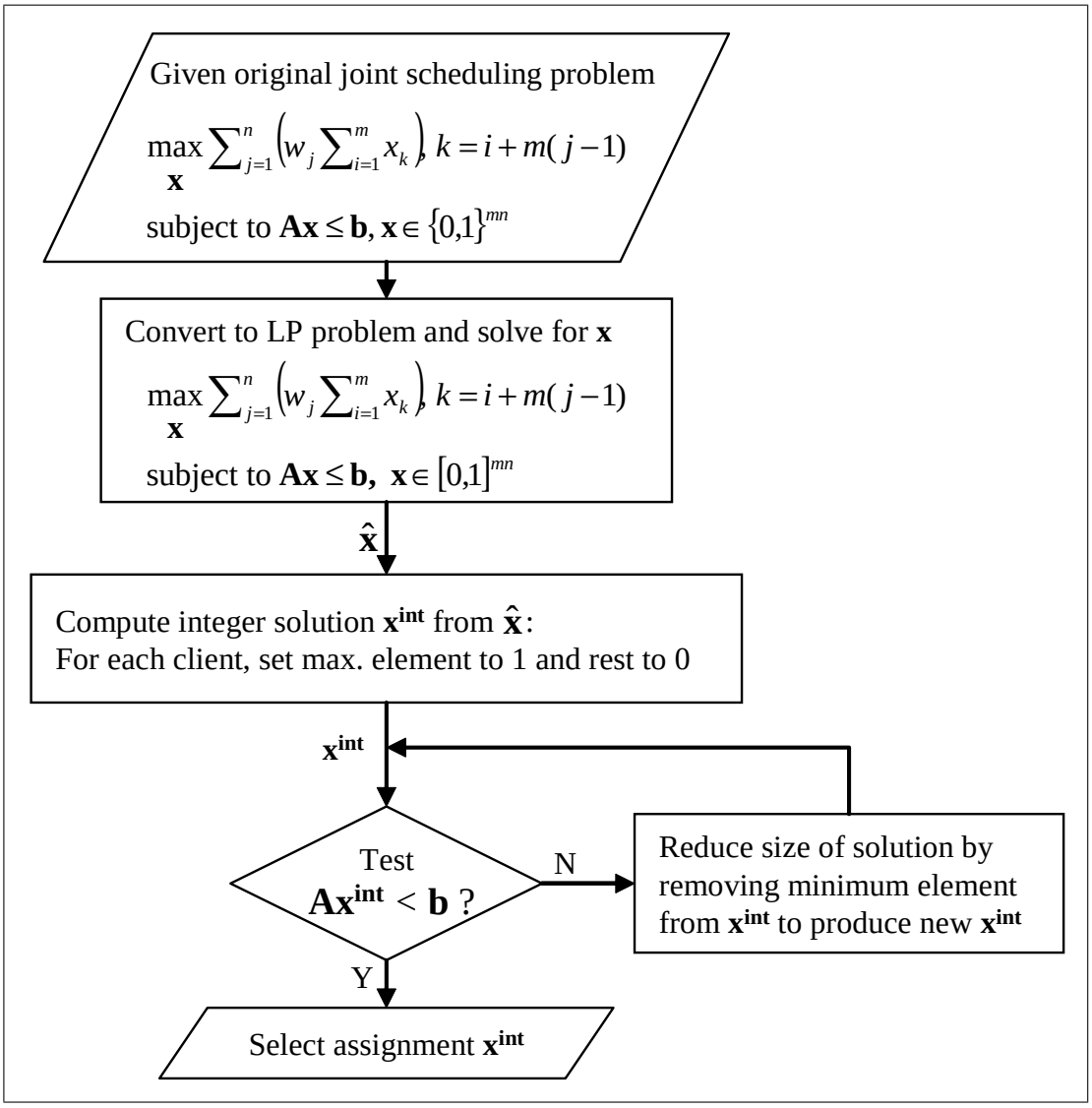


Figure 2.3: Flowchart for LP-based heuristic

The LP-based heuristic is fast because we avoid any backtracking, so the number of iterations is limited to n. Also, the steps that are repeated in the loop only

consist of testing the solution, which is a multiplication of two sparse matrices and can be executed quickly. The main step is solving the LP problem, which contains $2m + n$ linear constraints and can be solved efficiently through the Simplex method. On the other hand, the starting solution obtained from the LP may be distant from the optimal integer solution, and if there are very few feasible solutions it is possible to miss them all because the search path is limited to n steps.

2.6 FJRS Heuristic

Here, we present our approach, called Fast Joint Resource Scheduler (FJRS), which can achieve high scheduling performance while running at high speeds. To construct our solution, we first develop the approach to solve a special case of the problem, generated with an additional assumption. We then construct the heuristic for the more general case based on this solution. We begin with the special case where the resource demands from all clients are identical, and normalize the corresponding levels of available resources with respect to them, *i.e.* $r_j = 1$ and $p_j = 1$, $\forall j \in 1, \ldots, n$. The sequence of steps followed from here is shown in Fig. 2.4.

The algorithm is based on a greedy selection of up to n links from the set of all links. We begin by composing a list $Edgelist$ of all the client-AP edges with non-zero link capacity (l_{ij}) values, so that for any edge $e \equiv (i, j)$, $Edgelist[e].lcap \leftarrow l_{ij}$. We then annotate each entry in the list with a field $minlcap$ that keeps track of the client with minimum link capacity that can connect to the same AP, *i.e.* for any edge $e \equiv (i, j)$, $Edgelist(e).minlcap \leftarrow \min_{\hat{j} \neq j} l_{i\hat{j}}$. Now, $Edgelist$ is sorted in a descending order, using the rules shown in Algorithm 2.

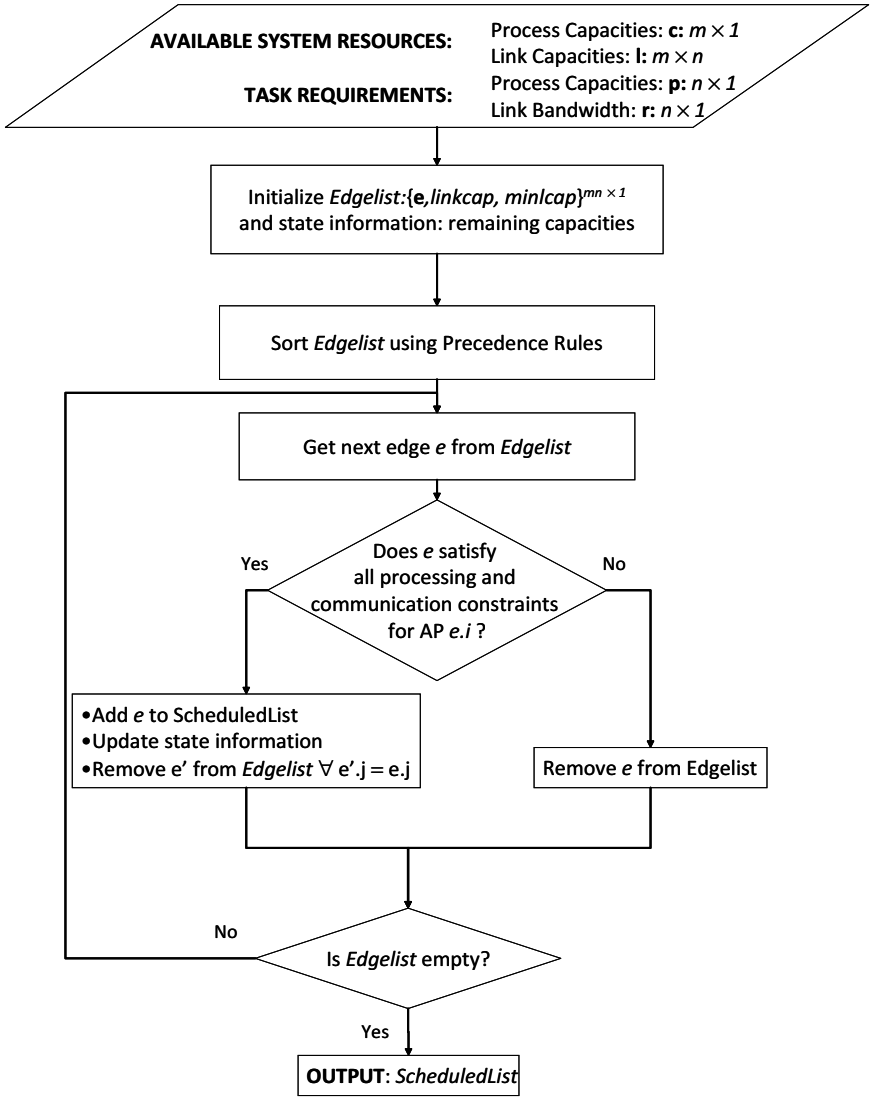


Figure 2.4: Flowchart for FJRS heuristic

Algorithm 1 Determining Priorities Among Precedence Rules for Sorting Edgelist

```
Let i1=e1.i, i2=e2.i
    lcap1 = Edgelist[e1].lcap,  lcap2 = Edgelist[e2].lcap
    MR_P=max_ratio(p1, p2)
    MR_Pcap=max_ratio(ProcCap[c1.i], ProcCap[c2.i])
    MR_L = max_ratio(lcap1, lcap2)

if i1 == i2 then                    //same AP
    if MR_P > MR_L then
        sorting_order: p, lcap, minlcap
    else
        sorting_order:  lcap, minlcap, p
    end if
else if q(i1,i2) == 1 then          //interfering APs
    if MR_Pcap > MR_L then
        sorting_order: ProcCap, lcap, minlcap
    else
        sorting_order:  lcap, minlcap, ProcCap
    end if
else if MR_P > 2 then
        sorting_order: ProcCap,lcap,minlcap
else
        sorting_order:  lcap, minlcap, ProcCap
end if
```

Algorithm 2 Precedence Rules for Sorting Edgelist in FJRS Heuristic

Let Example Sorting Order = (lcap, minlcap, ProcCap)

if $Edgelist[e1].lcap \neq Edgelist[e2].lcap$ **then**

$Edgelist[e1].lcap > Edgelist[e2].lcap \Rightarrow e1 \prec e2$

$Edgelist[e1].lcap < Edgelist[e2].lcap \Rightarrow e1 \succ e2$

else

if $Edgelist[e1].minlcap \neq Edgelist.minlcap$ **then**

$Edgelist[e1].minlcap > Edgelist[e2].minlcap \Rightarrow e1 \prec e2$

$Edgelist[e1].minlcap < Edgelist[e2].minlcap \Rightarrow e1 \succ e2$

else

$ProcCap[c1.i] \leq ProcCap[c2.i] \Rightarrow e1 \preceq e2$

$ProcCap[c1.i] > ProcCap[c2.i] \Rightarrow e1 \succ e2$

end if

end if

Once $Edgelist$ is sorted, the scheduler removes the top edge, including it in the solution set if it satisfies the remaining capacity constraints, or just discarding it

otherwise. If including the edge, the scheduler also removes all the other edges with the same client from the remaining $Edgelist$, and updates a set of state variables that keep track of the remaining resource constraints.

The main idea behind FJRS is that in general, given all r_j-s being constant, the links with higher link capacities have the least impact on the channel access capacity of its corresponding AP. Among two edges with the same link capacity, the client with the least number of alternative options should be assigned first. The processing capacity affects other nodes the least, so it is only used as a final check. However, if the difference between the processing requirements is exceedingly high, the remaining processing capacity in the two APs take a higher precedence. Apart from this general case, there are special cases when the links to be compared can potentially interfere with each other, or are connected to the same access point. In these cases, the requirements and available capacities for processing resources have a bigger role. For links going to the same AP, either the processing requirements or the link capacities may take higher precedence, depending on which of these parameters differs more between the two links. Similarly, when the two links are on different APs that can potentially interfere, instead of the processing requirements, the remaining processing capacities are considered. Here, Algorithm 1 shows how the relative resource requirements are used to determine the priorities among the sorting rules. Algorithm 2 illustrates how a particular ordering of these priorities is translated to precedence rules for the sorting engine.

The running time is the total time needed over three steps: annotation of Edgelist with $minlcap$ values, sorting the list, and the final pass through the list where the remaining list is scanned at each step. The annotation makes one pass through the full list to find the minimum edge for each AP (O(mn)), the actual setting of the $minlcap$ field being done when each edge is accessed. For the sorting, we used Quicksort which has an complexity of O($mnlog(mn)$) in the average case, but other sorting methods can be used for better worst-case performance. In the final step, as the size of the remaining list gets reduced by either n or one after each step depending on whether the top entry is selected or not, the complexity can be shown to be O(m^2n). The overall time complexity is thus dominated by the last step, which has a polyno-

mial time complexity of O(m^2n). The space requirement is dominated by the list of edges, which can be O(mn).

This heuristic can now be expanded to support the general case where clients can have different p and r values. First, the primary factor used to sort $Edgelist$ is changed to l_{ij}/r_j instead of l_{ij}. The $minlcap$ field is redefined to keep track of the client with the minimum l_{ij}/r_j. Also, the checks for remaining processor and channel access capacity are done through comparisons with p_j and $\frac{1}{l_j/r_{ij}}$ respectively. Although the objective function does not need to change, the weights $\mathbf{w}$ can be adjusted to account for any additional preference for specific clients without loss of generality.

It is interesting to compare the design of this algorithm with the original problem, which was cast as a problem of selecting one out of $m+1$ values for each of n clients, and thus had $(m+1)^n$ possible solutions. Here, we have redefined the problem in a less restricted form. The number of possible solutions are now $\sum_{k=1}^{n} \binom{mn}{k}$, since we are picking out a subset of up to n links from a set of size mn. However, structuring the solution set this way allows us to order them according to the impact on overall solution, and reach the solution in fewer steps.

2.7 Evaluation

We evaluate our approach in two ways. In the first part, we test the effectiveness of our scheduler in utilizing the available computation and communication capacity in the system. Given a particular set of scheduling problems, we look at how many client tasks can be scheduled by our scheduler, in comparison to simpler scheduling algorithms. Next, we wish to see, given a schedule or mapping of tasks to APs, whether we actually meet the demands for the tasks. We do this using a packet-level simulator that provides a close verification of the communication channel occupancy constraint.

2.7.1 Simulation Setup

In the first set of experiments, we measured the number of clients that can be scheduled by our scheduler under different network configurations and client loads. We implemented our scheduler in C, and evaluated its performance over a network scenario similar to the ones shown in Fig. 2.1. While our solution is general, for evaluation we pick a specific realistic scenario that determines the type of processing and communication resources in the network as well as the requirements of the client tasks. We first present a set of results for a simplified version of the problem that ignores the interference constraint, comparing the performance of FJRS with that of the optimal scheduler and the LP heuristic. For the simulations on the complete problem presented in the later part, it was not feasible to compare with the optimal scheduler due to extremely high running times for each run. Therefore, we implemented two other scheduling approaches for comparison: one that allocates each client to the nearest access point, and a second that chooses one access point at random among the ones within the transmission range of each client. In both cases, the clients were considered in a random order, and a client was not scheduled if its resource requirements exceeded the remaining processing and communication capacities. For each simulation result presented below, the performance was averaged over a set of ten random layouts of clients.

As an example application in our scenario, we chose real-time video compression for streaming, where video data from networked cameras are compressed in real time at the access points and streamed to the clients over a 802.11b Wireless LAN. The computation and communication requirements for the application were estimated from measurements made on MPEG-4 video encoding in *ffmpeg*. These measurements, made at different frame rates on standard video sequences in QCIF-format are summarized in Table 2.1 below. Each client was randomly assigned one of the application configurations from the table.

For the communication links, we used a 802.11b MAC layer which has three non-overlapping channels. The values used for the link capacities l_{ij} were the maximum data rates achievable for each client-AP pair in absence of interference, as listed

Table 2.1: Video clips used and their resource requirements

Video Sequence	Frame Rate (Frames/sec)	Processing Req. (% Full capacity)	Bandwidth Req. (kbps)
AKIYO	15	2.76	125
AKIYO	20	2.86	180
AKIYO	25	4.14	183
FOREMAN	15	4.32	224
FOREMAN	20	5.40	232
FOREMAN	25	6.25	234
NEWS	15	3.25	234
NEWS	20	3.89	257
NEWS	25	5.17	259

Table 2.2: Typical Achievable bandwidth with 802.11b

Range (m)	Basic Data Rate (Mbps)	Baseline throughput (Mbps)
≤ 125	11	5.1
125-162	5.5	3.3
162-262	2	1.52
262-331	1	0.79

in Table 2.2. The numbers in the table are based on throughput vs. range characteristics (See Fig. 2.5), as measured on CalRadio, a 802.11-based experimental platform with a software-programmable MAC which was developed in our group [JSP07]. The interference model that the scheduler operates on is $I = 2R$ (I=interference range, R=transmission range), as described in Section 2.3. This model is used to identify potentially interfering sources which the scheduler avoids scheduling together. It is noted that this is different from the RTS/CTS approach used in 802.11. The effect of this difference will be discussed in Section 2.7.3.

The processing capacities of the enhanced access points were based on general purpose computers with 2.6GHz Pentium 4 processors. To take into account the processing overheads of running the scheduler on the access points, we first measured the run-times of the scheduler for the specific loads, and then subtracted the equiva-

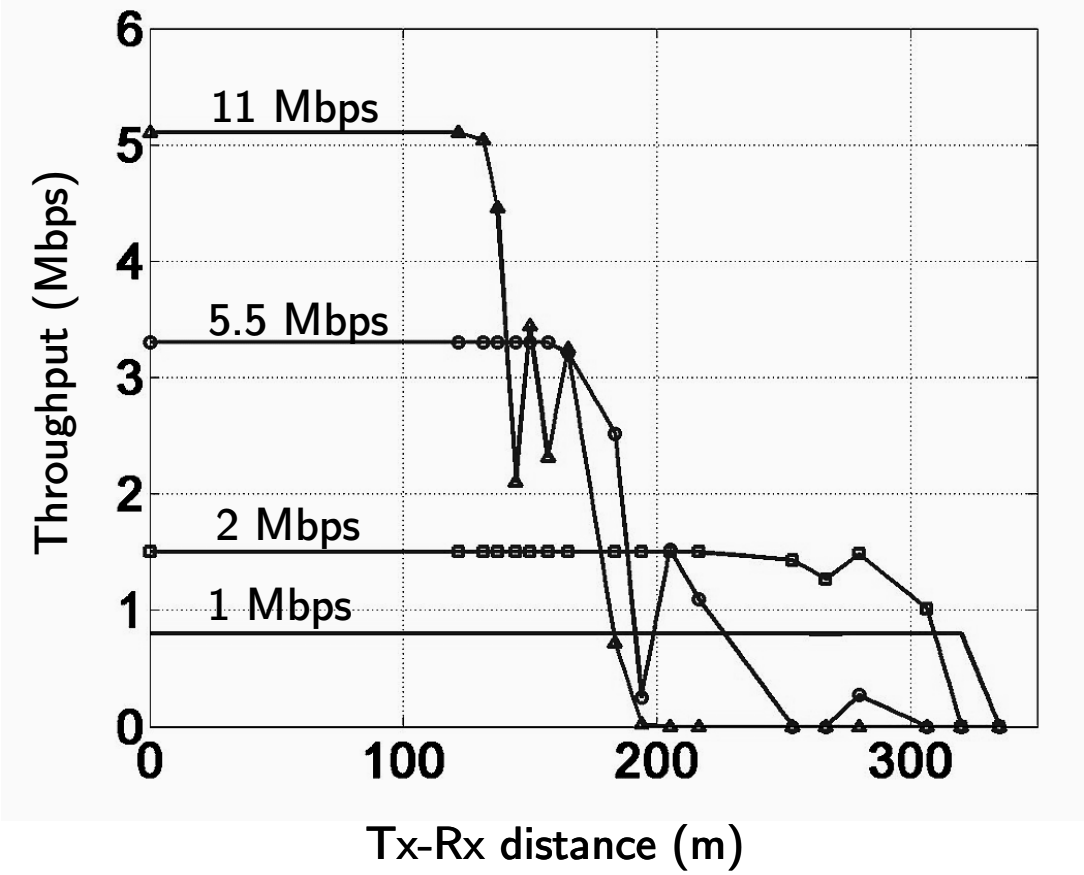


Figure 2.5: Measured Throughput vs. Range of 802.11b for outdoor transmission between a single pair of nodes.

lent processing times from the processing capacities in subsequent runs. The reported numbers correspond to these overhead-adjusted simulations.

Our simulated network consisted of APs placed in a regular triangular grid and clients randomly placed around them. Each AP was assigned one out of the three non-overlapping channels. Fig. 2.6 shows the pattern of channel allocation in an example grid, where the vertices represent the AP positions and the circled numbers on each vertex represents the channel used by that AP. This allocation maximizes the separation between interfering access points. The size of the grid is varied to generate scenarios with different levels of overlap between the transmission ranges of interfering APs. The client positions are distributed evenly among the coverage areas of all the APs. This means that while the total number of clients per AP can differ due to overlaps, there are still a minimum number of clients within the transmission range of each AP. We also conducted the experiments with the clients uniformly distributed over the whole region, but it was found to be less effective in simulating smaller networks due to edge effects that often leave one or more APs unloaded. We only present the results for the balanced client loads here, since the results in the other case were very similar after accounting for the edge effects.

2.7.2 Simulation Results

We show the performance of our algorithm in terms of the average number of clients scheduled. First, we present the simulation results for the simplified version of the system model, comparing the performance of FJRS with the LP-heuristic and optimal solution. Later, we present the results for the complete version of the problem, accounting for the effect of interference.

Simplified Problem without Interference

In the following graphs, we compare the performance of the schedulers in terms of number of clients scheduled, which are plotted against the total number of clients presented to the scheduler. Figure 2.7 shows the results from a simple case with a limited number of clients, where all the clients have identical resource requirements

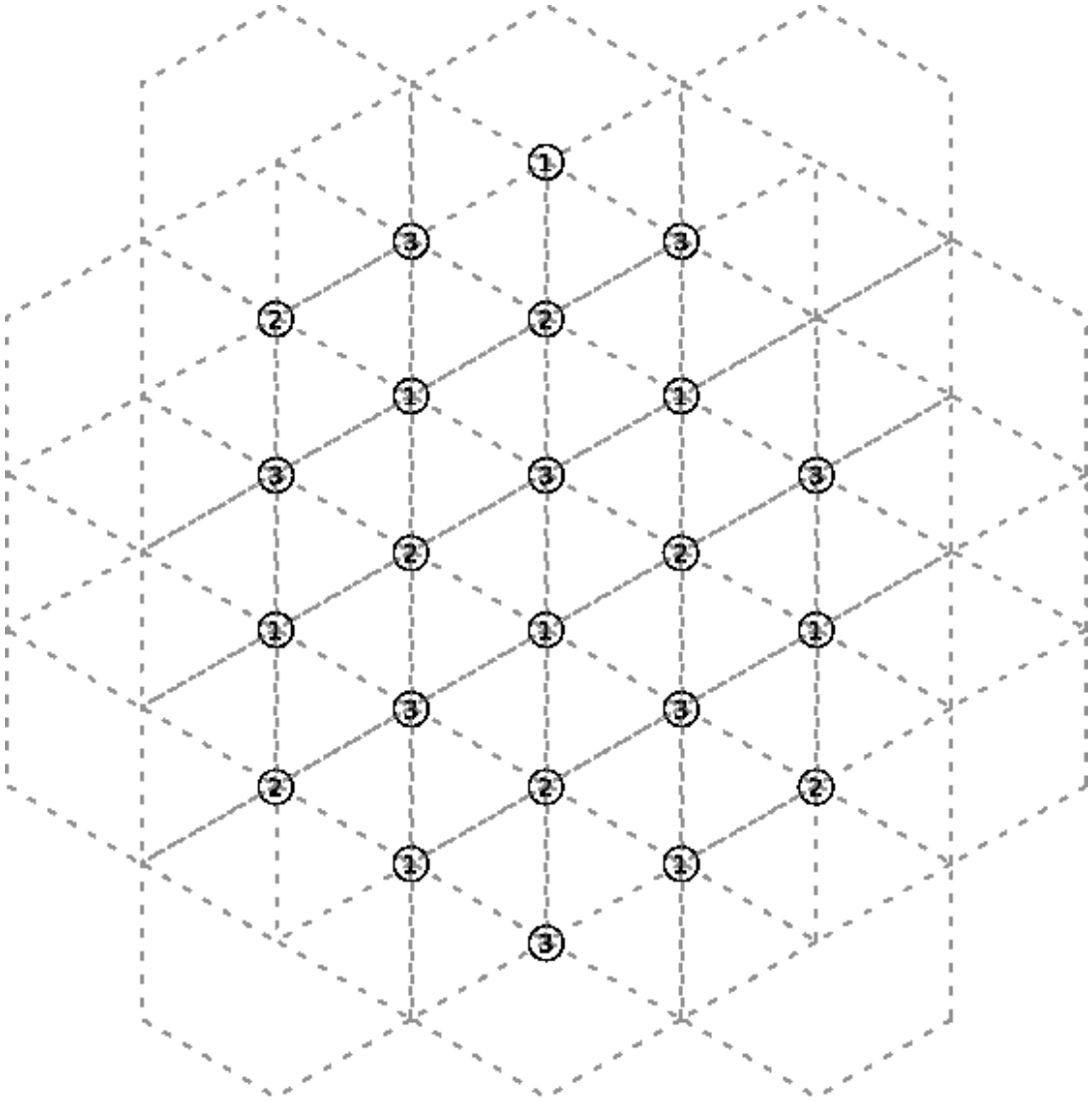


Figure 2.6: Allocation of channels to access points

(corresponding to the $FOREMAN-25$ clip). Figure 2.8 shows larger systems where clients have different resource requirements, based on randomly selected rows from Table 2.1. For each method, there are different lines representing plots for different numbers of access points(M).

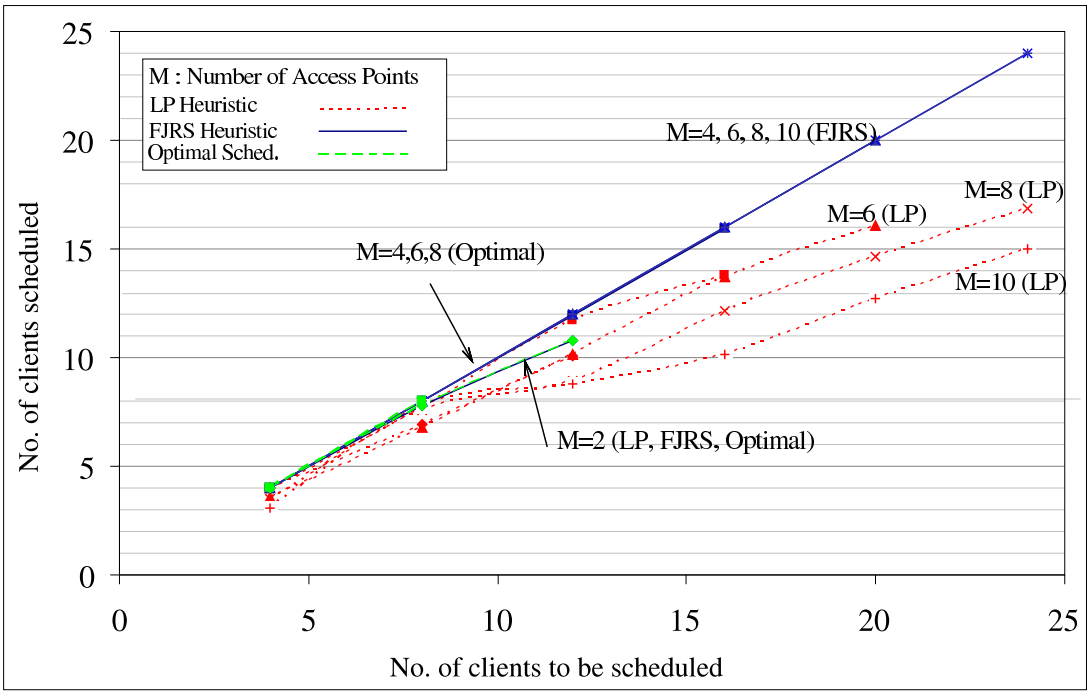


Figure 2.7: Scheduling performance for clients with identical resource demands

In Figure 2.7, we have a comparison of the LP and FJRS heuristics with the optimal scheduler. It can be observed that in most cases the FJRS scheduler achieves the maximum performance, *i.e.* scheduling all clients. In cases where it does not (*e.g.* for M=2), its performance matches that of the optimum scheduler. The LP performs well for smaller systems, but its performance degrades with increasing number of nodes. For example, with 8 APs (M=8) and 24 clients, LP-based method schedules 17 clients on the average, while FJRS can schedules 24 clients. From Figure 2.8, we observe that the performance of the FJRS scheduler flattens out as the number of clients increase, which is how the optimum performance would be expected to behave. We also see that the knee of the curve rises as more APs are added to the system. The LP method works well for small number of nodes, but as the size of the system and the number of possible solutions increase, its performance degrades rapidly. This is

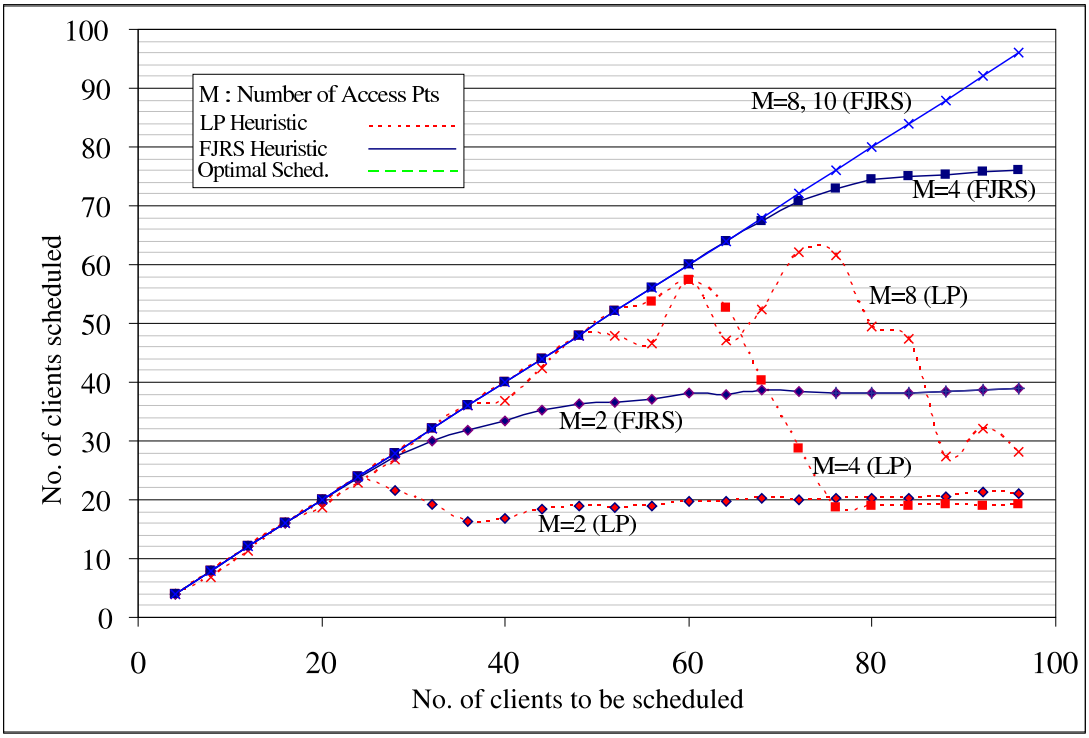


Figure 2.8: Scheduling performance for clients with varying resource demands

probably because the limited number of steps makes the quality of the solution very dependent on the initial point. However, FJRS is found to be a robust method as it performs well for larger systems and with a variety of client requirements as well.

Complete Problem with Interference

For the complete problem, we present two sets of results with three plots in each, corresponding to our FJRS scheduler, a random allocation scheme, and a nearest- access-point scheduler respectively. All three plots in each group are drawn to a common scale to allow direct comparison. The lines in all the plots represent the scheduling performance averaged over ten random distributions of client positions. For each line, error bars showing the standard deviations provide an indication of the statistical variations and a confidence measure of the results.

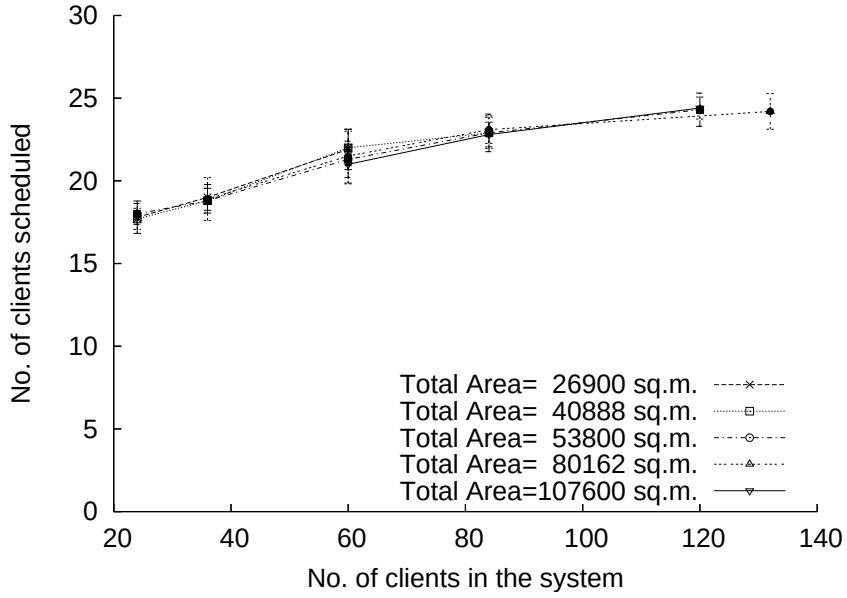
As mentioned in Section 2.7.1, the grid of access points shown in Fig. 2.6 is scaled to different sizes. In Fig. 2.9, each line shows the performance for a different

size of the terrain. So, for each line, the distance between the APs remains constant, while more clients are placed within the same area. The plots demonstrate how the scheduling performance changes with increasing client load on a given set of fixed APs. In Fig. 2.10, the terrain size (and therefore the distance between access points) is varied along each line, while the client density is held constant. Changing the distance between neighboring access points changes the degree to which interference affects the channel capacity for each AP.

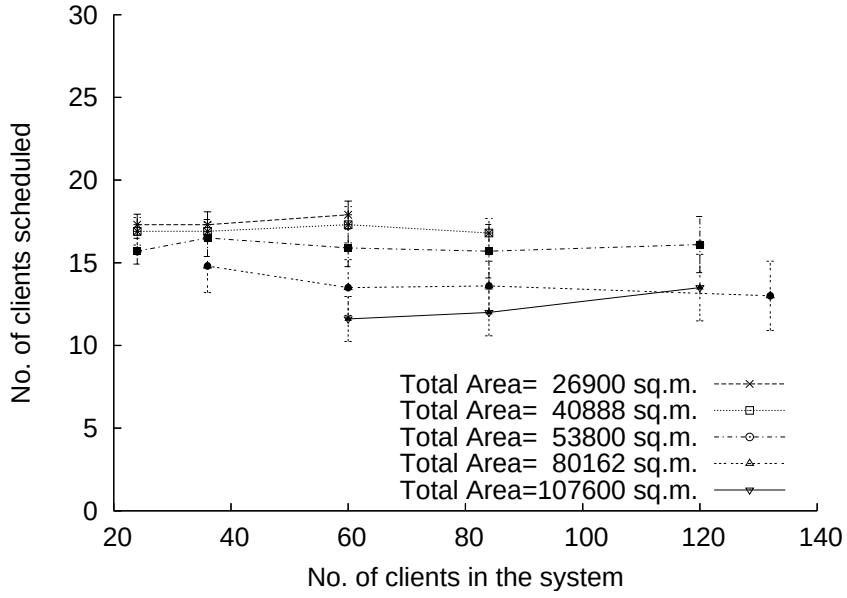
It can be observed that our approach (FJRS) results in a significantly larger number of clients being scheduled than the two other methods. Moreover, its performance scales smoothly with increasing client density. The performance of the random scheduler remains almost constant over an increase in client density, but shows a decrease in performance as the area is increased. This is expected, since it does not favor the nearer access points. So the bandwidth penalty for randomly choosing the access points far away increases when they are spread further apart. The third method, choosing the nearest access point, shows a gradual increase in performance with increases in density or terrain size.

2.7.3 Verification in NS

For the second part of our evaluation, we used packet-level simulation to verify whether a set of client assignments produced by the scheduler can be sustained in a real network scenario. As mentioned in Section 2.7.1, the scheduler makes its decisions based on an abstract model of the system. In this part, we evaluate how the scheduler's output is affected by limitations of this model. For example, the scheduler estimates interference based on a constant interference range, and uses a worst case estimate due to unknown client positions. In contrast, the simulator computes interference using a $SINR$ model which provides a more realistic estimate. Moreover, the MAC layer does not avoid interference completely, but uses a RTS/CTS mechanism that results in some wasted throughput due to collisions. Similarly, back offs during transmission also reduces throughput. Through these simulations, we observe the effect of these losses on the throughput of individual clients. We used the well-

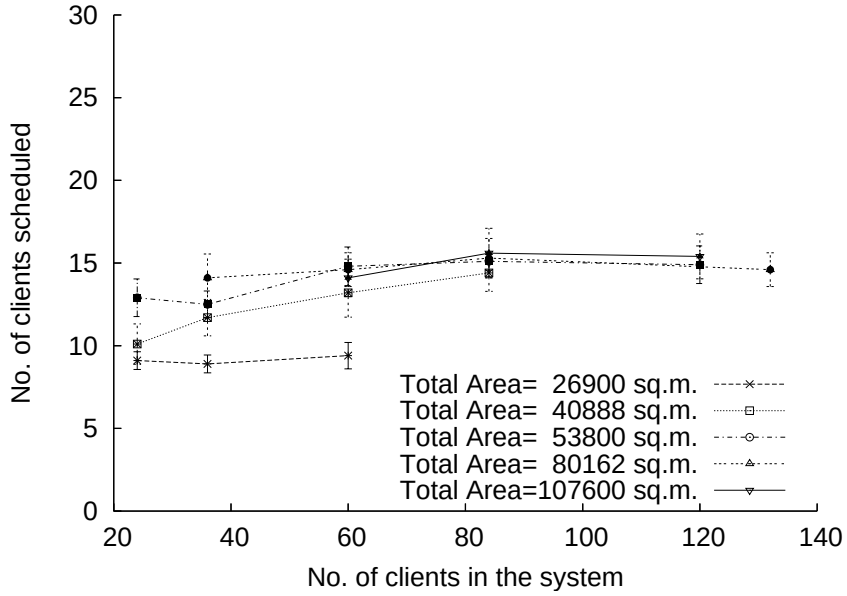


(a) FJRS



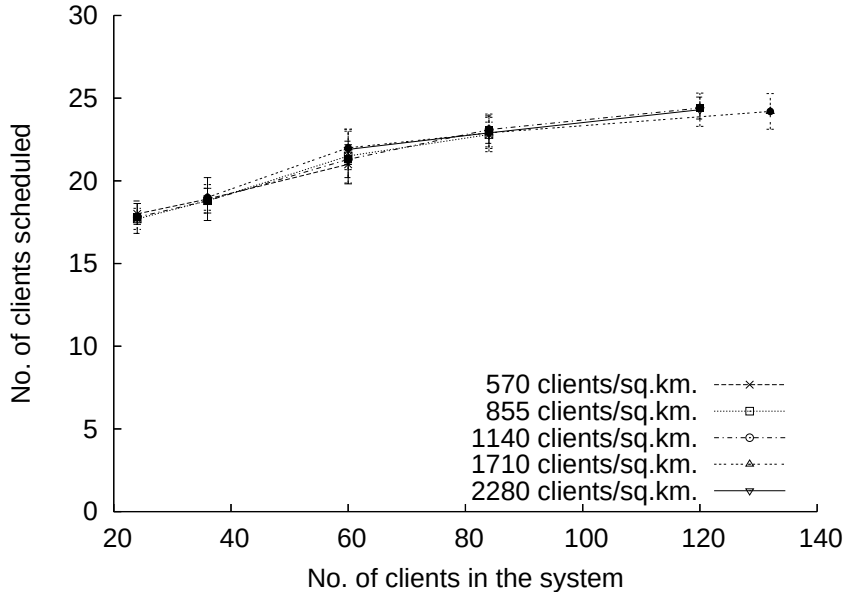
(b) Random AP

Figure 2.9: Scheduling performance for clients with varying resource demands for three scheduling approaches. 12 Access points, total area scaled with constant client density. Equal number of clients within range of each AP.

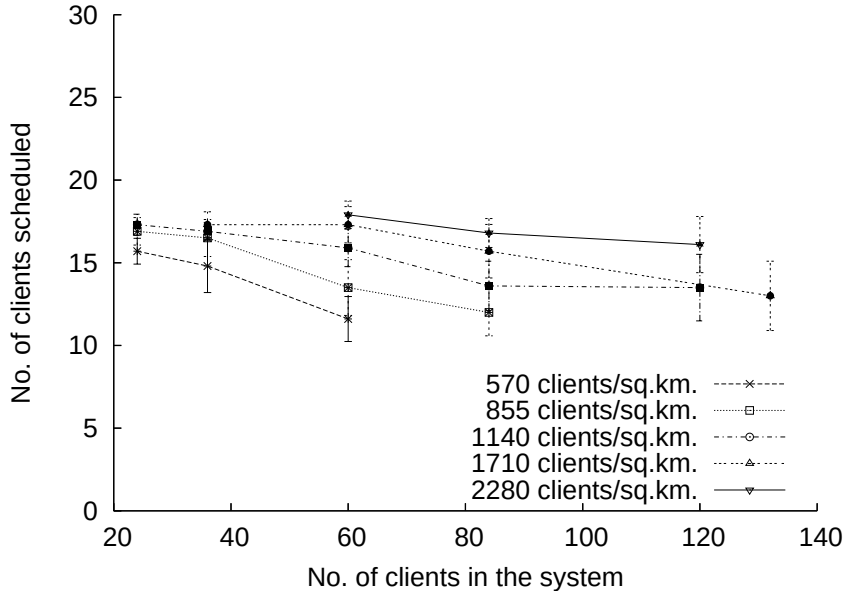


(c) Nearest AP

Figure 2.9: Scheduling performance for clients with varying resource demands for three scheduling approaches. 12 Access points, total area scaled with constant client density. Equal number of clients within range of each AP (Figure continued).

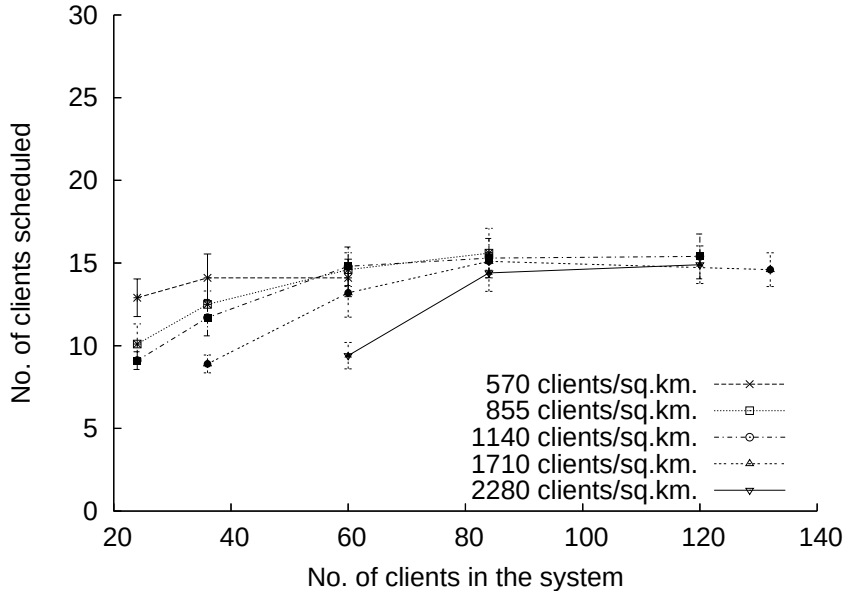


(a) FJRS



(b) Random AP

Figure 2.10: Scheduling performance for clients with varying resource demands for three scheduling approaches. 12 Access points, client density varied with constant total area. Equal number of clients within range of each AP.



(c) Nearest AP

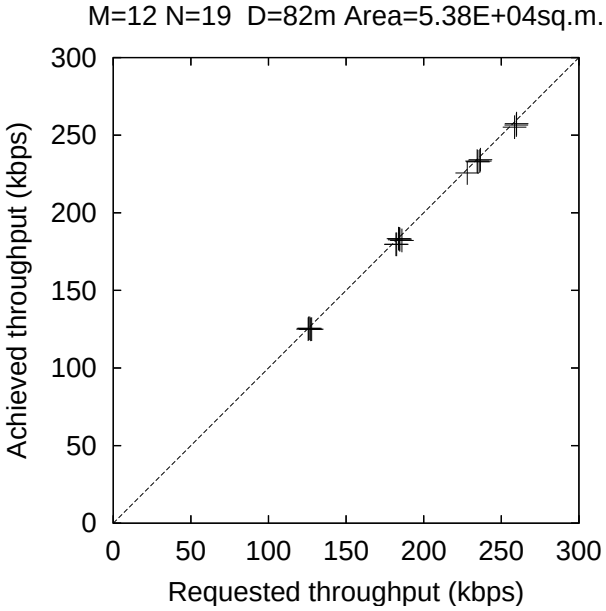
Figure 2.10: Scheduling performance for clients with varying resource demands for three scheduling approaches. 12 Access points, client density varied with constant total area. Equal number of clients within range of each AP (Figure continued).

known ns2 [NS] simulator for the packet-level simulations. We adapted the simulator to override all automated routing and AP selections, and enforce the access point assignments produced by our scheduler.

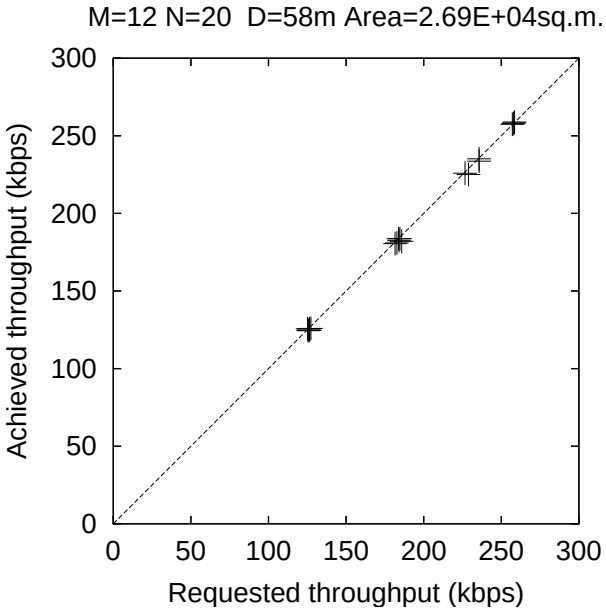
Fig. 2.11 shows the performance of our scheduler under three different sets of simulation parameters. The dotted line is included as a reference, showing the case when the achieved throughput equals the requested. Thus the distance of each dot from the reference line represents the difference between the requested and achieved throughput for that client. As mentioned earlier, any difference would be the effect of protocol overheads and interference effects that could not be accounted for by the scheduler.

In Fig. 2.11(b), the area is scaled down from 53800 sq.m. to 26900 sq.m., while doubling the client density (from 570 to 1140 clients per sq.km.) to maintain the same number of clients. The performances are similar, suggesting that the increase in interference due to the APs being closer is accounted for effectively by the scheduler. Fig. 2.12(c) shows the effect of increasing the density of clients by the same amount as before, without scaling the size of the area from Fig. 2.11(a). It can be observed that the scheduler selects more clients but with lower throughput requirements. It is expected that with an increased client density, there are many more clients with lower throughput requirements that are likely to be close to one or more APs. Since these clients have the least effect on the remaining communication capacity of the system, the scheduler selects them first.

We next demonstrate that our scheduler does not underutilize the network resources, *i.e.*, it schedules the maximum number of clients possible for a given system. One way to do so could have been a comparison with the output of the optimal scheduler. However, the high computational complexity and running times of the optimal scheduler makes it impractical to compute its output for every system configuration. Instead, we use the ns2 simulations of the scheduler output to indirectly test how closely the load generated by our scheduler matches the optimal allocation levels. To do this, we inject approximately 10% extra clients into a system that is already determined to be saturated, *i.e.*, where the FJRS scheduler was able to schedule only a subset of the given load. It is expected that after adding extra clients that attempt to

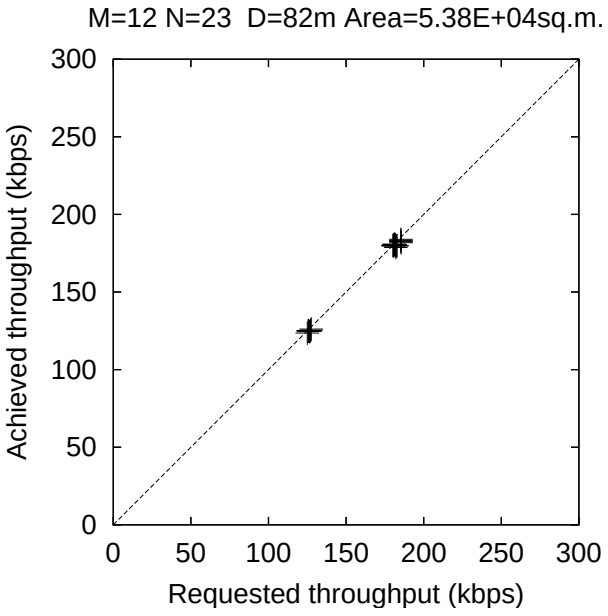


(a) N=19, Area=5.38E04sq.m.



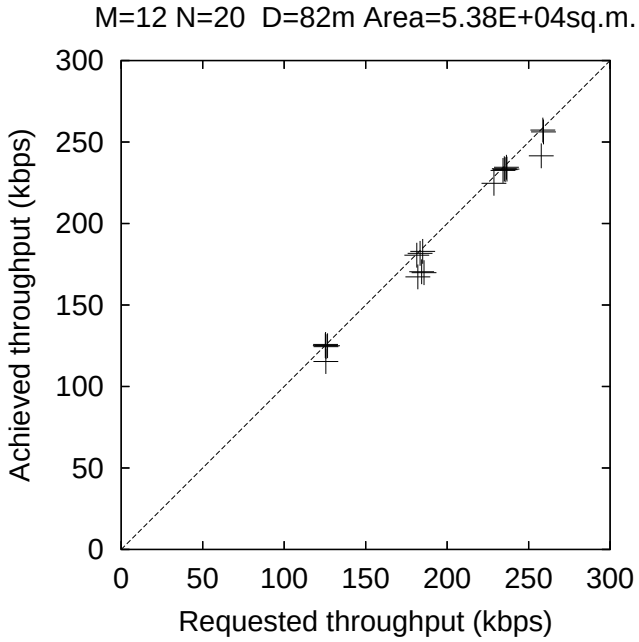
(b) N=20, Area=2.69E04sq.m.

Figure 2.11: Throughput for CBR traffic using NS2. (b) shows the effect of increasing client density and reducing the area to keep the number of clients in the system same as in(a). In (c), the client density is doubled while keep the AP density same as in (a).

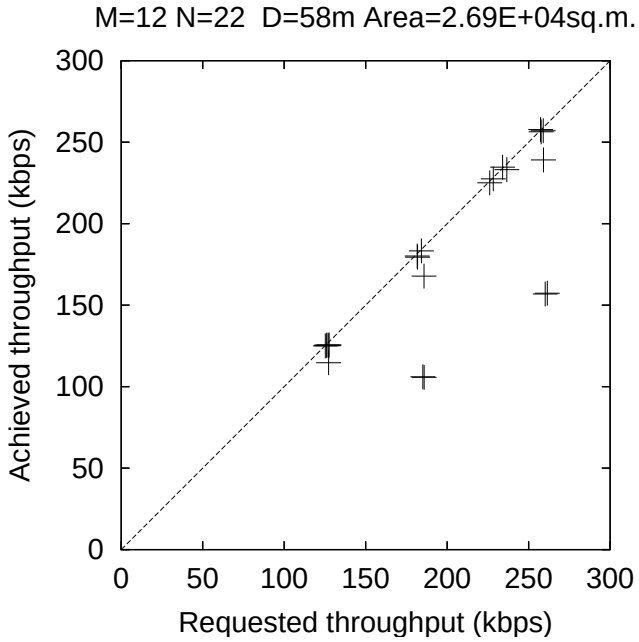


(c) N=23, Area=5.38E04sq.m.

Figure 2.11: Throughput for CBR traffic using NS2. (b) shows the effect of increasing client density and reducing the area to keep the number of clients in the system same as in(a). In (c), the client density is doubled while keep the AP density same as in (a) (Figure continued).

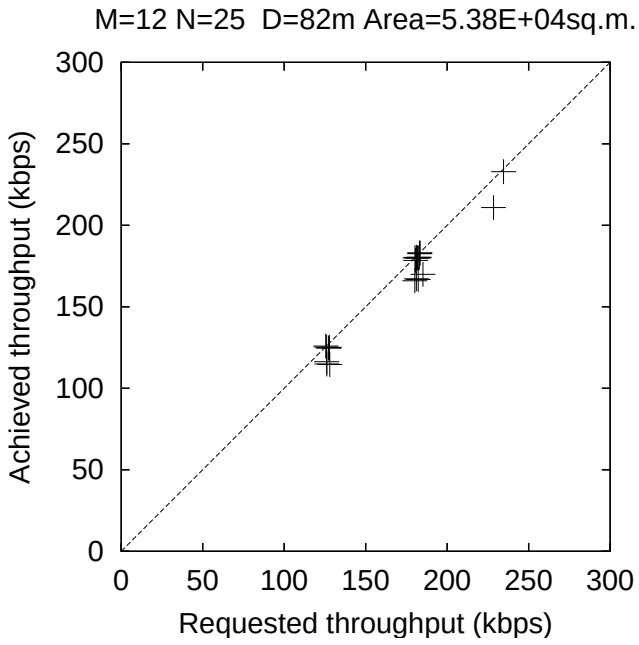


(a) N=20, Area=5.38E04sq.m.



(b) N=22, Area=2.69E04sq.m.

Figure 2.12: Throughput for CBR traffic after adding 10% extra clients to the each of the loads in Fig. 2.11.



(c) N=25, Area=5.38E04sq.m.

Figure 2.12: Throughput for CBR traffic after adding 10% extra clients to the each of the loads in Fig. 2.11 (Figure continued).

use more resources than available, the resulting system will not be schedulable anymore and will show a degradation in overall performance. The results are presented in Fig. 2.12, for the three network configurations used earlier in Fig. 2.11. The number of added clients are 1, 2 and 2 respectively. The reduction in achieved throughput across the set of clients is very apparent, as the points fall away from the reference line. This suggests that the load allocated by the scheduler was close to saturating the network resources.

2.7.4 Running Time

Apart from the the number of clients scheduled, another critical property of the scheduler that affects its overall performance is the running time. The running time limits how frequently a new load can be scheduled in the system. Moreover, since the processors in the APs are responsible for running the scheduler, the running time indicates how much of the system resources is consumed by the scheduler itself.

In Fig. 2.13, we show how the running times for our algorithm scales with increasing loads. When compared with Fig. 2.2 (Page 18), it can be observed that unlike the optimal approach, FJRS can feasibly handle large sets of clients. For example, even with loads of over 100 clients, the running time for the scheduler remains within the range of a few seconds. In contrast, an optimal approach would require hours for loads as small as 10-12 clients. Though the plot uses logarithmic scale to accommodate a large range of running times, when plotted on a linear scale (not included) it can be observed that the running time of FJRS roughly increases proportionally with the number of clients. This behavior agrees with the expected time complexity calculated in Section 2.4, and can be useful in estimating the scheduling overhead during system design.

2.8 Conclusions

In this paper, we presented a study of the joint computation and communication scheduling problem in a proposed network of enhanced access points designed to

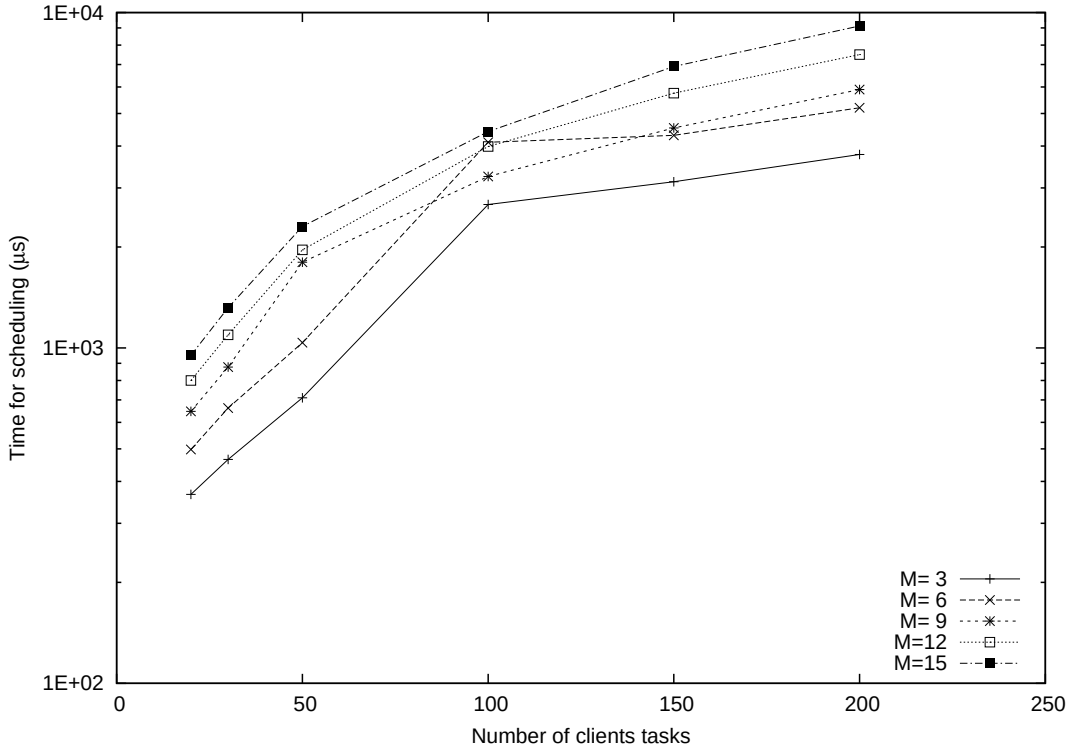


Figure 2.13: Scaling of running time for FJRS with large numbers of clients.

support rich mobile applications on thin clients. We have developed a set of heuristic approaches to solve the scheduling problem when the running time is a critical constraint. Through a set of simulation studies we demonstrated how our approach can achieve an excellent scheduling performance, while performing under tight time constraints.

The text of this chapter, in part, is based on material that has been published in the IEEE Global Communications Conference(Globecom) (S. Mukhopadhyay, C. Schurgers, S. Dey, "Joint Computation and Communication Scheduling to Enable Rich Mobile Applications", *Multimedia Communications, Software and Services Symposium at the IEEE Global Communications Conference*, Washington D.C., November 2007) and material submitted to the ACM Mobile Computing and Communications Review (S. Mukhopadhyay, C. Schurgers, S. Dey, "Enabling Rich Mobile Applications: Joint Computation and Communication Scheduling", *ACM Mobile*

Computing and Communications Review). The dissertation author was the primary researcher and author in the publications, and the coauthors listed supervised the research that forms the basis of this chapter.

Chapter 3

Model Based Techniques for Data Reliability in Wireless Sensor Networks

In the previous chapter, we discussed how the architectural principle presented in Section 1.3 can be applied to mobile wireless networks to increase the application functionality without expanding the footprint of the end nodes. In this chapter, we present how the same principle can be applied to wireless sensor networks to reduce the footprint of sensor nodes without affecting the functionality of the application. We demonstrate an approach for effectively performing the primary function of reliable data collection with these low-footprint sensor nodes.

3.1 Introduction

One of the results of the convergence of computing and wireless communication applications described in Chapter 1.1 has been the emergence of wireless sensor networks in the last decade. These are networks of computing devices with wireless communication capabilities that can interface with sensors to enable autonomous, fine-grained monitoring of the physical world. Numerous applications have emerged for Wireless Sensor Networks in the recent years, for example, structural monitor-

ing of building and bridges, monitoring of environment in farmlands and fine-grained climate control in office buildings, industrial process control, etc. Moreover, a large-scale, rapid growth in the deployment of such systems is predicted in the near future [HC05, ONW05].

The main driving force behind the growth of wireless sensor networks has been the phenomenal developments in semiconductor design, which have enabled the creation of the sensor network nodes: small devices capable of computing, wireless communication as well as sensing. The advancements made in VLSI circuits and microprocessor design technologies, like down-scaling of feature sizes and lowering of operating voltages, have in turn allowed the sensor nodes to become smaller and more power efficient [HSW+00]. In the future, sensor nodes can be expected to continue to become smaller, more energy efficient and cheap, qualities that would enable effective and pervasive deployment for them. This would, in turn, benefit many new applications that can benefit from large numbers of such low-footprint sensor nodes [F+06, MI06].

The primary functionality of a sensor networking application is the reliable collection of sensor data. As the sensor nodes become smaller and cheaper, errors in various stages of the data collection process become more and more prevalent, and ensuring the reliability of sensor data becomes progressively harder. On the other hand, traditional approaches for reliability require various overheads in hardware, processing and communication bandwidth. These overheads make it hard to achieve the required reliability concurrently with the footprint reduction of sensor nodes.

We believe that to perform reliable data collection under these conditions, fundamentally new approaches will be necessary. Here we present an example, based on the architectural principle introduced in Chapter 1 (Fig. 1.3 on Page 8). The principle of splitting the end-node is applied here by dividing the functionalities of sensing, processing and reporting data among two types of network nodes. The sensing is performed by dedicated, very low-footprint wireless sensor nodes, which represent the wireless nodes in Fig. 1.3(b). A second type of nodes, called aggregator nodes representing the shared fixed node in the same figure. These are equipped with substantial data processing capabilities and can connect to one or more sensor nodes. The

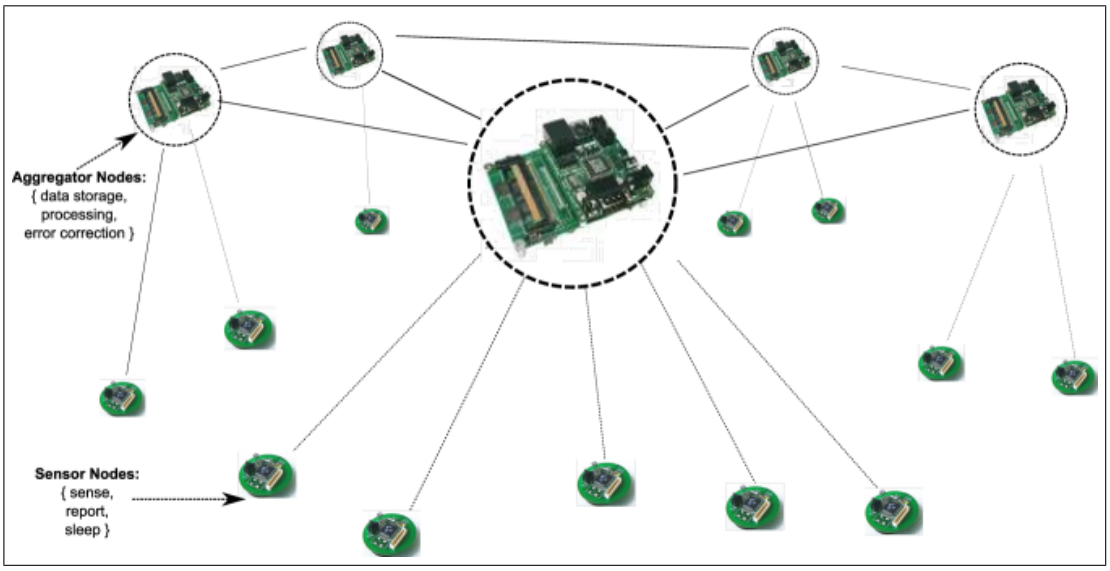


Figure 3.1: Network of sensor nodes and clusterheads

resulting architecture is shown in Fig. 3.1, with two types of customized sensor nodes available today representing the sensor and aggregator nodes respectively. In practice, the aggregator nodes can be made with generic off-the-shelf processors, while the sensor nodes are expected to be even smaller and simpler.

In this chapter, we specifically look at the problem of making the sensor data collection reliable in such an architecture, where the sensor node is not capable of incurring the overheads of traditional reliability approaches. We propose a solution that allows reliable data collection with no overhead at the sensor nodes, concentrating all of the necessary processing at the aggregator nodes. Below, we first present the problem of data reliability in more detail and explain the intuition behind our approach.

Data Reliability in Wireless Sensor Networks

The data reliability problem in wireless sensor networks arises due to errors introduced at various stages in the data collection. First, the hardware becomes less robust to many types of errors due to the effects of aggressive technology scaling. Another source of unreliability are the errors in the wireless communication channels, as limitations on transmission power due to tight energy constraints makes them more susceptible to noise and interference. The problem is further aggravated by exposure

to harsh physical environments, which is common for many typical sensing applications. Subsequently, ensuring reliability of the data in a sensor network is a growing problem, and will be a challenging part of designing sensor networks. Our proposed approach utilizes properties specific to sensor networks and has two main advantages: first, it handles multiple sources of errors together, and second, it imposes no overhead at the sensor nodes.

To introduce our approach, let us consider a sensor network that monitors the temperature distribution in an office environment by making new readings every few minutes. Now, since the 'meaning' of the data carried over this network is known beforehand, during regular operation certain properties of the data can be validated within the network and samples deduced to be impossible can be marked or ruled out. For example, if the temperature is measured every minute, and is always observed to report a value within two degrees of 25°C, an few isolated readings at 100°C or -10°C are likely to be due to errors, and should be verified against other sources before triggering any response. Similarly, if a light sensor that is used to automatically control lighting reports a sudden change in the light level but falls back to the prior level in the next second, it is probably not a good design to change the lighting level immediately. Both these cases illustrate how the meaning of the data, known to the application designer, can be used to test the validity of the samples within the network as they are recorded. Our proposed approach for reliability is a generalization of this idea, where knowledge of any properties about the data source can be systematically used to perform error correction. Such knowledge could be available during design, passed on during deployment, or inferred from historical data.

The different sources of errors affecting wireless sensor networks have been studied extensively, and many methods to handle them individually have been developed in the literature. However, the traditional approaches to provide reliability against each kind of error are based on adding redundancy, and lead to resource overheads in terms of communication bandwidth or hardware complexity, as discussed in Section 3.2 of this Chapter. On the other hand, our approach uses the properties of the data to distinguish them from errors introduced at any stage. Moreover, it introduces no hardware, processing or energy overheads at the simplified sensor nodes,

and allows simplification of the sensor node design and further miniaturization without impacting the robustness. This makes it a perfect fit for our proposed architecture, which is discussed in more detail in Section 3.3.2.

We demonstrate our approach through a methodology that uses temporal correlation in sensor data sets to perform error detection and correction at receiver nodes. Our method, described in Section 3.4, consists of generating predictions for future sensor data at run-time by using the knowledge about correlation, and comparing sequences of these predictions with the observed data within a decision tree. The correlation properties are embodied in data models, which generate predictions based on recent history of observations. While we limit our implementation only to autoregressive(AR) models, the methodology can be extended to work with other types of models that are suitable for specific properties of data, *e.g.*, periodicity. We also illustrate how our method can be used to complement traditional techniques like CRC-based error detection, *e.g.*, to adjust the level of protection depending on the relative levels of correlation in the data and the levels of errors. We present the problem of data modeling in Section 3.5, and discuss the criteria that makes a model suitable for use in our method. We also describe how the model can be adapted to changes in data properties at run-time.

3.2 Errors in Wireless Sensor Networks

In this section, we present a background on the types of errors in wireless sensor network, and motivate the need for new approaches of reliable data collection. We first describe the different types of errors that can affect the sensor data, and then look at some of the traditional methods for handling such problems. We also discuss why these techniques will not be able to meet the requirements of many sensor networking applications, and why a different approach for reliability is necessary.

3.2.1 Sources of Errors in Sensor Networks

In a wireless sensor network, sensor data samples are exposed to various sources of errors during the course of sensing, processing and communication. First,

the sensors can report faulty readings because of changing operating conditions (*e.g.*, temperature, humidity, etc.) or bias and calibration drifts caused by aging [EN03, BMEP03]. After sensing and quantization, the data can still be affected within the sensor nodes by various hardware errors such as crosstalk and radiation effects [KH04]. Afterward, the data samples are again exposed to channel errors in the wireless communication channel. These errors can differ widely in terms of severity, frequency of occurrence and statistical properties. We classify the sources of error into two types, transient and permanent, based on the effect they have on the sensor data. The method of error correction that we present here is designed to address the transient errors that affect the sensor data after quantization, and is effective against errors from multiple sources.

The main driving force behind the feasibility of ultra-small, cheap, low-power sensor nodes has been the aggressive technology scaling in VLSI circuits, which has enabled the small form factors and high battery efficiency of sensor nodes. However, as a side-effect of this scaling, transient errors in hardware are becoming a prominent problem, as the shrinking of feature sizes to nanometer scales and the lowering of supply voltages to sub-volt ranges are making them vulnerable to various noise and interference effects [S$^+$02, Sch06, Bau03]. These effects include various temporary environmental conditions such as power supply and interconnect noise, electromagnetic interference, electrostatic discharges and also neutron and alpha particle strikes [Bau05, H$^+$03, ZBD05]. Short term disturbances caused by any of these effects can lead to transient errors in logic or memory circuits that would affect any data being processed or stored.

The other source of transient errors in sensor data is the communication channel. Effects like noise, interference and fading are already substantial problems, and since they are fundamental properties of the physical medium, are going to be around for future generations of sensor networks too. Moreover, the need for longer battery life and low-power operation are going to limit the transmission power and number of retransmissions that could be used to compensate for such channel impairments at the lower layers.

For the current prototypes of sensor nodes, the dominant source of errors are

Table 3.1: Different reliability methods and their resource overheads.

Reliability Method	Sources of Error Addressed		Resource Overheads
	Sensor Node Hard-ware	Comm. Chan-nel	
TMR (hardware)	✓		Hardware area
ECC (memory)	✓		Hardware area
Channel coding		✓	Transmitted bits
Channel coding + Source coding		✓	Hardware complexity, Extra com-putation at sensor node
Model-based error correction	✓	✓	None at sensor node, Processing at clusterhead

the sensing errors and the communication channel errors. But the trends shown by semiconductor technology development indicate that as the nodes get smaller, the effect of soft errors in the hardware are going to become more dominant. Additionally, outdoor deployment in harsh physical conditions also aggravates the vulnerability of these nodes. So, for future generations of sensor networks that are envisioned (*e.g.*, Smartdust [MI06, WLLP01]), errors in both the hardware and communication channels are going to be significant problems. We now look at how these problems may be handled by traditional approaches to error correction.

3.2.2 Traditional Methods for Error-correction

The different sources of errors have been studied extensively in literature and various methods have been developed to handle them individually. Some of these traditional methods for error correction are listed in the first four rows of Table 3.1, with the check marks indicating which source of errors each method is able to address.

However, each of these approaches also adds some overhead into the system, which can be prohibitive for many applications of sensor networks. As also listed in the table, these overheads can be in terms of transmitted data, costs or hardware complexity. For example, the traditional methods to handle soft errors in circuits, such as Triple Modular Redundancy (TMR) or Error Correction Codes (ECC), result in adding extra hardware and complexity to the designs [ZD03, IHK06, MER05]. With the projected increase in the levels of soft errors, the cost and complexity overheads will continue to increase with successive generations of technology. Similarly, communication errors are traditionally handled by introducing redundancy, either through channel coding or Automatic Retransmission Requests (ARQ) coupled with error detection. Channel coding works using Forward Error Correction (FEC) codes, which add extra bits to transmitted packets that allow correct decoding when some of the bits are corrupted. One common example of FEC is Reed-Solomon coding [Wic95], which is widely used in telecommunications, broadcasting and data storage. It works by processing data in fixed-size blocks, adding a fixed number of overhead bits during encoding. The decoder can recover a block of data when the number of errors for the block does not exceed half the number of overhead bits. In Section 3.6.4, we compare our approach against Reed-Solomon coding, and demonstrate the substantial energy savings that can be achieved by avoiding the overheads in packet size. The overheads of channel coding can be offset by data compression through source coding, but this will increase the complexity and cost requirements on the sensor nodes. ARQ is simple to implement and has little overhead under good channel conditions, but in presence of errors it can have high energy and bandwidth overheads due to retransmissions.

These observations show that the traditional techniques for error handling have significant resource overheads, which will make them prohibitively difficult to apply in many sensor networking applications. The effect of these overheads will increase further in future generations of sensor networks, with sensor nodes becoming smaller and more resource constrained. As a result, newer techniques for reliability need to be developed to fit the needs of highly resource-constrained sensor networks, particularly for applications with large numbers of sensor nodes operating in presence of multiple sources of errors. We propose an approach that uses the redundancy within

sensor data sets to address transient errors in both computation and communication, thus enhancing reliability without any complexity overhead in the sensor nodes. We also demonstrate how this approach can be combined with traditional techniques depending on the requirements of the application and the level of redundancy within the data.

3.3 Overview of Model-based Error Correction

In this section, we present an overview of our approach and methods for model-based error correction. We first discuss our overall approach and its implications on the network architectures for sensor networks. Later, we discuss a proposed methodology for error correction based on this approach.

3.3.1 Using Sensor Data for Error Correction

The central idea of our proposed approach consists of using the properties of sensor data sources to detect and correct errors in the data. Sensor networks are designed to observe physical processes, and typically designed to oversample the sensors, which results in significant spatial and temporal correlations [VAA04]. The correlation provides built-in redundancy within the data which can be handled in different ways as part of system design. One option could be to eliminate this redundancy close to the source by compression techniques, as is done in traditional communication networks. However, while this improves utilization of the communication channel, it also reduces the robustness of the data against errors and necessitates strong error coding methods that can add complexity to the end-points. Instead, we propose to utilize this redundancy to correct errors from various sources. The approach is based on the difference between the correlation properties of the data and the sources of errors. For example, the soft errors are completely uncoordinated [KH04], while the sensor data can have different types of correlation depending upon the application, type of sensor, location of deployment, etc. Since the correlation properties of the sensor data can vary depending on many factors, the success of this approach depends on effectively identifying these properties. It also depends on how well the properties of the errors

distinguish them from the data. In the later part of this section, we describe a method that tries to achieve both these goals, and in the next section, we will discuss which properties of errors allow them to be distinguishable. When the level of redundancy in the data is extreme (too low or too high), it is possible to use our techniques in conjunction with traditional approaches like channel coding and source coding to tune the level of operation to current requirements. We illustrate this in Section 3.4.5 through a hybrid approach that incorporates the outcome of CRC checksum computation in our method.

There are two important benefits of being able to use the data properties for error correction. First, it makes the network design simpler and more efficient by addressing multiple types of errors together and removing the overheads of removing or adding any redundancy through compression or error coding. Secondly, it allows all the processing required for error correction to be moved out of the sensor node. This makes this approach fundamentally suitable for the network architecture principle outlined in Chapter 1, since it lets the sensor nodes to become simpler, and use specialized nodes to perform error correction, possibly for multiple sensor nodes simultaneously. The impacts of using this network architecture are described below.

3.3.2 Hierarchical Network Architecture

Hierarchical topologies are becoming increasingly popular for wireless sensor network design, *e.g.*, in IEEE 802.15.4 and other research networks [SKM04, BC03, SH05]. In a hierarchical topology, some of the nodes (clusterheads) are dedicated to work as gateways for group of other nodes (called leaf nodes), so that the latter can only connect to a node outside their cluster through the clusterheads. Typically, the nodes in a hierarchical sensor network architecture are heterogeneous, since the clusterheads carry more traffic while the leaf nodes are dedicated to sensing.

As noted earlier, our approach for reliability allows the splitting up the operations in data collection between sensing and error correction, which can be placed on different nodes, following the application design shown in Fig. 1.3. This, coupled with the resulting asymmetric resource requirements makes our approach a natural fit for hierarchical network architectures. Fig. 3.1 shows such an architecture, and

illustrates how the tasks of our correction approach can be mapped on it. The sensing is performed by clusters of dedicated sensor nodes that report the sensor data to more complex clusterhead nodes. Each cluster is organized in a star topology, but the clusterheads can connect between themselves using any flat or clustered connection topology. The clusterheads are distinct from the resource-limited sensor nodes by design as well as functionality. Unlike the very resource-limited sensor nodes, they have more processing and energy resources, and are capable of multiple complex functions that involve data processing and storage. Each clusterhead node can configure the scheduling of sensing, data reporting and sleeping cycles of the sensor nodes around it.

Using this architecture and our proposed approach for reliability can make a sensor network more flexible and scalable, by enabling the handling of diverse sensing requirements without sacrificing the reliability of the overall system. Thus, the sensor nodes can be simple and very cheap, with only sense-and-report functionality. Such sensors can be deployed in large numbers and can be replaced or supplemented when requirements change or individual nodes fail. On the other hand, the clusterheads will be complex, but can have most of their functionality implemented in software, which will allow a cost-effective, generic design to be used across applications. Since our approach is based on using the correlation properties of sensor data, it can also benefit from the large number of sensor nodes and lot of measurement points that such an architecture will enable. In summary, the use of the proposed architectural principle for WSNs promises to enable new applications by drastically shrinking the footprints of the sensor nodes, without sacrificing the reliability of the sensor data.

3.3.3 Overview of our Implementation

We implemented the idea of using data properties for reliability in our proposed methodology for error correction, called *Model-based Error Correction*. Our method consists of analyzing the sensor data and capturing the relevant properties in a data model, which is used during system operation to perform error detection and correction on incoming data. The algorithms in our approach run on the clusterhead nodes, and can also be implemented in hardware to improve their efficiency or allow

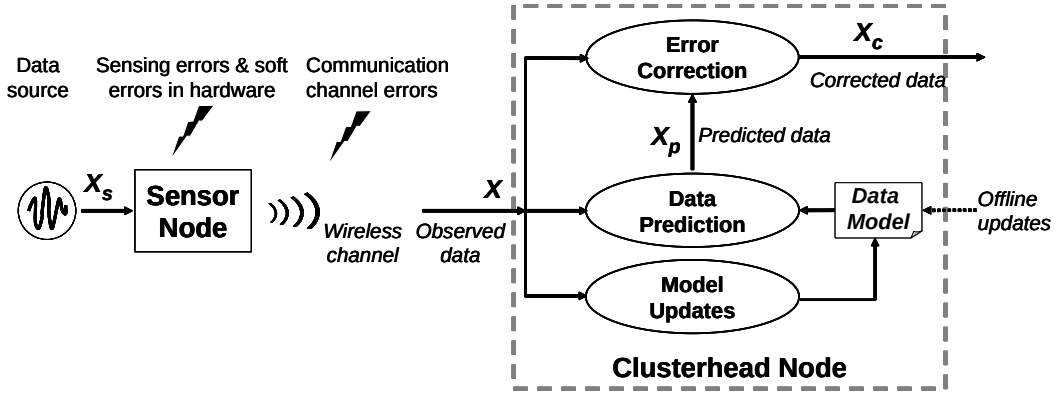


Figure 3.2: Overall scheme for model-based error correction.

parallel execution.

There are two main parts in this method: data modeling and correction. The first step is the construction of the data model by analysis of the properties of the source data. Certain statistical properties will be generally applicable to the type of application, and may be identified by offline analysis, *e.g.*, knowing that correlation time is on the order of hours rather than milliseconds. Other properties will need to be identified for each sensor and need runtime tuning of the data model. Once a model is identified, it is used during normal operation of the network to detect and correct errors, as illustrated in Fig. 3.2. Before each sample of sensor data is received, its predicted value (X_p) is computed using the data model. Upon receiving the observed sample (X), the error correction block uses X_p and the past observations to decide the likelihood of the observed data being erroneous. The algorithm chooses the corrected value (X_C) to report based on either the observed value or the predicted value, depending on the outcome of error detection. The proposed method also includes a facility for adapting the model online based on the collected data to improve the accuracy of prediction and correction. It should be noted that our approach is best suited for sensor networks designed to monitor slow variations over time in relation to the sampling rate. This excludes event-detection applications or those with low sampling rates, where the data is normally expected to contain sharp variations lasting for one or two samples, and the transient errors can be indistiguishable from application events.

3.4 Predictive Error Correction

In the previous section, we presented the idea of performing error correction using data models that can represent the correlation in the sensor data sources. Here, we first discuss how our approach for correction is shaped and supported by characteristics of errors occurring in wireless sensor networks. We then present the details of our method, and explain the correction algorithms and implementation framework. It is assumed in this section that a working data model is made available externally. The computation of the model is covered in the Section 3.5.

3.4.1 Error Models

The main idea behind our proposed approach for error correction is to use the correlation properties within the data. For this to be effective, it is necessary that the correlation characteristics of the data are different than that of the error. Since our implementation is based on temporal correlation across successive data samples, it is expected to be effective against errors that are uncorrelated in the same time scale. We observed that among the types of errors discussed in Section 3.2, many of the errors can be modeled as random bit-errors in the quantized samples, uniformly distributed across the bit-positions. Below, we explain how this model can be applicable for two different types of errors, and its importance in the context of our error correction methodology.

The first type consists of the transient errors that occur in hardware for the various reasons discussed in Section 3.2. These are well-known problems, and numerous fault models have been developed for different sources, *e.g.*, probabilistic models for process variations or random particle strikes, as well as deterministic models based on circuit layouts and signals for Crosstalk [ZS04, ZBD04]. However, these usually model the effect of the errors at the gate-level or register level [MKK05]. Since we are interested in the overall effect of errors on the data, we propose to model the various types of uncorrelated *single event upsets* as described in [ZBD05] as independent bit errors that are uniformly distributed across the data. This model is particularly appropriate for transient errors in memory chips, where the uniform structures for the

logic and layout of memory cells make the cells equally susceptible to radiation and interference effects. However, although it does not include design and input specific parameters, our model is also useful as a first order approximation for logic circuits.

The other source of errors that we consider is the wireless communication channel. There are various well-known models to represent communication channel errors as well. The choice can depend on the source of the errors, which can include noise, interference and fading due to mobility, as well as the specifics of the modulation scheme, channel conditions, transmission rates, etc. For channels dominated by fading or interference errors, the channel conditions are usually measured and used to estimate an error model, which is applied for a sequence of packets together. However, such models are most effective where data is transmitted in continuous streams, so that the channel properties across adjacent packets are correlated. On the other hand, in a typical sensor network scenario, the sensor data may be sampled and transmitted at larger intervals, probably on the order of seconds [S$^+$04]. This leads to the packet interval being larger than typical channel coherence times by orders of magnitude, so that estimating a channel model for shaping the transmissions of a sequence of packets will be very inefficient. In the absence of any estimates, we propose to use a model of uncoordinated random bit errors that are independent across packets.

If both the hardware and communication errors can be represented with the uniform random model, it allows our correction approach to be effective in distinguishing them from the data by relying on the data correlation. Moreover, this also allows both types of errors to be addressed together, which is one of the main benefits of our method. For our simulations in Section 3.6, we generate the errors using a Bernoulli process with an uniform error probability for all bits in the data. It should be noted that although we have used the uniform BER model for its simplicity, it is not a prerequisite for our method, which can be effective under bursty error conditions as well. The main requirement for our approach is that errors are uncorrelated across data samples. So as long as the packet transmission intervals outstrip the channel coherence time or burst lengths, the correlation in the data can still be effective in identifying errors. For example, if the sampling of the sensors happens every few seconds, 50-200 ms error bursts [WM06] are not going to affect multiple data samples,

so the errors in successive samples will be uncorrelated. Moreover, if the high BER during an error burst affects multiple bits within each packet, it helps our approach by making such a sample easier to detect. We also confirm this in Section 3.6, using traces of real errors with bursty, non-idealized characteristics.

3.4.2 Correction Methodology

Our error-correction method consists of computing a predicted value for each sample, and using the prediction error, *i.e.* the difference between the predicted and the observed value, to detect whether the observation is correct or erroneous. In general, an abruptly large prediction error implies that the observation has been corrupted by an error. However, since the data source is a random process, there will always be a certain level of prediction error for even the best model. Moreover, as described in Section 3.5, limitations like complexity or size of history can also introduce inadequacies in the models, which lead to an increase in the level of prediction errors. Therefore, the main challenge in our method lies in detecting the cause of a prediction error after it is observed, *i.e.*, whether it is due to the randomness of the data or from an error introduced into the data after sensing.

In our method, we compare the prediction error for each sample with that of its neighboring samples, and delay the reporting of the final value by a few sample periods to allow comparison with future samples. Whenever there is a prediction error, our method decides whether to report the observed or the predicted value by looking at how the choice affects the prediction errors for the subsequent samples. Since the correlation model makes the prediction based on recent history, choosing an erroneous observation affects the prediction of the future samples and results in a progressive degradation of prediction accuracy. On the other hand, if a prediction inaccuracy is due to randomness or modeling error, choosing the observed value is unlikely to impact the future levels of prediction errors. The details of the decision-making algorithms are discussed in Section 3.4.4. Since we assume that errors in adjacent samples are uncorrelated (see Section 3.2), whenever the decision algorithm detects an error in an observed data sample the observation is marked and treated as an erasure. The sample is also excluded from use in computing subsequent predictions

in order to limit propagation of errors.

The main steps in our implementation of error detection are outlined in Fig. 3.3. There are two blocks, *prediction* and *decision*, which implement the main control functions. There are also two storage blocks, *observation history* and *prediction history*, which maintain the state information required for modeling and prediction respectively. The prediction block implements the data prediction model obtained from an external model generator, which produces a predicted value for incoming data samples based on recent history of observations. The output from the prediction block is stored in the prediction history, and is used along with the history of observations as input by the prediction block to predict future samples. The decision block determines whether an error occurred in a sample by analyzing the effect of this decision on the predictions of the future samples. To enable doing this efficiently, the history of the predictions since the last corrected sample is stored in a tree structure, called *Prediction History Tree (PHT)*, which is processed by the decision algorithm. Below, we first explain the structure of the PHT, followed by a detailed description of the decision algorithms that can operate on the PHT structure. We also discuss a hybrid approach that can be used when the sensor node radio has some out-of-band error detection mechanism, such as a CRC checksum function, built into the hardware. In that case, we discuss how the result of the checksum calculation can be used to complement the model-based error detection to improve the correction performance.

3.4.3 Data Structure for Prediction History : PHT

The prediction history tree holds the few most recently observed sensor data samples at any time. It also stores the all the possible sequences of predicted values for these samples, along with the corresponding prediction errors. It is a complete binary tree, where all nodes at a given level, *i.e.* at the same distance away from the root, contain the different possible values for the same sample. The root node contains the last corrected data sample, and its two children hold the observed and predicted values of the next sample currently under consideration. The leaf nodes of the PHT hold the predicted values for the currently observed sample, and each path from the root to a leaf node holds a possible sequence of observed or predicted values for the samples

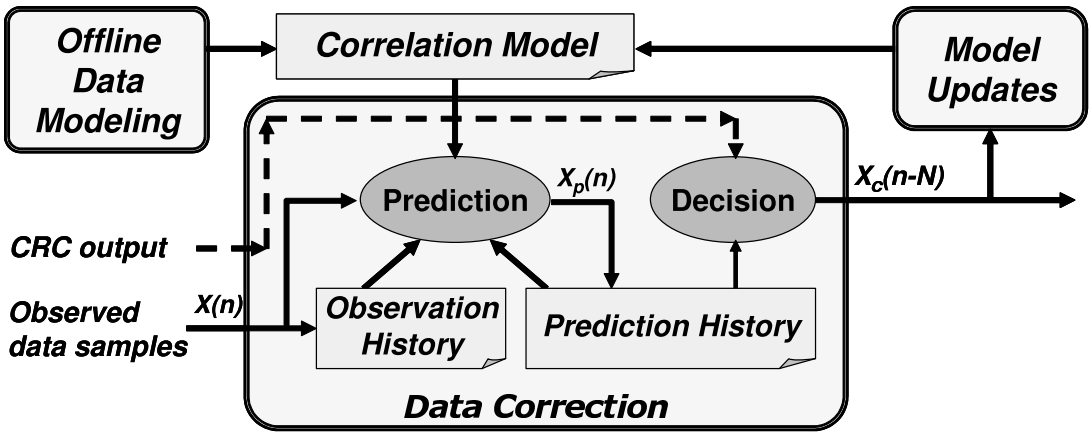


Figure 3.3: Functional diagram of predictive error correction block. The dotted line shows the hybrid approach which incorporates the output from CRC detection when available.

from the one last corrected till the one most recently observed. The depth of the PHT, N, defined as the number of samples held in it after the first level, is a key parameter in the decision algorithm. It is called *decision delay* and its choice is bounded by the delay tolerance of the application and the available processing resources in the node.

The structure of the PHT and the method of updating it are illustrated with an example in Fig. 3.4, where $N = 2$, so that the tree has 4 ($N + 2$) levels and 15 ($2^{N+2} - 1$) nodes. Each level $l \in 0, 3$ corresponds to a sample value at time $n - 3 + l$, and each node in this level holds a pair of values; the first represents the observed or a predicted value for the data sample and the second holds the prediction error for the nodes with a predicted sample. Outgoing links marked 0 and 1 connect each node to two child nodes, containing the observed and predicted data values for the next sample respectively. The nodes are sequentially numbered starting with 0 for the root, such that for any node i there are two child nodes $2i + 1$ and $2i + 2$ that hold the observed and predicted values respectively. The root contains the last corrected value $X_c[n - 3] = 100$ ($X_c[n - N - 1]$), and the even numbered leaf nodes contain the different predicted values of $X_p[n]$ that would be computed for different choices of previous values.

Once the new sample $X[n]$ is observed, the prediction errors for all the values of $X_p[n]$ are computed. The decision algorithm is run to choose the most likely value

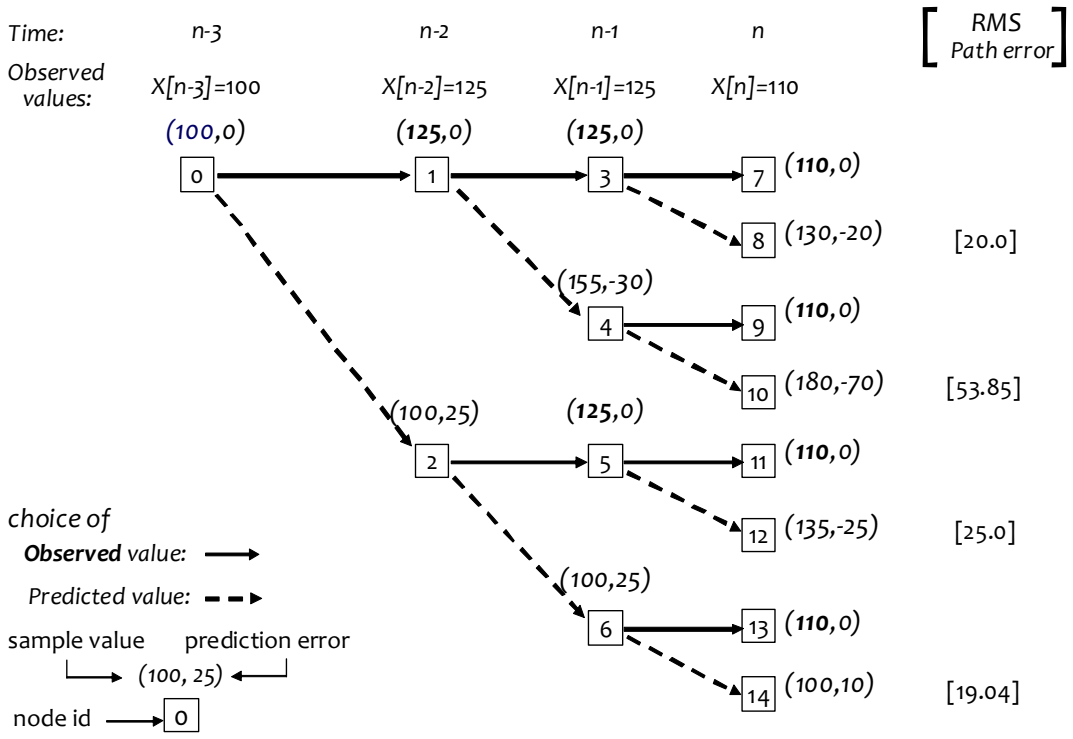


Figure 3.4: Prediction History Tree with example data ($N = 2$).

for $X_c[n-2]$ out of the values in nodes 1 and 2. The contending paths in the example are the ones ending in any of the even numbered nodes in the last level, *i.e.* nodes 8, 10, 12 or 14. The prediction errors for the nodes on each of the paths are used as a basis of comparison by the algorithm, and finally the value of either the observation 125 (node 1) or the prediction 100 (node 2) is chosen for $X_c[n-2]$.

After making the decision, the algorithm updates the PHT for the next sample. We call the two subtrees rooted in node 1 and 2 as the *observation subtree* and *prediction subtree* respectively. One of these nodes is chosen to become the new root node, and all the nodes in its subtree move up by one level. The values in the other subtree are discarded. A new level of leaf nodes is then added, with the even numbered nodes containing the next set of predicted values $X_p[n+1]$. The next observed value $X[n]$ is inserted in the all odd numbered nodes in level N, and the decision and update process is repeated for the next sample.

3.4.4 Decision algorithm

Given the recent history stored in the PHT, the task of the decision algorithm is to determine at time n whether $X[n-N]$, the observation N samples back, was erroneous. In other words, $X[n-N]$ has to be assigned either the observed or the predicted value, stored in nodes 1 and 2 of the PHT respectively (Fig. 3.4). The decision is based on how the choice affects the prediction accuracy of the next N samples (till $X[n]$). Each root-to-leaf path in the PHT contains a possible sequence of values for these samples, based on different choices of observed or predicted value for each. The algorithms compare the behavior of prediction errors along these paths to reach the decision.

One possible approach for the decision algorithm is to select the subtree of PHT that contains the root-leaf path with the minimum average correction error. However, since the samples in the PHT are yet to be corrected, the correction errors for them are still unknown. Our first algorithm, *MinErr*, follows this approach, using the prediction errors as estimates of the correction errors. The average prediction error (RMS) is computed over all the predicted samples in each path, and the path with the minimum RMS error is selected (Algorithm 3). This path will contain either node 1

Algorithm 3 $MinErr$ **Algorithm for Error Detection and Correction.**

Input: PHT, N `//N:depth of PHT`
Output: Choice of $Observed$ or $Predicted$ value

procedure `PathError(`$PHT, leaf$`)`
$patherror \leftarrow 0$
$num\_pred \leftarrow 0$
for all node $i \in path(0 : leaf)$ such that i is even **do**
`//Compute RMS prediction error for the predicted values`
$patherror \leftarrow patherror + PredErr[i]^2$
$++ num\_pred$
end for
return $(patherror/num\_pred)$

procedure `UpdatePHT(`i, X_{new}`)`
if path i contains node 1 **then** `//Select correct subtree`
$PHT \leftarrow$ Subtree rooted at node 1
else
$PHT \leftarrow$ Subtree rooted at node 2
end if
for all leaf node j of new PHT **do** `//Update last level of new PHT`
`//Add Observed value to 1st child`
$DataValue[2j+1] \leftarrow X_{new}$
$PredErr[2j+1] \leftarrow 0$
`//Add Predicted value from Data Model to 2nd child`
$X_p \leftarrow$ `PredictionModel(`PHT, j, X_c`)`
$Err \leftarrow X_{new} - X_p$ `//`Compute prediction errors
$DataValue[2j+2] \leftarrow X_p$
$PredErr[2j+1] \leftarrow Err$
end for

main
`//Find path with minimum average error`
$i_{min} \leftarrow \min_{i \in Leafnodes}$ `PathError(`PHT, i`)`
where $Leafnodes = \{2^{N+1}, 2^{N+1}+2, \ldots, (2^{N+2}-2)\}$
$X_{new} \leftarrow X[n+1]$ `//get next sample value at time` $n+1$
`UpdatePHT(`i_{min}, X_{new}`)`
if $i_{min} \leq 2^{N+2} + 2^{N+1}$ **then**
return 0 `//Use observed value`
else
return 1 `//Use predicted value`
end if

or 2, which is returned as the corrected value of $X_c[n-N]$. For example, in Fig. 3.4, among the four paths ending in nodes 8, 10, 12 and 14 the path with minimum error is 0:14. Since it contains node 2, the predicted value is chosen for $X[n-2]$. The PHT is

then updated as discussed earlier, and the steps repeated for the next sample [MPD04]. This algorithm is simple to implement, and consists of a small and fixed number of computations, but it suffers from a high sensitivity to the modeling performance. As mentioned in Section 3.4.2, prediction errors can occur even for correctly recorded samples due to modeling errors. While the modeling error reflects the quality of the model when averaged over a number of samples, the errors in individual samples can be unpredictably high or low. In the PHT, the effect of individual modeling errors is amplified for the paths that have a small number of predicted samples, which can allow a single sample to determine the overall decision. For example, for path $0{:}8$ in Fig. 3.4, the path error is the same as the error in node 8. Now, if the observed value of $X[n]$ were 111 instead of 110, the errors for paths 0:8 and 0:14 would have been 19 and 21 respectively, and $MinErr$ would have chosen the observed value for $X[n]$.

This problem is avoided in the *MinMax* algorithm (Algorithm 4), where the subtrees of nodes 1 and 2 are considered separately and the path with the maximum average error in each subtree is found. The subtree with the smaller maximum error is selected for the decision. This approach favors the solution that is expected to perform better in the worst case, *e.g.*, the predicted value is chosen in the example because it performs better over multiple paths. This makes $MinMax$ more resilient to modeling errors, since a path with a spuriously low average error will not affect the solution if the other paths in the subtree have higher average errors. For example, in Fig. 3.4, only paths 0:10 and 0:12 are directly compared. In this case, $X[n]$ would have to be as high as 145 (path errors = 15, 32, 19, 33) to affect the final decision.

Though more resilient than $MinErr$ to modeling errors, $MinMax$ does not take into account the specific properties of the model, which can cause spurious detections for certain types of models. Consider a case where the size of the history required to compute the prediction is smaller than the depth of the PHT. Now, for some paths, the value of $X[n - N]$ may not have any effect on the prediction for $X[n]$. For example, if the model used in Fig. 3.4 only uses the previous sample for prediction, then the choice of the predicted value in node 1 will have no effect on the predicted value in node 12. In some cases, the robustness of the decision can be further improved by excluding paths like these.

Algorithm 4 $MinMax$ **Algorithm for Error Detection and Correction.**

Input: PHT, N `//N:depth of PHT`
Output: Choice of $Observed$ or $Predicted$ value

```
obsErr ← 0
for k = 2^{N+1} to (2^{N+1} + 2^N − 2) do  //Find Max. Error in Observation Subtree
   err ← PathError(PHT, 1, k)  //RMS Prediction Error for path 1:k
   if err > obsErr then
      obsErr ← err
   end if
end for
predErr ← 0
for k = (2^{N+1} + 2^N) to (2^{N+2} − 2) do  //Find Max. Error in Prediction Subtree
   err ← PathError(PHT, 2, k)  //RMS Prediction Error for the path 2:k
   if err > predErr then
      predErr ← err
   end if
end for
if obsErr < predErr then
   return 0  //Use observed value
else
   return 1  //Use predicted value
end if
```

This is done in the *Peer* algorithm, shown in Algorithm 5. Here, individual pairs of nodes in each subtree are compared, instead of full paths. The algorithm compares nodes in parallel (peer) positions within the observation and prediction subtrees in terms of the absolute prediction errors. After all the comparisons, the subtree which has more samples with lower prediction errors than their peer nodes is chosen. The correction engine also uses available knowledge of the model to identify the predictions that are independent of the choice of $X[n-N]$, and excludes them from the decision making process. So, for the example PHT in Fig. 3.4, the prediction errors for nodes 4, 8 and 10 are compared against that of nodes 6, 12 and 14 respectively, and the result is to choose the predicted value, since its subtree has lower prediction errors in two comparisons vs. one higher. The property of the model is represented as the model order parameter M, which is the number of samples from history that are used by the model for prediction (Section 3.5.1). Before each comparison, it is ensured that either node 1 or 2, or a sample directly predicted from it is among the previous M samples of the ones under examination. As mentioned earlier, if $M = 1$, this

Algorithm 5 $Peer$ **Algorithm for Error Detection and Correction.**

Input: PHT, N, M, ETH
`//M:model order, ETH: error threshold, N:depth of PHT`
Output: Choice of $Observed$ or $Predicted$ value

```
for all n ∈ Prediction Subtree do
  s ← FindSibling(n) //Find sibling of n in Observation Subtree
  if s, n are even then //Only compare for predicted samples
     //Check if predicion of n depends on the root node
    comp ← CanCompare(PHT, n, M)
    if comp = true then
      if abs(PHT[n].err) > abs(PHT[s].err) + ETH then
        count ← count − 1 //Prefer observed value
      else if abs(PHT[n].err) < abs(PHT[s].err) − ETH then
        count ← count + 1 //Prefer predicted value
      end if
    end if
  end if
end for
if count > 0 then
  return 1 //Use predicted value
else
  return 0 //Use observed value
end if
```

step would exclude the comparison between nodes 8 and 12 in the example. Moreover, when the difference in prediction errors within a pair is much smaller than the average modeling errors, that pair is disregarded as well. The parameter ETH, representing this *Error threshold*, is used to implement this. The resulting algorithm offers the most stable results among the three approaches.

The three algorithms mentioned here have similar complexity of run-time computation and storage requirements, with the problem size defined to be the tree-depth N. Since the PHT has 2^{N+2} nodes, the space complexity is $\Theta(2^{N+2})$. The worst case time complexity has two components from the prediction [$\Theta(M.2^N)$] and the decision algorithm [$\Theta(2^{N+1})$] respectively, either of which can dominate depending on the relative sizes of the model order and the PHT. The correction accuracy is harder to analyze, because it depends on the statistics of the modeling errors as well as the actual errors introduced in the observation. We observed from our experimental studies (Section 3.6) that all the algorithms perform well when the models are accurate. However, as discussed before, they have different types of resilience towards

Algorithm 6 $PeerHybrid$ **Algorithm for Error Detection and Correction.**

Input: PHT, N, M, ETH
`//M:model order, ETH: error threshold, N:depth of PHT`
Output: Choice of $Observed$ or $Predicted$ value

```
if CRCError(1) then //Check CRC checksum for observed value in node 1
  return 1 //Use predicted value
else // Check rest of prediction history if CRC passed for node 1
  for all n ∈ Prediction Subtree do
    s ← FindSibling(n) //Find sibling of n in Observation Subtree
    if s, n are even then //Only compare for predicted samples
       //Check if predicion of n depends on the root node
      comp ← CanCompare(PHT, n, M)
      if comp = true then
        if abs(PHT[n].err) > abs(PHT[s].err) + ETH then
          count ← count − 1 //Prefer observed value
        else if abs(PHT[n].err) < abs(PHT[s].err) − ETH then
          count ← count + 1 //Prefer predicted value
        end if
      end if
    end if
  end for
  if count > 0 then
    return 1 //Use predicted value
  else
    return 0 //Use observed value
  end if
end if
```

modeling errors, with $Peer$ designed to perform best.

3.4.5 Hybrid correction

The method for model-based correction described above assumes that the error detection has to be done without any additional information apart from the observed sensor data. However, most radios used in currently available sensor nodes already have a checksum function built into the hardware [N+05], and it is possible to complement our approach using the CRC output.

When the checksum is available, we use it to complement the model-based error detection and increase the overall performance of correction. The pseudo-code for this hybrid method is shown in Algorithm 6. Here the result of the checksum computation is fed to the decision algorithm when it is available, as denoted in Fig. 3.3

with the dotted line. The normal model-based detection using the PHT is done on a data sample only when the checksum detects no errors. When an error is detected by the CRC, it is treated as a missing sample and the predicted value from the model is used. The modeling and update stages remain unchanged. This approach improves the correction performance by identifying communication errors which could have been misinterpreted by our approach as modeling errors. It does not capture errors originating before the communication link, *i.e.* sensing errors or hardware errors, which would still have to be detected by the model-based approach.

3.5 Data Modeling

In the previous sections we described how predictive data models can be used to perform error detection and correction on sensor data. So far, we have assumed that a data model, which can provide a prediction of the next data sample, is available to the error-correction framework. In this section, we discuss the problem of creating the data model, examine which characteristics make a model suitable, and describe the implementation of data modeling in our system.

The objective of the data model in our model-based correction system is to predict the next sample of the sensor data. Properties of the data source are identified and used to make the predictions. Previous research has shown that there are many possible types of model that can be used for this purpose [Aka69, Har89, KSW98]. However, there are a number of additional requirements characteristic to our error-correction framework that determine the suitability of a particular type of model. In this section, we first explore the most important of these requirements and show how our specific choice of model satisfies these requirements. In the second part, we discuss the implementation issues for data modeling.

3.5.1 Requirements of Data Model

The performance of the model-based error correction depends on the accuracy of the predictions, so maximizing the prediction accuracy is the primary goal of the

data model. However, for the model to be used effectively in our framework, the prediction also needs to be fast and have reasonably low computation and storage overheads. These requirements can place conflicting demands on the model, and it is necessary to a strike a balance among them when choosing the model.

Ideally, to maximize the prediction accuracy, the model should completely represent the correlation present in the data set. But when a data set shows strong long range correlation, a complete representation of the correlation can lead to prohibitive latency or resource overheads. It can be also caused by the choice of high sampling rates for slowly varying data sources. The resource limitation is most critical for the prediction step, since it is repeated for every path of the PHT, *i.e.* 2^N times, for each data sample during the data-correction process (Section 3.4.3). Moreover, with multiple sensors reporting to a clusterhead, the impact of any overhead is multiplied many times. For example, environmental sensor data like outdoor temperatures can exhibit correlation in the short term, on the scale of hours (*e.g.*, day vs. night), as well as in the long term, on a scale of years (*e.g.*, seasons) or multiple year cycles (*e.g.*, *El Niño*). Now, in order to capture the inherent correlation completely, the prediction model would have to refer to a very large data set spanning durations of years. For computing even a single prediction, the computations involved will lead to unacceptable latency. Similarly, the need to refer to large amounts of history would also incur prohibitive memory requirements. As a result, it may be necessary to choose a less complex model of limited dimensions to make its use on the clusterhead feasible.

Another problem affecting the accuracy of the predictions occurs when the data source is not strictly stationary, but has statistical properties that change over time. This will cause the the prediction accuracy to deteriorate progressively, so that a new model will need to be computed. Depending on the type of data source, it may be necessary to learn the model multiple times to reflect the current characteristics of the sensor data. However, while prediction accuracy is important, the best possible model may be too complex or too large to update efficiently.

Therefore, there are three main requirements that need to be considered when selecting a data model: it has to produce accurate predictions, it should be easy to use for prediction, and it needs to be easy to learn when necessary. There are many types

of models that can satisfy these requirements with different degrees of success. For our example implementation, we specifically chose linear autoregressive (AR) models, which capture the effect of recent history through an 'aging' process. Such models have been shown to be effective in capturing short-term prediction in time-series analysis and are widely used for forecasting [TJ99, Cha04]. Choosing AR models also satisfies the additional requirements very well, since they use linear prediction functions which can be computed very efficiently with minimal resource overhead.

The AR model captures the auto-correlation in a data set by expressing the prediction as a linear combination of previous samples, as shown in Equation (3.1):

$$X_p[n] = \sum_{i=1}^{M} a_i[n].X[n-i] \tag{3.1}$$

$$X[n] = X_p[n] + e[n] \tag{3.2}$$

where $X[n]$ and $X_p[n]$ represent the observed and predicted data values at time n respectively, and $e[n]$ is the prediction error. The model is characterized in terms of the model order M, which defines the *size* of the model, and the M coefficients a_i. This model has the advantage of low complexity of prediction, $\Theta(M)$, which reduces the impact on the resource cost at the clusterhead. Learning the model consists of estimating the coefficients a_is, for which we use recursive LS-estimation [Aka69, Hay96].

Some examples of the modeling performance are shown below using sensor data from four types of sources that are listed in Table 3.2 . The sources include an indoor light-level sensor from a testbed network we implemented [MPD04], publicly available environmental temperature and humidity sensor data from the California Data Exchange Center (CDEC) [CDE], and server rack temperature variations collected from a commercial data center [1]. Each of the data sets reports information about the sensed physical process as values quantized to 8 bit samples, *i.e.* the data values range from 0 to 255. Table 3.3 presents the modeling errors with estimated AR models offline.

[1] Unpublished data sets obtained from HP Data Centers through HP Lab, Palo Alto, CA

Table 3.2: List of data sets

Data set	Sensor type	Sampling period	Number of samples used in modeling
1	Data Center: Temperature	1 min	425
2	CDEC Alpine: Temperature	1 hr	398
3	CDEC Alpine: Humidity	1 hr	398
4	Testbed: Light	0.2 min	30000

Table 3.3: Modeling performance for data sets

Data Set	Model order	Modeling error (%)
1	4	0.021
2	22	0.430
3	4	1.150
4	2	0.007

3.5.2 Implementation of Modeling and Update

We designed the data modeling system in two parts: (i) off-line identification of the type of the model through the analysis of statistical properties, and (ii) run-time updates to the model (Fig. 3.5). As described earlier in this section, we selected the AR model as a balance between accuracy and ease of updates. In our implementation, the offline modeling consists of computing the order of the AR model, based on the correlation time and the sampling rate from training data. The process of run-time model estimation involves two operations, one to determine when an update to the model is needed by *tracking the prediction accuracy*, and a second block that performs the actual *model update*.

To support the run-time model updates, we introduce a special mode of operation, called *Estimation Mode*, in which the sensor nodes temporarily report the data with additional error protections. Operating in this mode makes more reliable data available for computing updated data models at the clusterheads. The Estimation

Mode can be implemented through a variety of temporary, software-based redundancy measures. For example, this may involve storing redundant copies to overcome hardware errors or multiple transmissions to protect against communication channel errors. During normal operation, the tracking process continuously monitors the trends in the prediction errors and triggers an model update request when necessary. The details of this monitoring process are discussed later. Upon receiving the update request, the error correction system pauses the regular data gathering process, instructs the sensor node to temporarily switch to the *Estimation Mode* and starts updating the model with the protected data. After the update is complete the sensor node goes back to *Correction Mode*, the normal mode of operation.

It should be noted that the sensor data collected in the Estimation Mode are still useful to the application using the data, and the switching between modes is transparent to the application. Moreover, the Estimation Mode can be implemented without any additional hardware overhead. However, the redundancy measures introduced in the sensor node increase the energy costs per bit when operating in this mode. Therefore, when triggering the updates, the benefits of updating the data model need to be compared to the overhead of switching to the Estimation Mode. In some cases, this overhead may make it preferable to continue to use the offline models instead of making the updates, especially if the properties of the data source do not change too much. For many applications, the model updates can be made more efficient by sharing them across multiple neighboring clusterheads, since a change in the observed physical phenomena that requires a model update is likely to affect many sensors deployed over a large area covering multiple clusters. In such a situation, a newly updated model from a neighboring clusterhead can serve as a good starting point for model estimation, or even be used directly. While this would require additional traffic and synchronization between clusterheads, it can lead to substantial savings in resources by reducing the time spent in Estimation Mode by each sensor node.

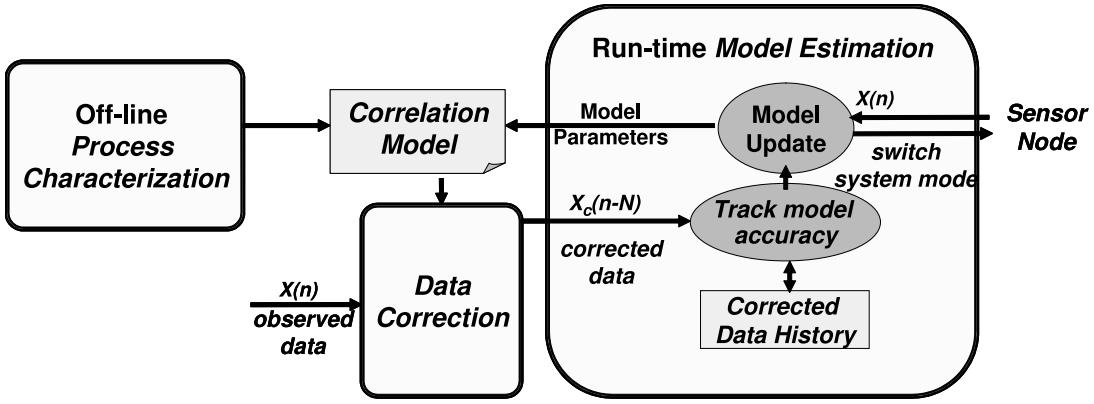


Figure 3.5: Detailed schematic of data modeling.

Model Tracking

We track the model accuracy using the prediction error, *i.e.* the difference between the observed and predicted values, for the recently obtained samples. When operating in Correction Mode, a running windowed average of the prediction error is maintained, and compared with a fixed threshold value to trigger a switch to Estimation Mode. Among the past samples, only the correctly received ones are used and the threshold is scaled for the number of correct samples in the averaging window. A comparison of the average estimation error with a fixed threshold gives a simple way to trigger model changes. The choice of the two parameters, threshold value and size of the averaging window, determine the frequency of updates. The optimal choice would depend on various characteristics of the data source and the system: the level of randomness within the data, the stationarity properties of the generation process, the accuracy of the data model, as well as the the cost overhead of operating in the Estimation Mode as compared to the Correction Mode. Thus the parameters provide a way to tune the overall system operation to achieve cost-performance trade-offs according to a global policy.

Model Updates

The run-time update stage operates in the resource-heavy Estimation Mode, so it needs to estimate the model with a minimal set of data points, unlike offline

modeling which can be done with unlimited data points. In order to reduce the overheads, we restrict the updates to only estimating the parameters of an existing model instead of recomputing the model. Many other optimizations are possible to make the online updates efficient as well. For example, if long term periodic variations (known as seasonality in Time-Series literature) can be identified in the data during offline modeling, it may be possible to characterize the data-generating process in terms of a set of states among which it operates. The states can be associated with pre-computed models, so that the run-time updates would only consist of matching the current conditions with the most likely state. For our specific case of AR models, we re-estimate the coefficients, but do not change the model's order at run-time.

The trade-off between the accuracy of the model and the resource costs of update is also important in determining when to stop the online estimation process. Increasing the number of samples used for estimation can increase the accuracy of the model. At the same time, sending more protected samples adds to the resource overhead at the sensor node and also reduces the number of data samples corrected by the clusterhead. In our implementation, we have used the RMS prediction error as a measure of the adequacy of the model. The system moves from Estimation Mode to the Correction Mode when the average prediction error over the current Estimation Mode window falls below a preset threshold.

Fig. 3.6 shows the effect of run-time updates on the modeling errors for the data set 4 from Table 3.2. The figure lists the coefficients of three data models computed using different subsets of the same data set as training data. Each subset is denoted in terms of a range of the time indices (R0-R3). The table shows the modeling errors observed when the models computed over some of these ranges were used to predict the values of samples in the other regions. The figure illustrates how the prediction error for region R3 can be vastly reduced by recomputing the model at region R1 (Model 2) or R2 (Model 3), compared to using Model 1, which is computed over R0.

Estimation Range (Indices)	Prediction Model	Prediction range	%Prediction Error
R0	Model1	R1	0.0082
		R2	0.0432
		R3	0.0967
R1	Model2	R2	0.0761
		R3	0.1571
R2	Model3	R3	0.1038

Model coefficients: $[a_0, a_1, a_2, a_3, a_4]$

Model 1: [1, -1.06, .027, .078, .165]

Model 2: [1, -.493, -.097, -.188, -.22]

Model 3: [1, -.97, -.38, -.02, .38]

Comparison of Estimation Ranges :

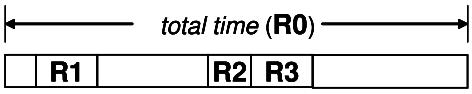


Figure 3.6: Effect of run-time model updates on prediction accuracy for light sensor data (Data set 4).

3.6 Experimental Evaluation

To evaluate our method of model-based correction, we used simulations over multiple data sets. We start with the description of the evaluation setup, and describe the performance of our approach without and with online model updates. For each case, we present the performance of the models, followed by the performance of the overall error detection and correction algorithms. We also show the performance of the correction system when used together with an external error detection system like CRC.

3.6.1 Evaluation Setup

In order to evaluate our model-based correction technique, we implemented our algorithms in C and Matlab ([MAT]), and evaluated their correction performance on real sensor data from different sources, under different levels of simulated error conditions. The specific data sets considered in the evaluation were listed in Table 3.2 on Page 80, along with their sampling period and the number of samples available. Each of the data sets reports information about the sensed physical process as values quantized to 8 bit samples, *i.e.* the data values range from 0 to 255. It is important to note that the data sets differ in terms of autocorrelation properties and degrees of stationarity, which provided an opportunity for evaluating the correction performance under real-world limitations of the model.

As discussed in Section 3.2, we model the errors in the sensor data as random

bit errors. In our model, the data received at the clusterhead is represented as a sequence of 8-bit values, with an independent error probability for each bit. This error represent the transient errors occurring both in the sensor nodes as well as the wireless communication links. For our simulations, We vary the error probability over the range 10^{-4} to 10^{-2}.

The performance of the correction is evaluated in terms of the errors in the sensor values after correction. To capture the higher impact of errors on smaller values and make a fair comparison across data sets with different average levels, we use a relative error measure. Using the notation described in section 3.4, we define the metric as the percent error in corrected output X_c as

$$E_{out} = 100.\sqrt{\frac{1}{N_{tot}}.\sum_{i=1}^{N_{tot}}\left|\frac{X_s(i) - X_c(i)}{X_c(i)}\right|^2}$$

where $X_s(k)$ is the k-th sample originally generated by the sensor, and N_{tot} is the total number of sample values in a data set. In our evaluations, the output error E_{out} is compared to the input error E_{in}, which represents the relative errors in the observed samples as received by the error correction block. E_{in} is computed by replacing X_c with X in the above expression.

3.6.2 Performance with Only Off-line Modeling

Data Modeling

We used AR models for predicting the sensor data for the different data sets mentioned in Table 3.2. For each data set, the model order which minimized the modeling error over the given data set using recursive LS-estimation [Hay96]. The model computed this way for each of the data sets are shown in Table 3.3. The table illustrates the differences in characteristics among the various data sets in terms of the correlation and sampling rate. For example, data set 4 is likely to have been oversampled because of the very low modeling error that could be obtained even with a order 2 model. On the other hand, the models for data sets 2 and 3 have high degrees of randomness, as denoted by the larger modeling errors.

Data Correction

Fig. 3.7 shows the variation of the sensor data (data set 4) with time, well as the effects of errors on the data. It also provides a qualitative idea of the error correction performance of the $Peer$ algorithm. The probability of error for each bit is 10^{-2}, and the large number of sharp peaks in the upper plot illustrate the instances when errors occur in higher order data bits. Some of the errors that were not corrected happen to occur near a point where the sensor data is also changing fast, in which case the modeling error is more likely to be high.

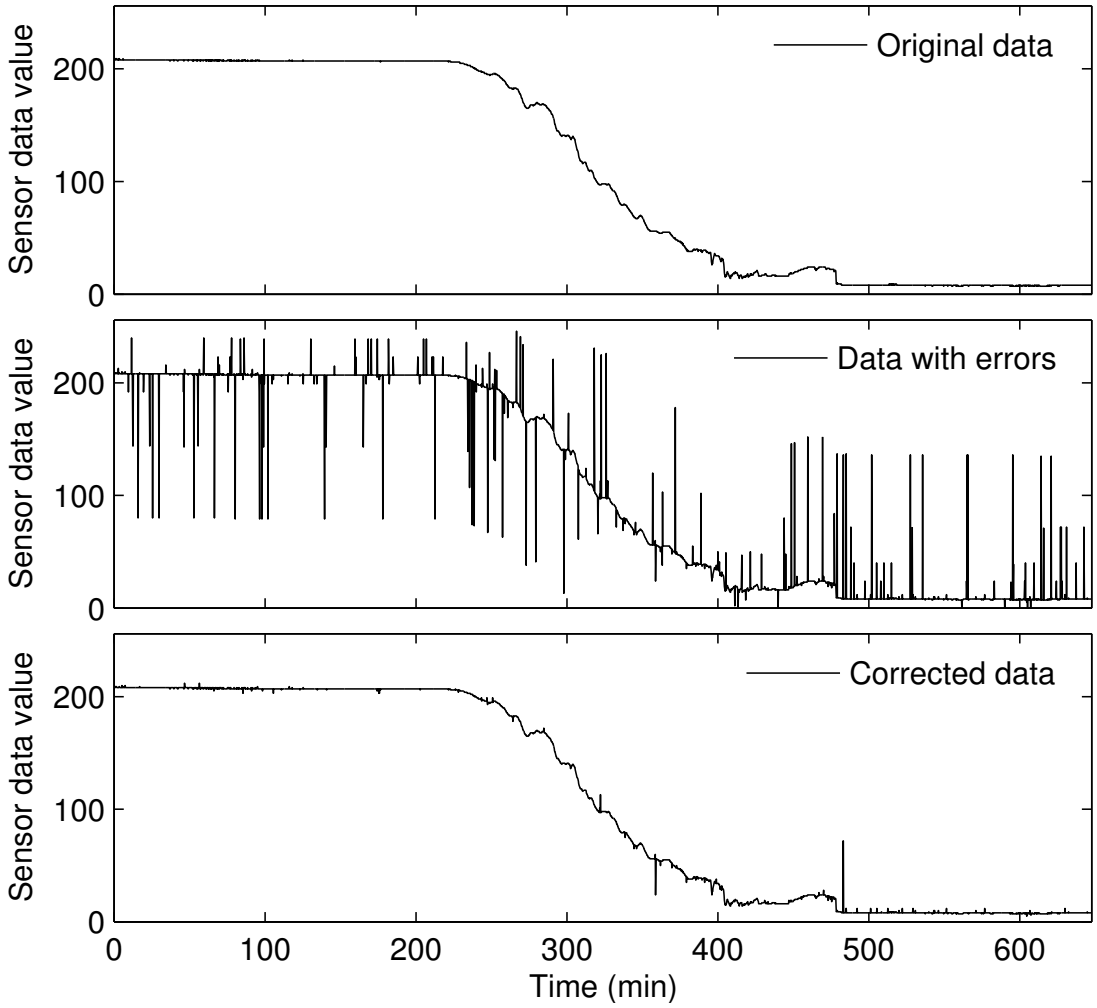


Figure 3.7: Error correction using $Peer$ algorithm on light sensor data (data set 4) with offline data modeling only (*BER*=10^{-2}).

The plots in Fig. 3.8 compare the performances of the three algorithms, $MinErr$, $MinMax$ and $Peer$, when the same model is used without run-time re-estimation. The final correction error (E_{out}) is plotted against the errors in the observed data(E_{in}). The results demonstrate that $Peer$ performs marginally better than $MinMax$ for most of the measured error rates, and they both outperform $MinErr$ substantially. Moreover, these performance gaps keep increasing with rising error levels. The two plots show the difference in comparative performance in two data sets that are different in the distribution of data values and the existing temporal correlation. From Table 3.3 it can be seen that the modeling error for data set 2 is substantially higher than set 4. This agrees with the observation that the final correction errors for data set 4 are smaller as well. For example, the output error in data set 4 at E_{in}=1 is less than the error in data set 2 at E_{in}=0.5. Also, all the curves show a knee above which the output error starts to grow at a faster rate, marking the point where the effects of modeling and observation errors have similar characteristics. The position of the knee increases with the modeling error.

To validate our error models, we also evaluated our algorithms over real communication channel errors, using the same data set as Fig. 3.8(b). The errors were introduced using traces of transmissions between Zigbee CC2420 (Chipcon) radios and GNURadio receivers, which were obtained from [JB07]. The resulting plots in Fig. 3.9 show that the results from our simulations correspond closely with performance over real error sources.

It can be observed from the plots that our method performs well in presence of high error levels, but at very low error levels of input errors there is a non-zero residual error remaining in the output. There are two sources of errors in the output: modeling errors that are introduced when a predicted value is substituted for an erroneous sample, and observation errors that pass undetected because they cannot be distinguished from modeling errors. Moreover, when the modeling errors are high, they can also cause observations to be falsely detected as erroneous by the decision algorithms, in which case the modeling errors get passed into the output. At low input error levels, the output error is thus dominated by the modeling errors, and the residual error is dependent on the model quality. Upon close comparison of the output and the input

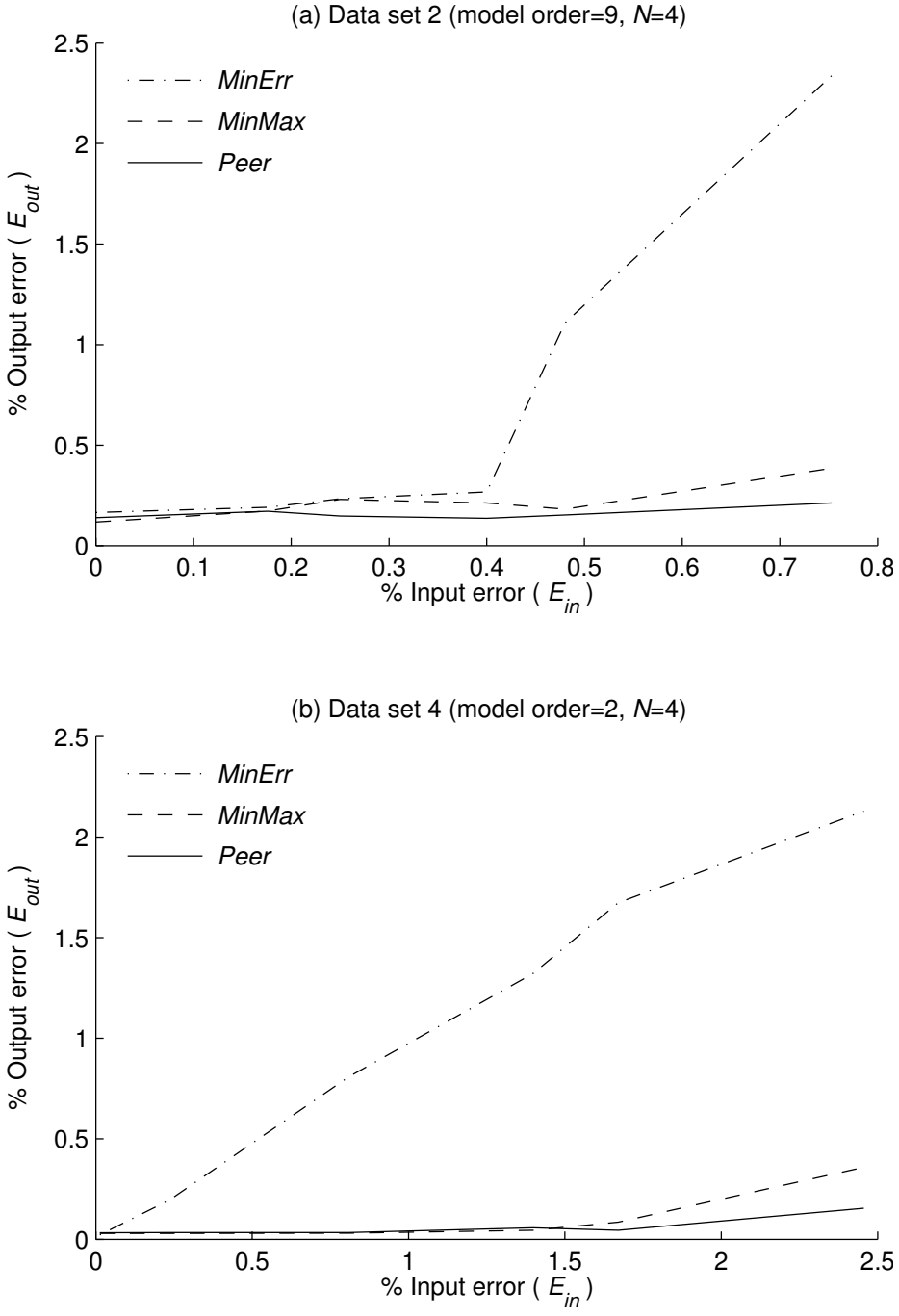


Figure 3.8: Comparison of correction performance of the three decision algorithms using offline modeling. (a) and (b) correspond to two data sets.

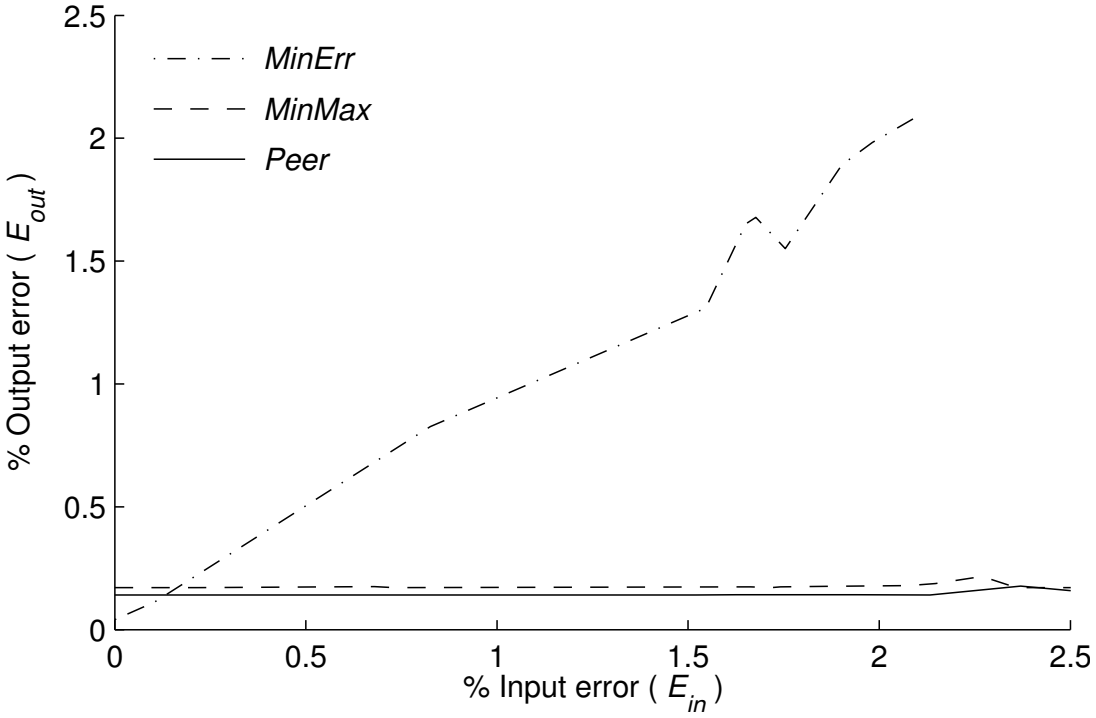


Figure 3.9: Comparison of correction performance of the three algorithms with error traces using offline modeling. *Sensor data set same as in Fig. 3.8(b).*

data, examples of the residual errors were observed where there were sharp and narrow peaks in the data itself, which can happen when the data is under-sampled. The decision algorithms interpret them as erroneous since the line quickly falls back to the previous trend, very much resembling a transient error. However, it should be noted that the rate of these errors is directly connected to a lack of correlation within the data. If this happens frequently, it may be possible to reduce the level of the residual error by adjusting the sampling rate so that no correlated variations in the process are interpreted as errors.

3.6.3 Performance with Run-time Model Estimation

Data Modeling

Table 3.4: Need for model updates: Data set 4, Model order 4

Range of indices for estimation	Data Prediction Model	Prediction Range	Prediction Error (%)
1-3251	$A1$: $1 - 1.06q^{-1} - .027q^{-2}$ $-0.07797q^{-3} + 0.1649q^{-4}$	550-700	0.0082
		1550-1700	0.0432
		1700-2000	0.0967
550-700	$A2$: $1 - .4935q^{-1} - .0974q^{-2}$ $-.1883q^{-3} - .2208q^{-4}$	1550-1700	0.0761
		1700-2000	0.1571
1550-1700	$A3$: $1 - .9714q^{-1} - .3835q^{-2}$ $-.02821q^{-3} + .3829q^{-4}$	1700-2000	0.1038

Table 3.4 shows the effect of updating the model parameters at run-time on the modeling error for the example data set 4. The first row shows the outcome of an idealized best-case scenario for offline-only updates, where all the available data are used to estimate the model parameters. While this is impractical to use, it provides an idea of the minimum modeling error that can be attained by using the largest possible

amount of data for model estimation. The second model $A2$ is estimated using only the early portion of the data (samples 550-700), and shows an increase in modeling error when used for later samples (1550-1700 or 1700-2000). Estimating the model again at 1550-1700 ($A3$) shows an improvement in prediction for subsequent samples. The above results illustrate the need for a way of modeling different parts of the data differently. As mentioned in section 3.5, we do this by run-time re-estimation of the model parameter, whose effect on the overall correction performance is shown below.

Data correction with run-time model estimation

The plot in Fig. 3.10 shows the effect of putting run-time model updates together with the *Peer* algorithm. The plot shows part of the same data with same error level as in Fig. 3.7, but with on line re-estimation of model parameters. The plot shows a better correction performance than without online updates. The time spent in the estimation mode is indicated by the line at the top. Since the variation in the sensor values is very regular, only a small fraction of the time is spent estimating the model in this case.

Table 3.5: Effect of dynamic model updates

Data set ID	E_{in} (%)	E_{out} w/o updates (%)	E_{out} with updates (%)	Factor of improvement (E_{in}/E_{out})	Modeling overhead (%)
1	0.83	0.012	0.008	1.5	1.4
	1.58	0.077	0.034	2.3	3.0
4	0.89	0.030	0.022	1.4	7.5
	2.40	0.240	0.042	5.7	11.1

Table 3.5 shows the performance improvements obtained for two data sets by using model updates. It also shows the overhead for the online estimation, as the ratio of number of samples used for model updates and those corrected using the model. It is observed that for either of the data sets, when the error at the input is higher, the improvement attained with model updates for a given data set is higher. This is expected, since the presence of higher levels of modeling error makes it more likely for larger observation errors to be passed through the correction block, thus degrading

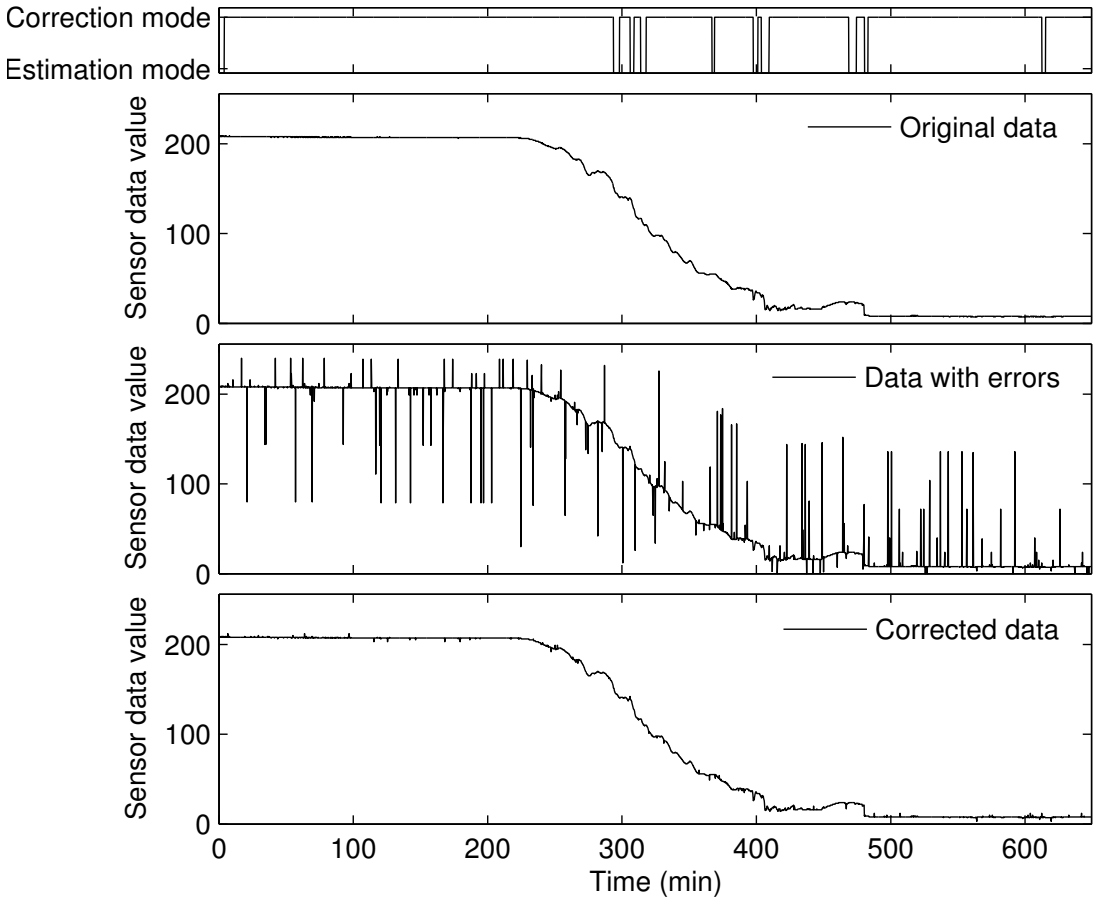


Figure 3.10: Error correction using $Peer$ on light sensor data (data set 4) with run-time model updates (*BER*=10^{-2}). The plot at the top shows when the system state switches between *Estimation* and *Correction* modes.

the performance further. We also notice that the frequency of updates is automatically adjusted based on the level of the error, therefore both data sets have substantially larger modeling overheads when there are more errors at the input.

3.6.4 Comparison with Traditional Approaches

Since the proposed approach addresses errors from multiple sources together, it is difficult to perform a one-on-one comparison of the correction performance with traditional techniques for error handling. Instead, here we compare the costs of using each approach in terms of the resource overheads incurred to perform error correction.

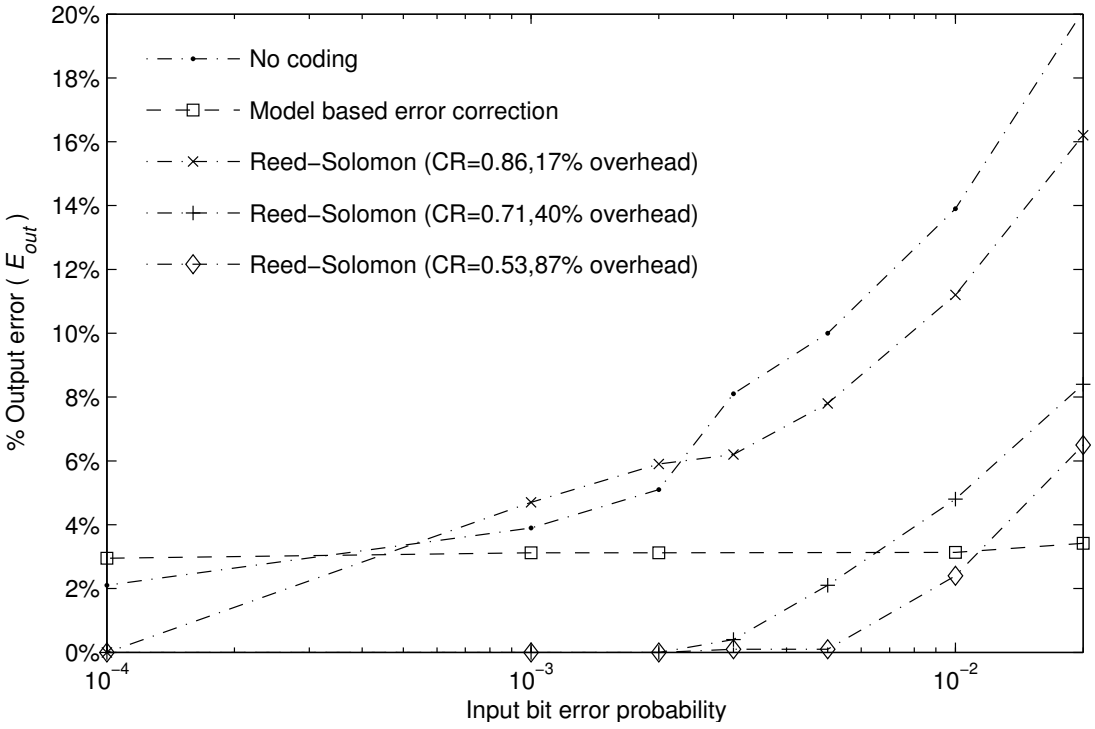


Figure 3.11: Comparison of Model-based error correction with Reed-Solomon Coding at different code-rates (Data Set 4, N=4).

Fig. 3.11 shows the comparison of our approach with Reed-Solomon coding, a traditional forward error correction method for the communication channel. The plot shows the error-correction performance for different configurations of the FEC method with different code rates and overheads. It can be observed that using Reed-

Solomon coding to match the performance of our approach would lead to overheads of up to 86%, for BERs ranging from 10^{-4} to 10^{-2}. This overhead will occur in terms of additional transmitted bits. In comparison, the costs of our method are the storage and processing overheads for the PHT, which had 64 nodes (N=4) for this experiment. Unlike FEC, the overhead for our approach only occurs at the clusterhead.

The traditional methods of hardware error correction are circuit hardening techniques like ECC and TMR. These methods incur overheads in terms of silicon area, up to 25-180% [ZD03]. In comparison, it should be noted that our proposed technique poses no resource or complexity overhead on the sensor nodes to correct transient errors. Moreover, the proposed approach handles all types of random errors, where each of the traditional methods address only one source of error.

3.6.5 Result of using CRC checksum

Fig. 3.12 shows the reliability obtained when the output of CRC checksum can be utilized, simulated with data set 4. For these experiments, the model-based correction approach used the $Peer$ algorithm in Fig. 5, while the hybrid approach used the modified version of the algorithm shown in Fig. 6. The error model used in this simulation was different, including only communication channel errors. The bit-error rate varied from 10^{-3} to 10^{-1} to simulate very strong error conditions. It can be observed from the figure that using the CRC output can reduce the error by about 50% under such high error conditions.

3.7 Related Work

In the second part of the dissertation, we presented a method for improving the reliability of sensor data against specific types of transient errors. We demonstrated that our method can handle uncorrelated errors introduced in the hardware as well as in the communication channel, which are two of the common sources of errors in wireless sensor networks. Below, we first present an overview of prior work that address different types of data unreliability in wireless sensor networks. Later, we

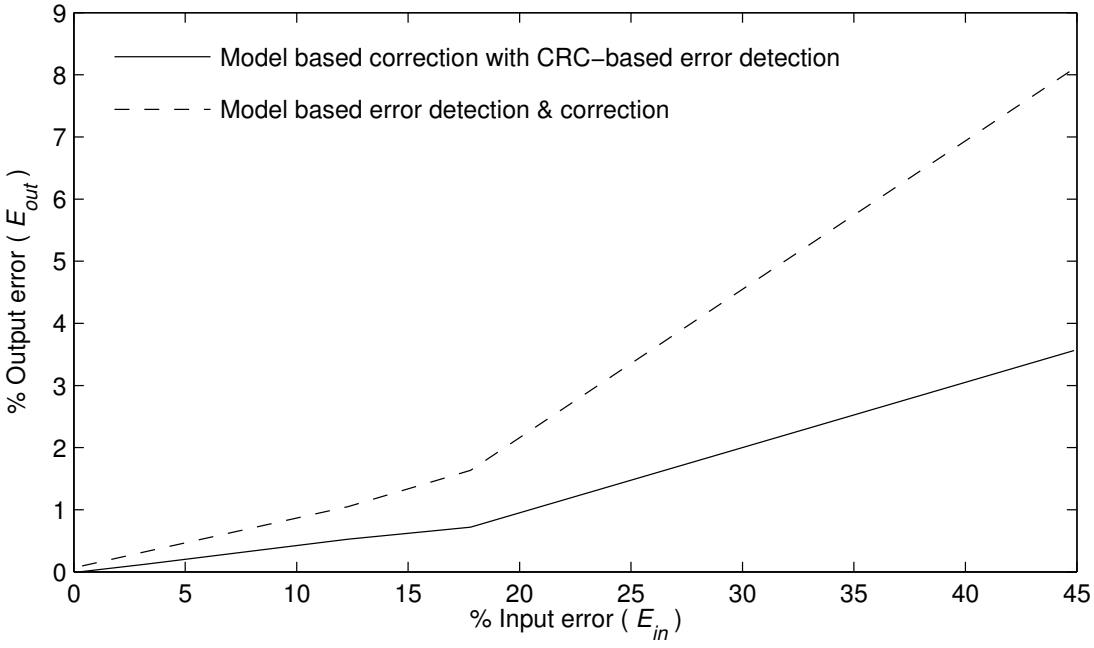


Figure 3.12: Error correction performance for hybrid approach (with CRC).

also discuss approaches that are traditionally used to target the same types of errors discussed in here.

The problem of ensuring data reliability is considered to be an important problem in sensor networks, and many techniques have been developed to address various types of reliability problems. In one of the most closely related works, the authors described a method which corrects measurement errors at the sensor based on prior knowledge of the statistical properties (distribution) of the data and an error model[EN03]. While there is a similarity in the main idea of using data models, this approach is designed only for additive errors with a small variance. On the other hand, the focus of our problem is the occurrence of errors in the circuits and the communication channels, which have non-linear effects on the sensor data values. Due to this non-linearity, these errors do not benefit from representation as additive models, and cannot be effectively corrected by such techniques designed to handle additive errors.

Permanent sensing errors like bias or faulty sensors have been addressed by calibration methods [BMEP03] or distributed detection schemes [LDH06b, KI]. Application level techniques have been proposed to address coverage or event detection

goals in spite of erroneous samples or malfunctioning nodes, like query processing in presence of erroneous data [HHMS03], or localization [SGMS05], where known properties of the data are used to maximize the probability of detection. Also, problems like link or node outages have been addressed by robust aggregation and routing techniques that ensure the reliability of the data aggregation tree [MNG05, KDG03, SH05, DGSE02]. These techniques are orthogonal to our approach, and can be used on top of our methods to address other failures.

We note that our approach of handling the errors in the post-processing stage with online updates is closely related to filtering techniques like smoothing[Hay96]. However, such techniques are typically based on linear additive error models. On the other hand, modeling the random-bit-errors as modeled as additive error leads to complex statistical properties, so that many of the common filtering techniques can not be applied. There are non-linear filtering techniques that could be designed for this error model, however we believe that the correlative properties of the sensor data offer us a simpler way of distinguishing errors from data.

The transient failures in the communication channel or hardware that cause loss or corruption of individual data bits have been a focus of extensive research outside the context of sensor networks as well. For circuit-level problems, there have been replication-based methods proposed to mitigate the effects of soft errors, like Triple Modular Redundancy or Error-corrected memory [ZD03, ZD06]. However, the overhead of replication makes them unsuitable for use in sensor networks.

Studies on real deployments have shown the losses due to communication errors to be significantly higher in sensor networks than in other wireless networks [ZG03]. Traditional ways of handling channel errors in wireless communication include various types of FEC e.g., Reed-Solomon, Turbo codes, all of whom have overheads in terms of transmitted data and energy [Wic95]. Some channel coding schemes have been proposed for wireless sensor networks, but these still incur overheads at the sender for computing the codes as well as for additional transmitted bits [HSI06], [JE06]. Another approach has been to add reliability in MAC or Transport layers, but these are based on packet retransmissions and hence incur even higher energy overheads [SH03], [WCK04]. On the other hand, our approach is based on adaptive

post-processing, and requires no overheads on the sensor nodes. One area where a similar approach is used is in voice communication over wireless links. As in our approach, the correlation in the data source makes it possible to handle erroneous frames by just replacing them with repetitions of the previous frames [SS99].

Another approach to handle communication errors has been in the context of multi-hop networks, where the effect of errors on routing and clustering can be taken into account for better performance [C+07],[VA06]. A different approach has been taken in joint source-channel coding [C+05, GV05], which try to optimize the communication architecture for properties of the data source to minimize the effect of communication errors. However, they require additional customization or processing to adapt to the properties of the data source which, in turn, increases the overhead on the sensor nodes.

While we use the correlation properties in the sensor data to distinguish data from errors, another approach for system design is to remove this correlation through compression. Several techniques have been proposed which use the correlations of sensor data to develop efficient compression algorithms ([PR00]), reduce memory usage for routing algorithms ([D+03]), etc. These methods do not consider the footprint limitations on the sensor nodes and are thus suitable for alternate system architectures with more complex end devices. On the other hand, our approach enables the application of the design principle proposed in Chapter 1, and enables drastic reductions in the footprints of the sensor nodes.

3.8 Scope and Limitations

The scope of our approach for reliability in wireless sensor networks can be defined in terms of the type of sensor data as well as errors. Below, we outline the scope and limitations of our approach.

Data Properties

Applications for wireless sensor networks can be classified into two types based on their primary functionality: event detection and sampling. Event detec-

tion networks are mainly interested in identifying and reacting to individual physical events or changes in the state of their surroundings. On the other hand, sampling applications are designed to collect and document the variations in physical processes over time and space for future analysis. There are further differences in the application characteristics among the event-detection systems. For some, the events of interest appear as sudden changes, perhaps for a single sample, while in others these events or changes consist of longer lasting, more complex patterns involving multiple patterns.

We note that the approach for reliability proposed in this dissertation cannot be applied effectively to the first type of event-detection applications. In these networks, interesting events can consist of sudden changes lasting only one or two samples, which would appear as anomalies or outliers. In our approach, individual outliers are ignored as errors, while frequently occurring ones may be used to trigger a change in the model. Therefore, it is not a good fit for a network whose main interest is in these outliers.

On the other hand, our approach works for other event-detection networks where the relevant changes in the data are expected to be slow, e.g., triggering a fault report when some equipment begins to perform below specifications, turning a fan on when the temperature in a room rises above a threshold or a sprinkler when humidity in the soil falls below one. Our work is also applicable to sampling or metering networks, like recording light levels or water flow, where the regular observations are more important than outliers. Furthermore, sensor networks where important events are detectable from single samples are likely to have some processing performed on them before reporting. Such applications would require more complex sensor nodes, which are unlikely to use the type of network architecture we proposed in Section 3.1.

Error Characteristics

The applicability of our approach is also shaped by the types of errors addressed by it. As discussed in detail in Section 3.2.1, these errors consist of transient errors introduced in the storage/processing hardware in the sensor nodes, and during wireless communications. The main characteristic of these errors is that they manifest

as inversion of random bits in the data. Representing such errors as additive leads to complex statistical properties, which is why this type of errors does not lend itself to common linear filtering or smoothing techniques that are designed for additive linear errors arising from sampling inaccuracies or bias in the sensor. Instead, our approach presents a complementary solution to other methods that correct additive sensing errors.

3.9 Conclusions

In this chapter, we investigated transient data errors in wireless sensor networks, and examined the problems of trying to apply traditional error correction methods in this context. We presented a novel approach to sensor network design that uses specific properties of sensor data to enable reliable data collection at no additional cost to the sensor nodes. We show that our approach, called model-based error correction, corrects transient errors introduced in the sensor node hardware as well as in the wireless communication channel. Through simulation-based study on real sensor data, we demonstrate that with the proposed enhancements, the presented framework can be used efficiently to address transient errors in different types of sensor data across diverse applications. We also implemented the correction algorithm in software on a sensor testbed.

The text of this chapter, in part, is based on material that has been published in the IEEE Wireless Communications and Networking Conference in 2004 (S. Mukhopadhyay, D. Panigrahi, and S. Dey, "Data Aware, Low Cost Error Correction for Wireless Sensor Networks", *Proceedings of IEEE Wireless Communications and Networking Conference* , Atlanta, USA, March 2004), the IEEE Sensor and Ad Hoc Communications and Networks Conference(SECON), (S. Mukhopadhyay, D. Panigrahi, S. Dey, "Model Based Error Correction for Wireless Sensor Networks", *Proceedings of IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks*, Santa Clara, October 2004), and accepted for publication in the IEEE Transactions on Mobile Computing (S. Mukhopadhyay, C. Schurg-

ers, D. Panigrahi, S. Dey, “Model Based Techniques for Data Reliability in Wireless Sensor Networks”, *IEEE Transactions on Mobile Computing*). The dissertation author was the primary researcher and author in all three publications, and the coauthors listed collaborated on or supervised the research that forms the basis of this chapter.

Chapter 4

Conclusions and Future Directions

This chapter concludes the dissertation with a summary of our principal contributions and some thoughts about the implications of this research for future generations of systems.

The underlying idea of this work has been the architectural principle of splitting the functionality of the end nodes, as presented in Figure 1.3 in Chapter 1. In subsequent chapters, we looked at two different types of wireless networks where this idea was realized to satisfy different goals specific to each system.

In Chapter 2, we presented an architecture for a wireless network where the wireless access points are enhanced with embedded processing resources, which enables them to support some of the application-specific processing for the client nodes. For this application, we addressed the problem of sharing the processing resources in the network among end nodes, by a designing a set of heuristic solutions for joint scheduling of computation and communication. We showed that these heuristics allow efficient usage of these shared resources in such a network architecture. As a result, this architecture can be successfully used to enable rich applications with enhanced functionality, while continuing to use low-footprint devices as clients.

In Chapter 3, we focused on wireless sensor networks, and showed how a hierarchical network architecture can allow the use of ultra low-footprint sensor nodes, and still remain capable of performing reliable data collection. To achieve this, we presented a method for correcting errors in sensor data that uses patterns and prop-

erties of the sensor data itself. Our method moved the processing overheads of error correction out of the thin sensor nodes and into the more powerful cluster-head nodes, and in the process effectively allowed the sensor nodes to be made smaller and thinner.

Though the types of systems covered in the last two chapters differed drastically in terms of goals and functionality, they can both be considered to be instantiations of the same architectural principle that was presented in Chapter 1. In both cases, the proposed architectural idea involved splitting what was formerly the functionality of the end nodes into two different parts. In the first instance, the resulting architecture allows the extension of the functionality of mobile applications even as the footprints of the end nodes continued to remain low. In the second case, the use of the same principle enables significant reduction in the footprints of the sensor nodes without sacrificing the primary functionality of the system, which is reliable data collection.

The two examples illustrate the ability of the proposed architectural principle to solve some of the technical challenges of next generations of wireless computing systems. As discussed in Chapter 1, there has been a convergence of computing and communication functionalities in the present generation of wireless networks, and future generations of these systems have to address the twin goals of enhanced functionality and reduced footprints. For any given generation of technology, any design involves a trade-off between these two goals. What we have demonstrated in this dissertation is that another approach, consisting of an architectural redesign, can transcend this trade-off in a way that is not feasible without breaking the current architecture. The architectural change comes with its own effects on the application design, and in the previous chapters we demonstrated how some of these challenges can be addressed in the context of two types of systems with widely disparate requirements and constraints.

We envision this idea to be an initial step in the direction of new design techniques specialized for future systems that emerge out of the convergence between computing and communication applications. The design of any such system involves a consideration of numerous trade-offs, and our work serves to hand an extra option to the designer in terms of balancing requirements and capabilities. In the future, we envision this architectural principle to become more generalized by designs that

push more and more of the user-specific functionality into the network. In the manner of the blurring of the lines between computing and communication systems, we expect this approach to eventually blur the boundaries between the user and the system, the additional flexibility allowing a designer to generate more efficient designs. As demonstrated in this dissertation, this holds the promise for increasing the capabilities of these systems, while at the same time making them more and more efficient.

Bibliography

[AC05] R.K. Ahuja and C.B. Cunha. Very Large-Scale Neighborhood Search for the K-Constraint Multiple Knapsack Problem. *Journal of Heuristics*, 11(5-6):465–481, 2005.

[Aka69] H. Akaike. Fitting autoregressive models for prediction. *Ann. Inst. Stat. Math*, 21:243–247, 1969.

[Bak05] T.P. Baker. An analysis of EDF schedulability on a multiprocessor. *Parallel and Distributed Systems, IEEE Trans. on*, 16(8), 2005.

[Bau03] R. Baumann. Technology scaling trends and accelerated testing for soft errors in commercial silicon devices. In *On-Line Testing Symposium, IOLTS. 9th IEEE*, page 4, 2003.

[Bau05] R.C. Baumann. Radiation-induced soft errors in advanced semiconductor technologies. *Device and Materials Reliability, IEEE Transactions on*, 5(3):305–316, 2005.

[BC03] S. Bandyopadhyay and E.J. Coyle. An energy efficient hierarchical clustering algorithm for wireless sensor networks. In *22nd IEEE International Conference on Computer Communications. INFOCOM*, volume 3, pages 1713–1723, 2003.

[Ber99] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 1999.

[BMEP03] V. Bychkovskiy, S. Megerian, D. Estrin, and M. Potkonjak. A collaborative approach to in-place sensor callibration. In *Proc. of Intl. Workshop on Information Processing in Sensor Networks(IPSN)*, 2003.

[CDE] CDEC: California data exchange center. http://cdec.water.ca.gov/.

[Cha04] C. Chatfield. *The Analysis of Time Series: An Introduction*. CRC Press, sixth edition, 2004.

[EN03] E. Elnahrawy and B. Nath. Cleaning and querying noisy sensors. In *Second ACM Intl. Workshop on Wireless Sensor Networks and Applications (WSNA)*, 2003.

[F+06] N.A. Ferzli et al. An application of smart dust for pavement condition monitoring. *Smart Structures and Materials : Proc. of the SPIE*, 6174:976–987, 2006.

[H+03] P. Hazucha et al. Neutron soft error rate measurements in a 90-nm cmos process and scaling trends in sram from 0.25-μm to 90-nm generation. In *Electron Devices Meeting, IEDM Technical Digest. IEEE International*, pages 21.5.1–21.5.4, 2003.

[Har89] A. Harvey. *Forecasting, Structural Time Series Models and the Kalman Filter*. Cambridge Univ. Press, Cambridge, MA, 1989.

[Hay96] S. Haykin. *Adaptive Filter Theory*. Prentice-Hall, Inc, 2nd edition, 1996.

[HC05] M. Hatler and C. Chi. Wireless sensor networks: Growing markets, accelerating demand. Technical report, ON World, October 2005.

[HSW+00] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. In *Proc. of the ninth Intl. Conf. on Architectural support for programming languages and operating systems*, pages 93–104. ACM Press, 2000.

[IHK06] K. Itoh, M. Horiguchi, and T. Kawahara. Ultra-low voltage nano-scale embedded RAMs. In *Proc. IEEE Intl. Conf. on Circuits and Systems (ISCAS)*, pages 25–28, 2006.

[JB07] K. Jamieson and H. Balakrishnan. Ppr: partial packet recovery for wireless networks. *SIGCOMM Comput. Commun. Rev.*, 37(4):409–420, 2007.

[JSP07] A. Jow, C. Schurgers, and D. Palmer. Calradio: a portable, flexible 802.11 wireless research platform. In *MobiEval '07: Proceedings of the 1st international workshop on System evaluation for mobile platforms*, pages 49–54, New York, NY, USA, 2007. ACM.

[KH04] T. Karnik and P. Hazucha. Characterization of soft errors caused by single event upsets in CMOS processes. *Dependable and Secure Computing, IEEE Transactions on*, 1(2):128–143, 2004.

[KSW98] H. Kantz, T. Schreiber, and D. Wojcik. *Nonlinear Time Series Analysis*. Pure and Applied Geophysics. Birkhauser Verlag, 1998.

[MAT] MATLAB: A high-level technical computing environment. http://www.mathworks.com/products/matlab.

[MER05] S.S. Mukherjee, J. Emer, and S.K. Reinhardt. The soft error problem: an architectural perspective. In *High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on*, pages 243–247, 2005.

[MI06] I. Mahgoub and M. Ilyas. *Smart Dust:: Sensor Network Applications, Architecture, and Design*. CRC Press, 2006.

[MKK05] S.S. Mitra, N.M.Z.Q.S. Kee, and S. Kim. Robust System Design with Built-In Soft-Error Resilience. *Computer*, 38(2):43–52, 2005.

[MPD04] S. Mukhopadhyay, D. Panigrahi, and S. Dey. Data aware, low cost error correction for wireless sensor networks. In *Proc. IEEE Wireless Communications and Networking Conference*, pages 2492–7, Atlanta, GA, USA, Mar. 2004.

[N+05] L. Nachman et al. The intel mote platform: a bluetooth-based sensor network for industrial monitoring. In *Proc. of 4th Intl. Symp. on Information Processing in Sensor Networks*, page 61. IEEE Press, 2005.

[NS] The Network Simulator - ns2. http://www.isi.edu/nsnam/ns/.

[ONW05] Wireless sensor networks market expected to skyrocket. http://www.controldesign.com/industrynews/2005/040.html, 2005.

[S+02] P. Shivakumar et al. Modeling the effect of technology trends on the soft error rate of combinational logic. In *Dependable Systems and Networks. Proc. Intl. Conf. on*, pages 389–398, 2002.

[S+04] R. Szewczyk et al. Application driven systems research: Habitat monitoring with sensor networks. *Communications of ACM Special Issue on Sensor Networks*, pages 34–40, June 2004.

[Sch06] G. Schindlbeck. Trend in DRAM soft errors. In *On-Line Testing Symposium. IOLTS. 12th IEEE Intl.*, page 272, 2006.

[SH05] S. Soro and W.B. Heinzelman. Prolonging the lifetime of wireless sensor networks via unequal clustering. In *Parallel and Distributed Processing Symposium, Proceedings. 19th IEEE International*, page 8 pp., 2005.

[SKM04] L. Sankaranarayanan, G. Kramer, and N.B. Mandayam. Hierarchical sensor networks: capacity bounds and cooperative strategies using the multiple-access relay channel model. In *Proc. IEEE Sensor and Ad Hoc Communications and Networks(SECON)*, pages 191– 199, 2004.

[SS99] C. Schurgers and M.B. Srivastave. Voice over wireless internet: performance interaction of signal processing algorithms and network protocols. In *Vehicular Technology Conference, 1999 IEEE 49th*, volume 3, pages 1935–1939 vol.3, Jul 1999.

[TJ99] M. Thottan and C. Ji. Fault prediction at the network layer using intelligent agents. In *Integrated Network Management. Proc. of the Sixth IFIP/IEEE Intl. Symposium on*, pages 745–759, 1999.

[VAA04] M. C. Vuran, B. O. Akan, and I. F. Akyildiz. Spatio-temporal correlation: theory and applications for wireless sensor networks. *Comput. Networks*, 45(3):245–259, 2004.

[Wic95] S. B. Wicker. *Error Control Systems for Digital Communication and Storage*. Prentice Hall, 1995.

[WLLP01] B. Warneke, M. Last, B. Liebowitz, and KSJ Pister. Smart Dust: communicating with a cubic-millimeter computer. *Computer*, 34(1):44–51, 2001.

[WM06] A. Willig and R. Mitschke. Results of Bit Error Measurements with Sensor Nodes and Casuistic Consequences for Design of Energy-Efficient Error Control Schemes. *Proceedings of EWSN, Zurich, Switzerland*, 2006.

[XBO] Crossbow technology. http://www.xbow.com.

[ZBD04] C. Zhao, X. Bai, and S. Dey. A scalable soft spot analysis methodology for compound noise effects in nano-meter circuits. In *Proc. of the 41st annual conference on Design automation*, pages 894–899, 2004.

[ZBD05] C. Zhao, X. Bai, and S. Dey. A static noise impact analysis methodology for evaluating transient error effects in digital VLSI circuits. In *Proc. of Intl. Test Conference*, page 40.2, Austin, Texas, USA, October 2005.

[ZD03] Y. Zhao and S. Dey. Separate dual transistor registerss: A circuit solution for on-line testing of transient errors in UDSM-IC. In *Proc. of Intl. Online Testing Symposium*, pages 7–11, 2003.

[ZS04] M. Zhang and N. R. Shanbhag. A soft error rate analysis (sera) methodology. In *ICCAD '04: Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design*, pages 111–118, Washington, DC, USA, 2004. IEEE Computer Society.