IMPROVED NETWORK CONSISTENCY AND CONNECTIVITY IN MOBILE AND SENSOR SYSTEMS

A Dissertation Presented

by

NILANJAN BANERJEE

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

August 28, 2009

Computer Science

IMPROVED NETWORK CONSISTENCY AND CONNECTIVITY IN MOBILE AND SENSOR SYSTEMS

A Dissertation Presented

by

NILANJAN BANERJEE

Approved as to style and content by:

ACKNOWLEDGMENTS

I would first take this opportunity to thank my academic advisor Prof. Mark D. Corner without whose able guidance I would not be a doctorate. When I joined UMass I was inclined towards theoretical computer science and had little idea of what computer systems research looked like. In these five years, Mark has completely and successfully converted me from a theorist to a researcher who applies strong theoretical foundations to build real systems which impact real people.

Next, I am indebted to Prof. Brian Levine who has been my (unofficial) co-advisor for the last five years. His insights into the design of the Throwbox and Epsilon systems were invaluable. Moreover, personally he has always encouraged me (as his other students) to follow my personal intuition towards solving research problems. I would also like to thank Prof. Don Towsley, Dr. Kevin Fall, Prof. Sami Rollins, Dr. Sharad Agarwal, and Prof. Deepak Ganesan with whom I have worked very closely on various projects which have (and will) lead to many high quality research papers. I would also like to thank Prof. Dennis Goeckel whose insight and comments on my thesis have been invaluable.

I would like to thank my colleagues Jacob Sorber and Hamed Soroush with whom I have worked on several projects which form part of this dissertation. Jacob and I developed the Turducken and Triage systems. Hamed and I developed the Epsilon system described in this thesis. I am obliged to my mother and my fiancee Neha Raikar without whose personal support "hanging on" these four and a half years as a masters and Ph.D. student would have been impossible. Finally, I would like to thank all the PRISMSers (present and past) for being great friends and lab mates.

ABSTRACT

IMPROVED NETWORK CONSISTENCY AND CONNECTIVITY IN MOBILE AND SENSOR SYSTEMS

August 28, 2009

NILANJAN BANERJEE

B.Tech (Hons), INDIAN INSTITUTE OF TECHNOLOGY, KHARAGPUR

M.S, UNIVERSITY OF MASSACHUSETTS, AMHERST

Computer Science, UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Mark D. Corner

Edge networks such as sensor, mobile, and disruption tolerant networks suffer from topological uncertainty and disconnections due to myriad of factors including limited battery capacity on client devices and mobility. Hence, providing reliable, always-on consistency for network applications in such mobile and sensor systems is non-trivial and challenging. However, the problem is of paramount importance given the proliferation of mobile phones, PDAs, laptops, and music players.

This thesis identifies two fundamental deterrents to addressing the above problem. First, limited energy on client mobile and sensor devices makes high levels of consistency and availability impossible. Second, unreliable support from the network infrastructure, such as coverage holes in WiFi degrades network performance. We address these two issues in this dissertation through client and infrastructure end modifications.

The first part of this thesis proposes a novel energy management architecture called Hierarchical Power Management (HPM). HPM combines platforms with diverse energy needs and capabilities into a single integrated system to provide high levels of consistency and availability at minimal energy consumption. We present two systems Triage and Turducken which are instantiations of HPM for sensor net microservers and laptops respectively. The second part of the thesis proposes and analyzes the use of additional infrastructure in the form of relays, mesh nodes, and base stations to enhance sparse and dense mobile networks. We present the design, implementation, and deployment of Throwboxes a relay system to enhance sparse mobile networks and an associated system for enhancing WiFi based mobile networks.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Unlike the wired core of the Internet, edge networks like mobile, sensor, and disruption-tolerant networks suffer from topological uncertainty and disconnections due to a myriad of factors including limited battery capacity on client devices, radio characteristics, and mobility [55, 107, 147, 149]. Hence, providing reliable, improved connectivity and always-on consistency for network applications in such mobile and sensor systems is non-trivial and challenging. However, the problem is of paramount importance given the proliferation of mobile and sensor devices like cellular phones, laptops, PDAs, and music players [1, 7].

Some of the above problems have been addressed in isolation. For example, energy management in embedded and mobile systems has been an active area of research for nearly a decade [29, 75, 85]. However, most solutions propose using power state optimizations such as CPU voltage and frequency scaling [58], disk spin-down [122] or power saving modes on wireless cards [13] which can lead to high idle power consumption of the system. As we move towards an era where services need high network consistency, we require techniques, which differ radically from what state-of-the-art systems can provide. Similarly, a large body of research has looked at the problem of providing enhanced connectivity in both sparse and dense mobile networks [174]. However, most of the proposals are simulations with little-to-no real world validation. This thesis addresses the above problems through the design, implementation, and deployment of robust, reliable, and highly optimized mobile and sensor systems capable of providing improved connectivity and availability in a diverse range of pervasive scenarios.

Enhanced Connectivity and Consistency in Mobile and Sensor Systems

The primary focus of this dissertation is on how to provide improved connectivity and improved availability in mobile and sensor systems. The problem is challenging because of two factors. For a client device like a laptop or PDA, providing improved communication is difficult due to limited battery capacity [145]. For example, with the wireless radio on, the maximum battery lifetime of a laptop is a few hours [59]. This leads to the following question: *Is it possible to provide always-on operation at minimal energy consumption for mobile and sensor devices?* However, an always-on mobile client is not useful without available connectivity. For example, consider popular organic WiFi deployments. While open WiFi access can provide network coverage in large fraction of cities [136], many regions have little or no connectivity [148]. Even in areas where access point density is high, coverage holes are common due to channel interference and obstacles [107]. A similar problem exists in sparse mobile networks such as wildlife tracking networks [147, 167]—packet delivery performance in such networks is poor since nodes solely depend on meager mobile-to-mobile contacts to transfer data. This observation leads to a second question: *Can inexpensive enhancements to already-existing networks provide dramatic improvement in performance?* This dissertation provides the basis to answer these two questions through client and infrastructure end modifications.

Client-end modifications

Mobile devices such as laptops do not have adequate battery capacity for constant processing and communication. For example, even by powering off unnecessary components, such as the screen and disk [122], current laptops only have a lifetime of a few hours. However, while PDAs and sensors are similarly limited in lifetime, a PDA's power requirement is an order-of-magnitude smaller than that of a laptop, and a sensor's is an order-of-magnitude smaller than that of a PDA. This thesis proposes a new architecture called Hierarchical Power Management (HPM) which combines platforms with such diverse energy needs and capabilities into a single mobile or sensor platform. Through intelligent task sharing and duty-cycling of these platforms, HPM provides high levels of availability at minimal energy consumption. We present the design and implementation of Triage and Turducken—two instantiations of HPM for building sensor net microservers and laptops. Both systems can provide high levels of availability and consistency while consuming 2x-6x less energy than state-of-the-art solutions.

Infrastructure enhancements

While HPM clients can provide always-on operation at minimal energy consumption, without proper infrastructure support the goal of improved connectivity will remain a myth. For example, in sparse mobile networks such as wildlife networks [82, 147] mobile nodes may be highly energy efficient, but network performance can be dismal since packets are transferred only through node mobility. Similarly, organic WiFi deployments and cellular networks (like 3G, GPRS) fail to provide 100% reliable connectivity in dense mobile networks—coverage holes are common even in networks planned for blanket coverage.

The second part of this thesis proposes improving mobile network performance through the addition of supporting infrastructure such as base stations, meshes, and relay nodes. We first study the trade-offs for each type of enhancement through a comparison study using a large scale deployment and a demonstrably accurate ODE-based analytical model. Encouraged by the enhancement estimates that relays can provide, we present the design, implementation and evaluation of a Throwbox, an architecture for a solar powered stationary node capable of relaying traffic between mobile nodes. Finally, this dissertation explores the feasibility of enhancing open WiFi access point networks with a 900MHz long range low bitrate bridge.

Thesis Organization

The organization of this thesis is as follows. Chapter 2 motivates the problems addressed in this dissertation in the broader context of related literature. Chapter 3 describes the mobile network testbed used in the thesis to deploy and evaluate several of our systems. Chapter 4 introduces the idea of Hierarchical Power Management as an effective paradigm for energy management in mobile and sensor systems. Chapter 5 presents a comparison of alternative infrastructure in the form of relays, mesh, and base stations for enhancing mobile networks. Chapter 6 presents the design, implementation, and evaluation of Throwboxes. Chapter 7 motivates the need to enhance WiFi based mobile networks and presents the design of 900MHz bridge nodes to patch WiFi coverage holes. Chapter 8 concludes the thesis and presents future directions of research.

CHAPTER 2

MOBILE SYSTEMS, ENERGY, AND INFRASTRUCTURE

Advances in wireless and sensor systems has fueled the growth and evolution of a new class of computing: *mobile computing*. The growing importance of this field is clearly illustrated by the popularity of mobile handhelds—in 2009, the number of cellular phones, Bluetooth enabled devices, and WiFi enabled smartphones worldwide is estimated at 269, 583, and 141 million respectively [4–6]. In addition to mobile handhelds, another class of mobile networks has seen rapid growth—those based on sensor systems and radio technologies such as Zigbee, IEEE 802.15.4, and Z-Wave [10, 57, 82, 147, 157]. Due to the confluence of these diverse technologies, mobile systems are poised to have myriad applications. An incomplete list include location-aware services [62, 90, 113, 124, 159], social networking applications [2, 3, 8, 9, 53, 93, 113], wearable computing [39, 61, 92], and peer-based services [21, 21] on mobile handhelds, and wildlife tracking [57, 73, 82, 147, 157], robotics [49, 95], underwater networking [30, 123, 131, 143], and battlefield applications [49, 95, 172] on sensor platforms.

While the possibility of mobile applications are endless, on the flip side mobile systems research confronts unique challenges such as energy bottleneck on clients, network disconnections due to mobility, channel and radio characteristics, and scanty node density. To address every known mobile computing issue is a gargantuan task, if not intractable. The key insight lies in identifying deterrents that are most relevant to mobile networks of the future.

2.1 Mobile networks of the future

Trends in mobile computing over a last decade points to two broad categories of applications which are likely to grow: (a) a wide corpora of common applications such as web-based email [25,60], search [18,20], voice [12,19,44,164], and location-aware services has been and will remain popular in the future (b) Monitoring and surveillance applications such as wildlife tracking [57,147], ocean/river monitoring, robotics [26,46,74,141], and defense [68,112] has also received considerable attention. The above applications combined with the proliferation of handheld and micro-controller class devices are indications that two broad mobile network classes will emerge and evolve.

- *One-hop mobile networks:* These are networks comprising of laptops, PDAs, and cellular phones where clients either on foot [41,76,120] or in vehicles [33,54,78] desire connectivity to an Internet server using either free roadside WiFi or an expensive cellular infrastructure such as 3G or GPRS. In such a network, client devices have one-hop wireless connectivity to the Internet through open WiFi access points or 3G towers. Applications in such networks use disruption tolerant techniques [19,37,80] (such as in email) or common transport protocols [14,15,69,109,165] (such as in web email, web search) to forward data.

- *Highly partitioned multi-hop networks:* These are networks of embedded sensor class devices such as wireless Motes [129,130], Stargates [47], Gumstix [71], or XYZ-motes [105]. Examples of such networks include wildlife networks, underwater networks, and battlefield networks. Nodes in these networks transfer data using disruption tolerant techniques [80]. When two nodes are within communication range of each other, they exchange data—the receiver node *stores* this data and uses mobility to *carry* it. Finally, data is forwarded using a base station to an Internet server.

These two classes of mobile networks span opposite ends of the node density spectrum. While one-hop networks such as open WiFi vehicular networks are fairly dense, partitioned mobile networks such as wildlife networks are extremely sparse. In spite of the differences in node density, performance, requirements, and goals of these networks, two common challenges emerge. First, client devices in both networks are untethered battery powered devices. Hence, energy management is a common concern. While embedded devices are becoming more capable with powerful processors, more storage and RAM, improvement in battery capacity has been slow. This makes energy a primary constraint/bottleneck in such systems. Second, in both types of networks connectivity and network capacity are primary determinants of network performance. In highly partitioned networks where nodes forward data using mobility, network performance could be dismal due to scarce network capacity. Moreover, *uncertainty* in node meetings due to nodes mobility makes network performance unpredictable.

2.2 Energy as a first class concern

Component	Active Power (W)	Idle Power (W)
CPU (1.3GHz)	27.45	16.90
Hard Drive	0.58	2.78
LCD Screen	1.01	0 (off)
Wireless Card	3.69	0.39

Table 2.1. Idle and Active power measurements on an IBM R40 Thinkpad [13, 108].

Mobile and untethered systems are synonymous. Whether it is a laptop, PDA, sensornet microserver, or a Turtle or Zebra tracking device [82, 147], all systems are powered by a battery or renewable energy source such as solar. Ideally, these client devices should be highly available to support a wide range of applications and services. For example, laptops and PDAs should be always on to support web search or instant messaging; sensornet microservers should be available to serve queries from client nodes. However, component energy consumption and limited battery capacity on these devices make this goal of constant vigilance difficult. To quantitatively understand why, consider an IBM Thinkpad R40 laptop.

Table 2.1 shows the power consumption for different components of the device when idle and active [13,108]. With a 4.4Ah battery, the device when active will approximately last for two hours. Moreover, with the backlight off and different components idling, the device can last for a maximum of four hours. Such a short lifetime requires regular recharging, which can be an irritating proposition for users and infeasible for nomadic deployments [82,147]. Having larger batteries or solar panels is simply not an option since it adversely affects the portability of the device—making it less mobile.

Node mobility further complicates the energy management problem. To understand why, consider a solar powered client in a highly partitioned mobile network. Its primary source of energy drain is communication over the data transfer radio and searching for other nodes [52,83,84,147]. For such a node, schemes that duty cycle the radio without regard to node mobility may either be too conservative or too aggressive—the node might either lose transfer opportunities or may have a short lifetime. Therefore, *balancing* energy with *availability* in mobile systems is non-trivial and challenging.

Why is the state-of-the-art not sufficient?

The most common approach to energy management on mobile and sensor systems is to leave the device "on" but opportunistically switch off hardware components. These include aggressively spinning down the hard drive [50,51,72], turning off banks of RAM [75,91,103,114], using power saving modes on wireless cards [12,13,87,132], and CPU voltage and frequency scaling [58,67,101,126,161]. A multitude of algorithms and operating system primitives [59,106] have been proposed for tuning this duty-cycling to application and user needs [24,31,100,102,110,115,118,166,175]. Although switching off or duty-cycling components work well for a micro-controller class device [163], it is unlikely that such shallow power saving modes can provide order-of-magnitude longevity in lifetime for microserver class devices, PDAs, or laptops. The caveat is well explained using Table 2.1. Note that the idle energy consumption of several components of the system such as the CPU is very high, hence just idling system components can only produce a feeble reduction in energy consumption.

To circumvent the idle energy overhead of a PDA or a laptop class device, combinations of heterogeneous hardware platforms and radios have been proposed [29, 119, 125, 137, 151]. For example, the Wake-on-Wireless [138] system proposes augmenting a PDA with a sensor where an in-network server notifies the low power sensor as to when it should wake up the PDA. m-Platform [104], PASTA [137], and LEAP [48] are examples of tightly integrated multi-processor systems which combine a high power processor with a low power sensor device to expose a wide spectrum of low power modes. Analyzing these wide collection of systems, we find that all of them suffer from the following two drawbacks: (1) Most of these systems are hardware implementations, which lack a software architecture for optimal task sharing to exploit the augmented low power modes. A hand-crafted fine-tuned software design is important especially in a mobile network where the degree of uncertainty in connectivity or energy harvesting is huge. (2) Many of these systems require in-network support for wakeup and are not completely autonomous. Building autonomous systems is important especially in highly partitioned mobile systems where network support is not guaranteed. In this dissertation (see Chapter 4), we propose, implement, and evaluate a radically different approach called Hierarchical Power Management (HPM) which combines an intelligent software and hardware design to provide high levels of consistency at minimal energy consumption. We show that HPM is (1) *general:* can be applied to a wide spectrum of systems and applications (2) and *balanced:* it can balance energy consumption with availability and application requirements.

2.3 Performance Improvement in Mobile Networks

Energy is a first class primitive in the design of mobile and sensor systems. However, energy alone cannot dictate the performance of a mobile system. *What other factors affect the performance of mobile systems?* The answer lies in understanding why mobile systems are so popular. Quite simply they empower clients to remain connected (sometimes sporadically) irrespective of their locations. For example, a PDA user can access her email, sync her calender, or even make a voice call when she is mobile—something tethered systems like desktops cannot provide. Two important factors make this possible: mobile device portability and pervasive connectivity. With miniaturization of devices, portability is no longer an issue. However, connectivity at mobile clients is still far from ideal.

The definition of "optimal connectivity" is highly dependent on the mobile network and service under consideration. For example, if a client device such as a laptop or a PDA is used in a vehicular context for instant messaging or voice, almost always available access using open WiFi APs is required [19]. However, if we consider a Zebra tracking network [82], good connectivity would imply longer pairwise meeting times between Zebras and Zebras and base stations. In either scenario, mobility, wireless characteristics, and flaky network support make connectivity uncertain. For example, in WiFi access networks [63, 136] RF holes produce disruptions even in deployments tuned for blanket coverage [11, 27, 28, 38, 107, 133]. Traditional transport protocols such as TCP perform poorly in the presence of these disruptions, strangling performance [15]. Similarly, in highly partitioned DTNs, packet delivery delays can be of the order of days [17, 33] and throughput could be dismal due to scanty meetings among nodes.

How can connectivity in sparse mobile networks be improved? Several papers have proposed altering node mobility to enhance network performance [34, 36, 173]. Unfortunately, movement patterns are often inherent to the mobile nodes and cannot be modified. Additionally, intelligent DTN routing protocols has also been proposed to improve performance [17, 33, 34, 37, 111, 135, 139, 158]. While routing schemes can determine the best path from a source to a destination, they cannot add capacity to the network—a primary cause of poor performance in highly partitioned mobile networks. Further, use of mobile ferries [171, 172], mobile robots [35], and stationary relays [97] have been proposed to increase network capacity. However, most of this work are based on computer simulations with little-to-no real world validation.

Similar to sparse mobile networks, client and infrastructure modifications have been proposed to make mobile WiFi access more predictable. Client side modifications include tweaking network transport for wireless [15, 69], using multi AP virtualization to improve coverage [42, 86, 117], using disruption-tolerant techniques [18, 55], faster handoffs [140], and better access point selection [116]. Schemes which advocate WiFi infrastructure modifications such as multi-AP diversity [19], infostations [88] and drive through Internet [121] have also been proposed. Most of these techniques improve the quality of connectivity or transport protocol performance where there *is* WiFi coverage.

In spite of this varied collection of related literature, there still exists a paucity of practical schemes that can (1) improve capacity in sparse multi-hop networks and (2) patch disruptions where is *no* WiFi coverage in one-hop networks. In this dissertation, we argue (through analysis and deployment) that stationary infrastructure in the form of relays [79], mesh [32], and wired base stations [77] are the most practical enhancements for one-hop and multi-hop mobile networks.

Why is deploying and managing infrastructure difficult?

Without building and deploying infrastructure it is impossible to understand the practical real world issues involved. This is especially important in mobile networks where the deployment environment may be urban/semi-urban [149] or a remote location [82]. To appreciate the difference between a virtual world simulation and practical deployment, consider the following example. Autonomous robots [36] and ferries [171] have been proposed to enhance mobile networks. However, deploying moving nodes in a city can raise administrative alarms, making them impractical. While deploying stationary infrastructure in the form of relays or a mesh seems more manageable, it raises other practical concerns. First, since these nodes may have to be placed at arbitrary locations, tethered connectivity cannot be guaranteed. Therefore, these additional infrastructure nodes should be designed to last perpetually on energy scavenged from renewable energy sources such as solar. Second, while simulations can place nodes at locations that are outputs of optimal placement algorithms [174], practical obstructions such as buildings and trees constrain node placements to sub-optimal locations—making performance of a simulation different from a real world experiment. Third, long lasting remote deployments face technical challenges such as remote debugging especially if the systems are built on inherently unreliable hardware [96, 129, 130]—making infrastructure management a overly tedious and painful proposition. This dissertation addresses these unique challenges through the design, implementation, and deployment of Throwboxes (see Chapter 6), a relay architecture for enhancing sparse mobile networks and Epsilon, a mesh architecture for enhancing WiFi based mobile networks (see Chapter 7).

2.4 Discussion

This chapter places the problems addressed in this dissertation in the wider context of related work. It argues why energy management in mobile and sensor systems and enhancing mobile networks with infrastructure are still virtually unsolved problems. In the next chapter, we introduce our mobile testbed, DOME, which is the primary platform used in the evaluation of our systems.

CHAPTER 3
EVALUATION PLATFORM: UMASS DOME

This dissertation uses a diverse research methods to validate several research hypotheses. We have used rigorous mathematical analysis to derive performance bounds of our algorithms and analytical modeling to compare the performance of alternative infrastructure enhancements to mobile networks. However, a part of this thesis uses prototype implementation and deployment to study the performance of our systems. Most of our deployments were made using our diverse heterogeneous mobile testbed—DOME (Diverse Outdoor Mobile Environment) [149]. Here, we briefly describe the salient components of our testbed which are important for understanding the evaluation framework in the rest of the thesis.

3.1 Diverse Mobile Outdoor Environment

The DOME testbed has been operational since 2004 and provides infrastructure for a wide range of mobile computing research. It includes 40 transit buses equipped with computers and a variety of wireless radios, 26 stationary WiFi mesh access points, and thousands of organic access points. It provides support for diverse radio technologies, including WiFi, 900MHz radios, 3G, and GPRS. It covers an area of 150 square miles and provides spatial diversity; parts of the network form a sparse, disruption-tolerant network while others are denser. With proper isolation the testbed can be used for research ranging from infrastructure-based networking to sparse and dense ad hoc networks. To place the system evaluation in this dissertation into perspective, we first describe the principal hardware and software components of DOME.

Hardware Components

The DOME testbed consists of a DieselNet vehicular network [33], an outdoor mesh network, and thousands of organic access points. Below we underline each of these components in detail.

DeiselNet

Mobility in DOME is provided by a vehicular network called DieselNet [33]. DieselNet is comprised of 40 transit buses, each equipped with Hacom OpenBrick 1GHz Intel Celeron M systems with 1GB of memory running Ubuntu Linux, 60GB 2.5 IDE inch hard disk, 2GB Compact Flash disk, Deluo USB GPS receiver based on the SiRF Star III chipset, Compex WLM54AGP23 802.11abg mini PCI cards using the Atheros AR5413 chipset (upgraded from 802.11b Prism2-based USB WiFi dongles), 802.11g wireless access point used as a bridge to an Ethernet port on the OpenBrick, SierraWireless 881 3G USB Modems operating on the AT&T network, Digi XTend 900MHz USB RF modem, and an inverter to convert 24VDC to 120VAC.

The node's access point allows other buses, or bus riders, to establish 802.11 connections into the brick, giving them access to the Internet via the 3G modem. The WiFi interface is used by a brick to connect to foreign access points, including the APs on other buses. The SSID broadcast by a brick's AP allows it to be identified as belonging to a DieselNet bus by other buses and any other DOME device.

Mesh Network

In cooperation with the Town of Amherst, we have installed 26 Cisco 1500-series WiFi access points. These are lightweight access points, managed by a central controller, and they support seamless hand-off for mobile nodes. The APs use two radios: a 802.11g radio for the public and mobile nodes to connect to, and an 802.11a radio to mesh APs together.

The nodes are mounted on a variety of town-owned buildings and light poles. While both locations provide power, only the buildings provide connectivity to the local fiber infrastructure. Consistent with research findings [40] and Ciscos direction, the network is laid out such that there are never more than three hops without connectivity to the wired network.

Software Components

There are several important software modules built into DOME. Here, we only describe the link management module which is most relevant to the evaluation of systems in this dissertation.

We have implemented our 802.11 discovery and link management policies on the vehicular nodes in a software module referred to as LiveIP. The purpose of LiveIP is to scan for SSIDs, to establish and maintain WiFi connections as defined by the policies set by the currently executing experiments, and to notify applications of the state of the WiFi link. Selecting an access point and deciding when to terminate a connection is an ongoing research topic [107, 116] and is a crucial factor in the performance of opportunistic systems. The LiveIP configuration allows us to define regular expressions for prioritized and blacklisted SSIDs, as well as the policy for dropping an association. We can tailor the policy to individual experiments, such as only connecting to public APs and not to APs on other buses, or vice-versa. By default, if no preferred SSIDs are found LiveIP uses signal strength to select an AP. We also maintain a weight based on any prior attempts to connect to an AP. However, this weight is currently only used as the final tiebreaker when choosing between APs with the same signal strength. We have adopted some of the same mechanisms described by the CarTel project [78], including DHCP caching, which reduces the amount of time required to associate with APs. Similarly, there is an application on the buses to manage the 3G link, and the buses have services to manage the XTend radio links and listen for XTend beacons.

Throughout this thesis, we have used results from several deployments. A subset of these deployments took place in the summer of 2005 and the Fall of 2007.

3.2 Discussion

In this chapter, we introduced our mobile testbed, DOME, which is the primary platform for evaluation of systems described in this dissertation. In the next chapter, we introduce our novel energy management architecture, Hierarchical Power Management (HPM), which provides high levels of consistency in mobile and sensor systems at minimal energy consumption.

CHAPTER 4

HIERARCHICAL POWER MANAGEMENT: PROVIDING NETWORK AVAILABILITY AT MINIMAL ENERGY

A primary deterrent to providing high levels of consistency and availability on mobile and sensor systems is limited energy. While recent years has seen the metamorphosis of mobile devices such as laptops and PDAs into general purpose compute platforms, battery capacity of these devices has always remained a primary bottleneck. The short battery lifetime of mobile devices such as laptops and sensor network nodes such as microservers has led to aggressive power management techniques, which extend device lifetime by reducing the amount of time the device is on. However, such an approach leads to energy savings at the cost of reduced functionality and may not be suitable for a wide range of applications. Therefore, designing highly available and long-lived untethered mobile and sensor systems poses the following unique challenge: *how can a device provide both performance guarantees, high levels of availability, and a long lifetime?* These goals are in direct conflict with each other: supporting performance guarantees requires examining requests immediately, providing such vigilance is energy intensive.

In this chapter we resolve this tension through a new architecture for energy management in mobile and sensor systems—*Hierarchical Power Management (HPM)*. HPM is predicated on the fundamental property that rather than choosing a single hardware platform, the needs of untethered mobile and sensor systems are best met by combining platforms with complimentary hardware characteristics. Using two instantiations of HPM (*Triage* and *Turducken*) from a diverse application space (laptops and sensor net microserver design), we show that systems built hierarchically can provide high levels of availability at minimal energy consumption.

4.1 A New Paradigm: Hierarchical Power Management

The amount of non-reducible (idle) power varies for different devices. For example, the non-reducible power of a PXA-based PDA-class device is on the order of twenty-times smaller than the non-reducible power of an x86-based laptop. As another example, the non-reducible power of a small sensor is significantly smaller than that of a device such as a wireless music player. Typically, devices are carefully optimized to provide their promised functionality at the lowest possible energy cost, and devices that provide less functionality have less non-reducible power. Fortunately, there is significant overlap in the functionality provided by high-power and low-power devices. For example, maintaining a consistent view of a file requires only the ability to connect to a network and download data; a variety of devices can provide this functionality, or serving simple queries in a sensor network requires computational and storage ability; something also provided by a spectrum of devices.

The goal of Hierarchical Power Management (HPM) is to reduce the energy cost of maintaining high levels of consistency on mobile and sensor devices by combining several optimized platforms into one integrated system. By combining a very low-power platform such as an Mote class sensor with a very high-power platform such as a laptop or a PDA, we can produce a system with reduced energy consumption and still have all the functionality of a laptop or a PDA.

A HPM system is composed of a set of *tiers*, each with a set of capabilities and a power mode. The system as a whole executes tasks (e.g., downloads data updates) by waking the tier that has the capabilities to execute the task in the most efficient manner. For example, one tier might include a StrongARM processor, along with its memory and storage. This tier could be integrated with a standard x86-based laptop. We can then suspend the x86-tier and rely upon the StrongARM-tier to wake up and perform periodic tasks.

For instance, if the StrongARM-tier wakes up periodically to cache a copy of frequently-used web pages, when the user opens the laptop, those pages will be available and highly consistent. If the laptop alone were to frequently wake itself up and cache those same pages, it would attain the same level of consistency; however, the overall *lifetime* of the system would be greatly diminished. As another example, consider a sensor net microserver which serves image queries to clients. If the microserver is composed of a low power Mote class device and a high power PDA class device, the Mote class device can provide always-on availability at minimal energy consumption—accept queries, and even process them if it has the resources. When required, it wakes up the PDA device and uses its CPU and storage to process more complex queries. For a PDA-alone system, providing such levels of availability would require an order-of-magnitude more energy!

Note that in this integrated system all of the tiers use a common battery, are connected by a common bus, and effectively form a tightly coupled distributed system. However, from the user's/client's perspective it appears to be a single device. The addition of extra components does increase the weight and cost of system. For instance, adding a StrongARM mobile processor and memory to the inside of a laptop may add $100 and a few ounces. However, the extra capabilities the system provides outweigh these costs. Another observation is that this system could be commercially built using commodity components. The architecture is fully composable: any set of tiers can be used together to give a wide variety of power modes and can be applied to many mobile and sensor devices.

4.2 System Design

The design of a HPM system is composed of three parts: the hardware, the underlying system architecture, and the model for distributing applications across the tiers. In general, the design is similar to many distributed systems; each tier is under autonomous control while decisions are made in a distributed manner. Tasks that support client applications are distributed among the various tiers.

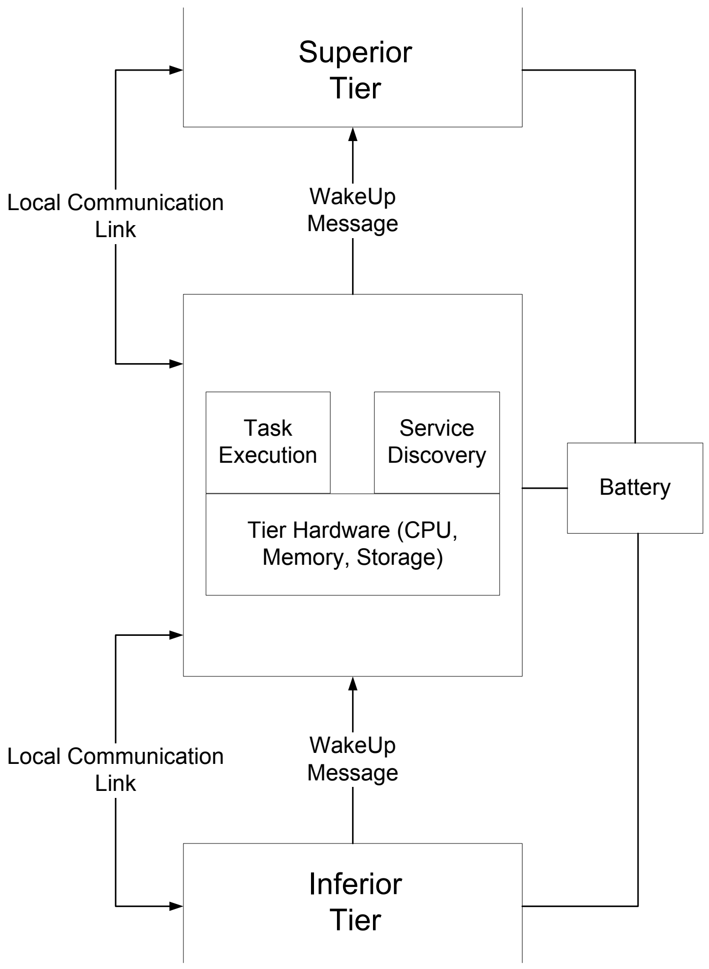


Figure 4.1. HPM System Design

4.2.1 Hardware Design

A HPM system is designed in a strictly hierarchical manner, and each tier is more powerful than any tier below it. Each tier can communicate with a *superior* tier and an *inferior* tier—the two exceptions being the top and bottom tiers of the hierarchy. Communication occurs via a local communication network and the tiers are connected to a common power source. Moreover, each tier has the ability to draw its superior tier out of a suspended mode. It is fully composable; the system will still operate correctly if tiers are added, removed, or changed. This provides a flexible architecture that can accommodate the evolving number of hardware platforms available in low-power computing. For example, later in this chapter we discuss how we have applied the architecture to variety of hardware platforms, including Motes, x86-based laptop, and PDA-class devices. An overview diagram of our design is shown in Figure 4.1.

Each tier contains an independent processor, memory, internal bus, and persistent storage system. Each may also have an independent external wireless network interface, although these can be shared by routing through the inter-tier communication network. The set of tiers can be architecturally homogeneous and span a range of power requirements. By limiting the interface between tiers, we achieve composability. Integrating new tiers with differing instruction sets, capabilities, operating systems, and power requirements is straightforward.

A HPM system is also fully autonomous and does not depend on any special hardware from the external network. For instance, HPM does not require external networks to be equipped with hardware wakeup signals, such as those used in the Wake-On-Wireless project [138].

4.2.2 System Architecture

The system as a whole is responsible for accepting tasks from the user/client and executing them in a way that extends the lifetime of the system. Tasks can be anything from keeping time synchronized to ensuring that the local copy of the user's email is current to queries in a sensor network. The user/client, or a service executing on behalf of the user/client, introduces tasks at a tier and the system distributes these tasks among the different tiers in a way that extends the lifetime of the overall system. For example, if the task is a query in a sensor network, it could appear at the lowest-power tier (a Mote) and if it is sending an email on a laptop it could appear at the highest power tier (the x86 platform). Each tier is capable of several operations: perform tasks or discover services; inform other tiers when necessary; and manage its local consumption of power. We discuss each responsibility in more detail below.

Perform a task. A tier can perform a task if the required service is reachable and ready to be used. Ideally, a task will be executed by the most efficient tier capable of performing that task. For example, the highest-power tier would be required to synchronize a very large media file while a PDA-tier can perform the task of synchronizing a cache of web pages. For some applications, a tier will also need to pass the results of task execution to its superior tier. For example, a web page cached by a StrongARM-tier will ultimately be delivered to the highest tier in response to a user request.

Perform service discovery. A tier can also monitor the availability of a service required by a higher-power tier in order to perform a task. Service discovery may simply discover the existence of a service, or it may determine whether a particular service needs to be used (e.g., whether a user has new email that needs to be fetched). Again, service discovery should be performed by the lowest-power tier is capable of discovering the service. In many cases, it is also possible to further decompose service discovery. For example, to determine if a large media file is available to be synchronized, an ATmega-tier can monitor the network for connectivity, a StrongARM-tier can determine if the file has changed, and the x86-tier can actually perform the task.

Enter a suspension state. If a tier is not needed to perform a task or service discovery, it may put itself to sleep in order to conserve energy. In some cases, this may require that the tier delegate tasks or service discovery jobs to its inferior tier. For example, a PDA-tier may notify a Mote-tier that it is going to sleep and needs to be woken when a network connection is available or there is a task for which it is required.

Wake its superior tier. Once a tier has discovered an appropriate service, it may need to wake its superior tier so that it can perform the task. Each tier is capable of waking its superior tier. In this way, a tier can rely on its inferior tier to tell it when there is something to do rather than requiring the system to wake periodically and check.

4.2.3 Distributing Applications

There are several methods of distributing application responsibilities over the tiers. We describe each of these options here:

System-Aware Architecture. The first option is to build an application that is customized for the system. Such an application requires designers to create application components for each tier. In addition, the application must define the messaging protocol used to communicate between components. This hand-coded option is useful for new applications and also for applications, such as time synchronization, which are fairly simple to implement. We show the efficacy of such a software architecture in the design of *Triage* later in the chapter.

Proxy-Based Architecture. A second option is to use a proxy-based architecture that can take advantage of existing distributed application components. Using this architecture, a tier that executes tasks appears as a proxy service provider or a replicated server to superior tiers. Many distributed applications, such as distributed file systems, email, and web caching, already support this design. Therefore, the advantage is simplicity and deployability—proxies only require recompiling and reconfiguring the application rather than recoding. Unfortunately, not all applications will tolerate a proxy that queues responses, requiring some modification of applications. One possibility is to use queued RPC as found in the Rover toolkit [81]. We use this architecture in the design of *Turducken*.

Transparent Architecture. A final option is to develop a HPM system component that is capable of transparently migrating application processes. One way to support this is by using traditional process migration [98, 128, 152, 155]. Another possibility is to use virtual machines, either programming-language virtual machines such as those used for Java, or a lightweight, OS-level virtual machine such as Denali [162].

Till now we have described the overall system architecture for a HPM device. To demonstrate the efficacy of the HPM architecture, we next present two instantiations of HPM from a diverse application space—(1) *Triage:* a HPM architecture for building sensornet microservers. Triage is capable of providing always-on availability for sensornet clusterheads at minimal energy consumption. (2) *Turducken:* a HPM architecture for building highly energy-efficient yet highly available laptops. In this chapter, we focus on the evaluation of Triage and present the prototype implementation of Turducken. For detailed evaluation of Turducken, please refer to the associated research publication [146].

4.3 Triage: A Sensor net Microserver Design

Microservers form an intermediate battery-powered tier between tethered base stations or access points, and resource constrained nodes. Microservers provide services including complex data processing [94] and high-capacity storage to augment storage limited nodes [16]. They respond to user queries in a low latency manner [94], provide greater range and coverage [89], and act as gateways between short range and longer-range radio networks [65]. Designing microservers face unique challenges. While microservers need to have the capability and availability of conventional servers; being untethered they must be highly energy efficient for long lifetime.

Triage is a HPM instantiation which provides Quality of Service (QoS) and energy-efficiency in a microserver through a combination of a high-power, resource-rich platform and a low-power, resource-constrained platform. The low-power tier, or *tier-0*, remains always-on ensuring responsiveness at minimal energy cost. The high-power tier, or *tier-1*, remains in a power saving mode until its resources are required for a given service. The low-power platform acts similar to a medical triage unit, examining requests to identify the critical ones, and as a scheduler to minimize the number of times the high-power platform is woken. Such a scheduling mechanism requires accurate *in-situ* profiling of the time and energy needs of each task, which is performed at the low-power platform. The scheduler can optimize for different criteria—in this chapter we focus on two: minimizing energy consumption while meeting soft-real-time latency constraints, and meeting a lifetime goal while satisfying as many task deadlines as possible.

4.3.1 Hardware Architecture

Triage employs a *tiered* hardware platform. The tier-0 platform is a very low-power platform, in this case a TelosB Mote [129], and tier-1 is a more capable and higher-power platform, in this case a Stargate [160]. In Triage, the two tiers are tightly coupled and directly communicate over a wired link. This enables the lower tier to trigger the wake-up of the higher tier when necessary.

The TelosB Mote is extremely resource-constrained, but consumes less than one-tenth the power of the Stargate. This platform works well for always-on operation, simple packet processing, and providing low-latency responses. The Stargate platform is significantly more capable but less responsive due to the high latency of sleep to active transitions. We augment this two-tier platform with a custom fabricated interface board that provides necessary voltage conversions for the two platforms, the wakeup interface, as well as a fuel-gauge chip to measure energy-consumption in-situ.

4.3.2 Software Architecture

Figure 4.2 illustrates the components of the software architecture for Triage. Tier-0 virtualizes resources available on tier-1 using a collection of *surrogates*. Surrogates receive requests from the network and can service them in one of three ways: by using locally cached information, by performing local execution, or by passing them to tier-1 for execution. The surrogates decide on how to service the request based on information provided by the profiler and the scheduler. A *profiler* measures the energy and processing requirements of a task and a *scheduler* determines, based on the predicted energy cost of a task, when and where it should be executed to meet QoS requirements. Requests that have been scheduled for execution at tier-1 are written into a *log* and delayed until their scheduled execution time. This log also serves as a *cache* of recent requests.

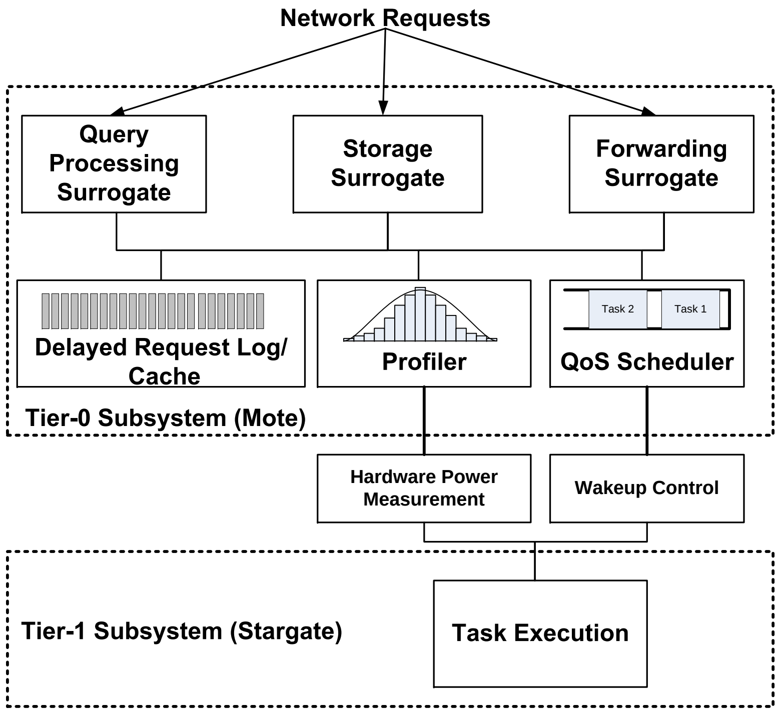


Figure 4.2. Microserver Software Architecture

4.3.2.1 Surrogates

Surrogates are small software modules that run on tier-0 and provide a service such as storage or forwarding. Though tier-0 can process simple tasks, such as routing updates or time synchronization, most tasks require resources only available at tier-1. When a request arrives at the microserver the surrogate performs the following: (1) immediately execute requests from information cached at tier-0; (2) if a request cannot be serviced from the cache, ask the scheduler to determine where and when to execute the task; (3) if the scheduler determines that the task should be executed at tier-0, execute it immediately; (4) otherwise write the request into the delayed request log. The delayed request log acts as a priority queue—the task with the smallest deadline lies at the head of the queue.

Triage also uses the delayed request log as a cache for the surrogates. This functionality is particularly useful in storage applications; a read closely following a write to the same data can be serviced from the log. In order to maximize the amount of cached data, Triage does not erase the tier-0 log when a batch of requests is played at tier-1. Instead, the previously committed log entries and cached results are lazily overwritten by new requests using an LRU eviction policy. Though writing into the log consumes energy, we argue that it is insignificant to the savings from minimizing the number of tier-1 wakeups.

To enable applications to compose the functionality of several surrogates, Triage also permits communication between surrogates using primitives provided by the operating system. For instance, a client may query the microserver for information, and request that the results of the query be sent to another node. This requires a combination of a storage surrogate as well as a forwarding surrogate.

4.3.2.2 Scheduler and Profiler

Triage uses a scheduler, running on tier-0, to provide QoS. The scheduler relies on a profiler to provide information regarding how long each type of task will take to process, and how much energy it will consume on both tiers. The profiler measures the execution time of each task and builds a model of task execution time. Using this information, the scheduler determines *where* and *when* to execute each request. The scheduler relies on the execution time profiles generated by the profiler. However, if a task exceeds the typical execution time, it is not preempted and is still executed to completion.

The question of where to execute a task can be answered by comparing the amount of energy required to execute it at each platform. This decision may be even simpler if the task requires the resources of tier-1 and cannot be executed on tier-0. The question of when to execute requests is more complicated. There are two cases when the scheduler must wake tier-1 and dispatch outstanding tasks. The first case occurs when the log becomes full. In this case, the scheduler is automatically invoked by the delayed request log; it wakes tier-1 and dispatches each outstanding task to the appropriate service. The second case us ub response to QoS constraints. Each request can optionally contain a soft-realtime deadline, or latency constraint, which indicates the time by which a task should be executed. In order to meet the deadlines with maximum energy efficiency, the scheduler will delay execution of tasks as long as possible to increase the amount of time tier-1 remains in a low-power state. However, if a batch is already being processed, all tasks irrespective of their deadlines are executed on tier-1. This is done to avoid the high transition cost of waking up tier-1. We describe the algorithms used by the scheduler and profiler in Section 4.5.

4.4 Example Surrogates

Some of the common and basic functions found in servers for sensor networking, mobile networking, and pervasive computing are storage, query processing and forwarding. To this end, we present three example surrogates: a storage system surrogate, a network forwarding surrogate, and a query processing surrogate. As untethered networks proliferate, this library of surrogates will be expanded, enhanced, and further optimized.

4.4.1 Storage System Surrogate

The storage surrogate enables in-network storage applications. It accepts read, write, and delete requests for the tier-1 storage system. Upon receiving a request from the network, it first determines whether the request is a read request that can be satisfied by a recent write cached in the delayed request log. If so, it immediately provides the result. Otherwise, it asks the scheduler to schedule the task. The scheduler considers any soft real time deadline provided with the task and tells the surrogate when the task has been scheduled. The surrogate then inserts the task into the delayed request log.

4.4.2 Network Forwarding Surrogate

The network forwarding surrogate enables efficient routing by utilizing both the tier-0 and tier-1 network interfaces. When a packet arrives at the surrogate, it examines the destination address, consults a routing table, and determines over which radios the destination is reachable. It immediately passes this information, along with any latency constraint, to the scheduler which determines which radio should be used to send the packet and when the packet should be sent. If the scheduler determines that the tier-0 radio should be used, the packet is sent immediately. Otherwise, the packet is inserted into the delayed request log.

4.4.3 Query Processing Surrogate

The query processing surrogate provides a simple query interface for data stored on the microserver. Clients may use simple queries, such as *retrieve all images from the last ten seconds*, or more complex queries, such as *retrieve all images that contain 2 or more objects and are from a particular geographic region*. The queries are specified by the following fields (1) the type of query (simple/complex) (2) the images queried (3) the number of objects desired per image for complex queries (4) latency deadline associated with each query. The query processing surrogate uses other surrogates to create a combination of services, including processing, routing, and storage. Tier-1 can execute any query since it has access to the powerful radio, the primary storage system, and a powerful processor. However, tier-0 can respond to only simple queries. For example, any query that can be performed using simple comparisons of cached metadata can be performed at the tier-0 system. Therefore, tier-0 maintains an index of results stored from previous queries in its cache/log.

When the surrogate receives a query, it first determines whether the query is a simple query that can be executed using data cached in the delayed request log. We assume that tasks are statically mapped into simple queries, which can be executed at either tier, and complex queries that require the resources of tier-1. If the query can be executed at tier-0, it is executed immediately. Otherwise, the surrogate passes the query and any latency constraint to the scheduler. Once the scheduler has scheduled the query, it is inserted into the delayed request log.

4.5 Profiling and Scheduling

At the heart of Triage is a profiling engine that is used to estimate the execution time and energy usage for different tasks, and a scheduling engine that determines how to meet QoS constraints. In this section, we describe the algorithms employed by the profiling engine and the scheduling engine to enable energy-efficient scheduling of tasks.

4.5.1 Task Profiling Algorithm

Triage employs a profiler to measure the execution time and energy usage for different tasks. In this discussion, we focus on determining the *typical energy usage* and *typical execution time* for each type of task. Such online profiling is necessary to deal with the variability in execution time and energy usage of tasks that involve a combination of processing, communication and storage.

Online profiling involves two steps, task grouping and parameter estimation. The online profiling engine first identifies a task as belonging to a certain group based on the nature of task. This grouping information is assumed to be provided a priori by the system designer. We believe that such a grouping is appropriate since many applications of microservers involve a small and well-specified set of tasks. For instance, in a camera sensor network, a typical set of tasks might be {`Motion Detection`, `Face Recognition`, `Store Image`, `Send Data`}.

For each of these task groups, the profiler uses a separate history of execution times and energy consumption to build corresponding probability distributions. We focus on the estimation of the typical execution time since a similar algorithm can be used to estimate typical energy usage.

Let $f(t_i)$ be the probability density of time taken to execute task type i. Further, let $\overline{X}_i$ and σ_i denote the average execution time and standard deviation for task type i. The profiler uses Chebyschev's inequality (shown in Equation 4.1) to determine an interval of time such that the task executes within that interval with probability at least p. Triage consequently takes the upper bound on the interval as the typical execution time for the task. This is a conservative estimate, however for tasks whose execute time distributions are not known apriori, a conservative strategy is required.

$$Time(i,p) = [\overline{X}_i - \frac{\sigma_i}{\sqrt{1-p}}, \overline{X}_i + \frac{\sigma_i}{\sqrt{1-p}}] \tag{4.1}$$

The parameter p can be tuned depending on the guarantee required by a user. For instance, in a camera network used for surveillance, a `Face Recognition` task might require a tight guarantee as the person might move out of the field of view of the camera sensors. In this case, p can be set to a high value, say 0.9. Other tasks such as `Send Data` might be more elastic, and `Store Image` might not have any deadline at all.

Using information collected after the task executes, the profiler builds two kinds of dynamic models: histograms and parametrized models. When prior models of execution time are unavailable, a simple histogram approximates the probability density, $f(t_i)$, and tracks bins of execution times and energy consumption for each task group. In contrast, when prior models are available, these can be used to more accurately model task execution time and energy consumption. For instance, the execution time and energy used in a communication task is a linear function of the number of bytes transmitted. In this case, the execution times are fit to a simple linear model to determine the costs of the two radios in Triage. Our prototype uses parametrized models for the storage and network surrogates tuned to the size of the file and the size of the packet respectively. More complicated models can be built for applications such as video coding, compression, and encryption. For the query processing surrogate we use the histogram-based profiling.

4.5.2 Task Scheduling Algorithm

The scheduler that resides on tier-0 uses the profiling information about tasks to minimize the number of times tier-1 is woken while still satisfying the task deadlines. The Triage scheduler uses different algorithms depending on the optimization criteria. In this work, we discuss two schedulers — the first is optimized to satisfy task deadlines, and the second is optimized to achieve a target lifetime for the microserver. While we limit our discussion to these two schedulers, we note that alternate schedulers that optimize, or balance, other QoS constraints can be plugged into our system.

4.5.2.1 Scheduling for Deadline Constraints

The deadline scheduler tries to minimize the energy consumed by the microserver, such that the deadlines of the incoming tasks are met. Let the set of tasks which are already batched at tier-0 for delayed execution at tier-1 be denoted by $S = \{T_1, ..., T_k\}$ where task T_i has deadline $D(T_i)$, latest start time $L(T_i)$, and execution time $E(T_i)$. The latest start time is the latest time at which a task can begin executing on tier-1 such that the deadlines of all tasks after and including itself are met, and the execution time is the time it takes to execute the task on tier-1. $L(T_i)$ and $E(T_i)$ are provided by the profiler. Further, we assume that the list S is sorted by deadlines *i.e.* $D(T_i) > D(T_j)$ if $i > j$. Let the wakeup time for tier-1 be W, and the current batch time, B, correspond to the latest time at which tier-1 needs to be awakened so that the deadlines of all tasks in the list S can be satisfied.

Our scheduling framework is based on the well-known As Late As Possible (ALAP) scheduler. When a new task arrives, the scheduler first queries the profiler for the typical execution time for the task at tier-1. Next, the algorithm recomputes the batch time, B, *i.e.* the latest time at which tier-1 can be woken such that all the batched tasks and the new task meet their deadlines. Let the new task be inserted at index l into the sorted list S based on its deadline. The scheduler now needs to ensure that the insertion of the new task does not result in missed deadlines for any of the other tasks in the list. The scheduler only lowers the batch time and never increases it, hence only the tasks that are before T_l in the list need to be checked for deadline violation. Thus, for each task $T_i : l \geq i \geq 1$, the scheduler sets the latest start time such that it does not violate the deadline constraint of T_i or any task with

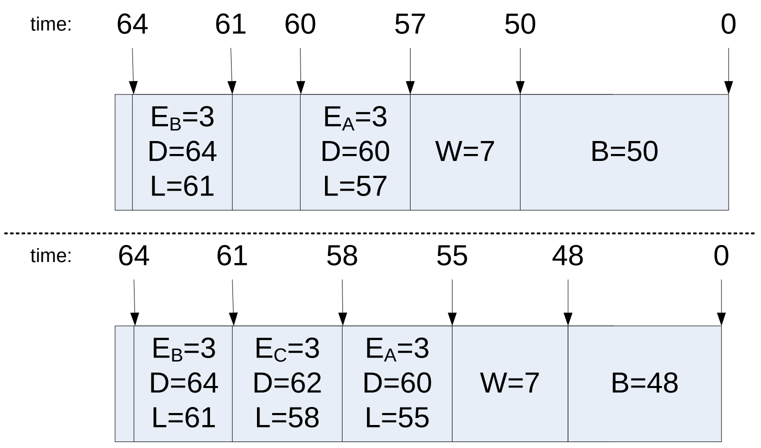


Figure 4.3. ALAP Example: The figure shows the execution time and deadline for each task, the wake up latency for tier-1, and the resulting batching time. A new task T_C is inserted into the scheduling decreasing the batch time.

deadline after T_i, *i.e.* $L(T_i) = min(L(T_i), L(T_{i+1}) - E(T_i))$. The new batch time, B, is updated to reflect the latest start time for the first task in the list, *i.e.* $B = L(T_1) - W$. If $B \leq 0$, tier-1 is immediately awakened and the batch executed. If $B > 0$, a timer will fire at time B and tier-1 will be awakened. The time required to update the schedule is linear in the number of tasks currently in the queue.

We illustrate the deadline scheduling algorithm with a simple example, shown in Figure 4.3. Let there be two batched tasks, T_A with deadline 60 seconds and execution time 3 seconds, and T_B with deadline 64 seconds and execution time 3 seconds. The latest start times of the two tasks are $L(T_A) = 57$ seconds and $L(T_B) = 61$ seconds respectively, and the batch time, B is 50 seconds, assuming a tier-1 wakeup time, W of 7 seconds. Now, a new task T_C arrives with deadline 62 seconds and execution time 3 seconds. The task is inserted between T_A and T_B. The scheduler checks whether the current batch time satisfies T_C's deadline, notices a violation, and pushes T_A forward in the schedule. Hence, the batch time is set to 48 seconds.

4.5.2.2 Scheduling for a Lifetime Constraint

While the goal of the deadline scheduler is to miss only a small percentage of deadlines while minimizing energy usage, the objective of the lifetime scheduler is to meet a target lifetime for the microserver while satisfying as many deadlines as possible. The scheduler should also be capable of handling periods of burstiness. To accomplish this we use a token-bucket algorithm for the lifetime scheduler. Given a target lifetime, L, and battery capacity, E, energy tokens are generated at a constant rate of $\frac{E}{L}$. The total number of accumulated tokens represents the amount of energy that is available for use by the system. The amount of energy used by the system is continually monitored by the energy profiler, and is queried periodically by the scheduler to determine the rate at which energy is depleted by the system. The difference between the accumulated energy tokens and depleted energy tokens at any time represents the surplus of energy that can be used by the system to execute the batched tasks.

The lifetime scheduler builds on the deadline scheduling algorithm that we described in Section 4.5.2.1. When a new task arrives, the deadline scheduling algorithm is used to queue the task and determine the batch time. When this batch time becomes zero, the lifetime scheduler checks to see whether there are sufficient energy tokens to wakeup tier-1, execute all the tasks in the batch, and shutdown tier-1. If so, tier-1 is awakened and the tasks are executed before shutting it down. The energy profiler is queried to determine how much energy was used during this batched processing, and the number of available energy tokens is updated accordingly. If the number of energy tokens is insufficient to execute the batch, the wakeup of the tier-1 platform is delayed until sufficient tokens have accumulated. During this period of waiting for tokens to accumulate, task deadlines could be missed, and tasks could be dropped if the size of the task queue exceeds the storage capacity of the tier-0 platform.

Neither of our scheduling algorithms take into account the availability of DVFS (Dynamic Voltage and Frequency Scaling) states at tier-1. Evaluating and profiling multiple

DVFS states would possibly permit greater efficiency by allowing tier-1 to sleep longer, however this pushes the complexity of the scheduler to $O(k^n)$ for n tasks and k power modes — the scheduling problem can be shown to be NP-Hard. We are currently investigating approximate heuristics that are computationally feasible for the tier-0 node. These heuristics can be implemented on top of the scheduling algorithms described in this section to determine when tier-1 should be awakened.

4.5.2.3 Idle State Management

One remaining issue is the state in which the scheduler leaves tier-1 while it batches new requests at tier-0—there are two options, suspension and shutdown. Suspension requires more idle power than shutting down the platform, but the transition cost is lower. For example, the Stargate platform with a CF 802.11 card draws 60.46 mW in suspend mode and costs 3.67 J and 32 J of energy to wake up from suspend and shutdown respectively.

The Triage scheduler determines the appropriate idle state for tier-1 based on the expected times between wakeups of tier-1. The expected time between wakeups are calculated over the last k wakeups, where k is a constant. We estimate the cost of each state based on the expected idle time, and choose the state that minimizes this cost.

While this approach is sufficient for many applications, the choice of idle state does enforce a minimum latency that the microserver can support. For example, if tier-1 is shutdown due to infrequently arriving tasks, the next task that requires tier-1 will have to wait at least as much time as tier-1 requires to wake from shutdown. In the case of the Stargate, this minimum latency is 15.6 seconds. In order to deal with this problem, more sophisticated profiling techniques could be used to anticipate the latency requirements of upcoming tasks and choose the proper idle state accordingly.

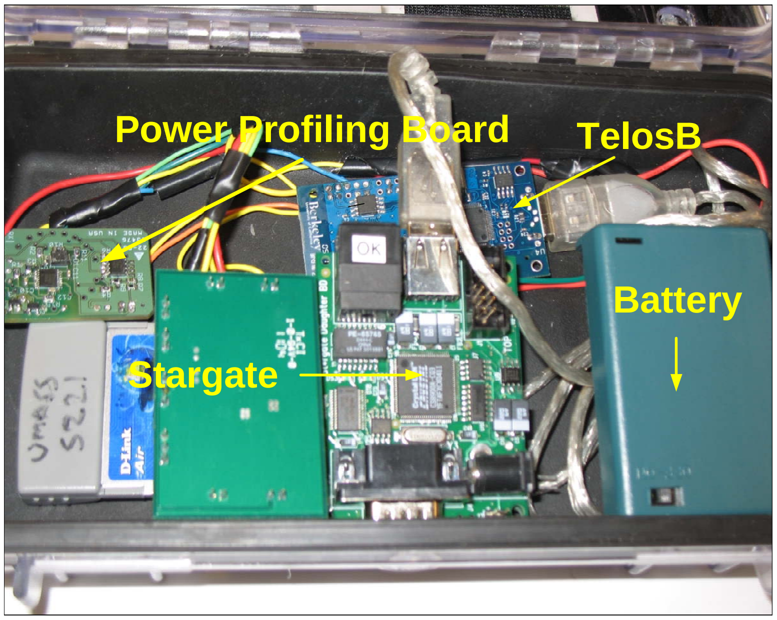


Figure 4.4. Prototype Triage System

4.6 Implementation

In order to evaluate our approach we have implemented a working prototype of Triage, shown in Figure 4.4.

4.6.1 Prototype Hardware

We constructed a prototype using a Crossbow Stargate (tier-1) [160] and a TelosB Mote (tier-0) [129]. The Stargate contains a 32-bit, 400MHz PXA255 XScale processor, 64 MB of RAM, 32 MB of internal flash, and a WiFi interface. The TelosB Mote contains an 8-bit, 8 MHz microcontroller, 10kB of RAM, 1 MB of external flash, and an 802.15.4 radio. These hardware platforms were chosen because they handle the range of workloads that we have targeted, are separated in power consumption by more than an order-of-magnitude (300mW-3000mW and 20mW-100mW), are easily programmable, and are well supported. The Stargate platform runs Linux, making available a broad range of software tools and services. We used the power-gating technique with a relay to switch off power to the CF card when the Stargate is suspended [125]. This technique brought down the suspension power of the Stargate to a modest 61 mW.

We used a power supply board designed to allow us to attach both boards to a single battery. This board provides an additional hardware element necessary to Triage, a Maxim DS2770 fuel gauge chip. The fuel gauge chip gives accurate readings of the energy left in the battery, and the amount of energy used by the system during task execution. As described, the scheduler uses this profiling information to control the platform energy policy.

One limitation of our current implementation is the transfer speed from the TelosB Mote to the Stargate. The two devices communicate over a USB line which is limited to 230 kbps. However, data needs to be read from the flash and then transferred over the USB which increases the total time required for the transfer. As a result, transferring 1024 kB of batched work along with protocol overhead takes more than 150 seconds and wastes a great deal of energy while blocked on serial I/O. We are currently investigating more efficient means of communication. Even with this limitation the current prototype shows extremely high gains in energy efficiency.

4.6.2 Surrogates and Log

As part of this prototype, we implemented three surrogates: storage, network forwarding, and query processing. The delayed request log is managed on the TelosB's flash storage using a custom designed file system. We implemented these components as TinyOS modules written in nesC [64]. The forwarding, storage, and query processing surrogates comprise 535, 480, and 540 lines of nesC code respectively, and the custom file system consists of roughly 1600 lines of code. The profiler and scheduler consist of 1360 lines of code. We also implemented an execution engine that runs on the Stargate and executes tasks when they are received from the Mote.

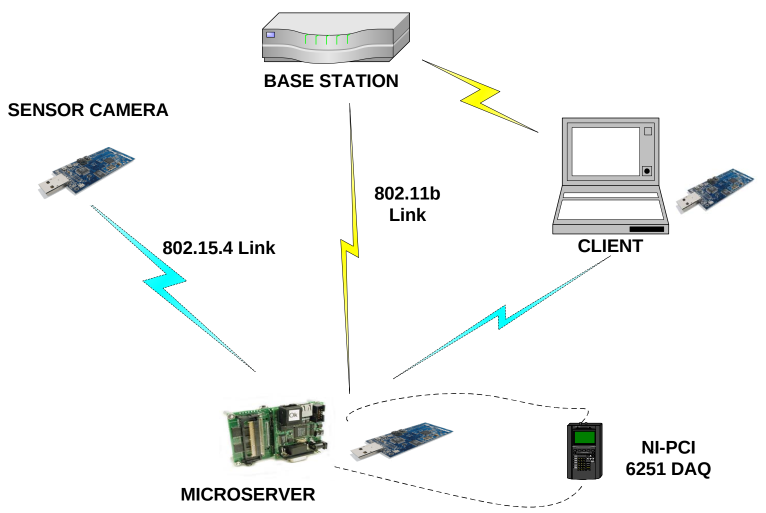


Figure 4.5. Experimental Setup

4.7 Evaluation

We evaluate the performance of Triage through an extensive set of experiments. First, we provide micro benchmarks that validate our use of a two-tier platform to achieve a combination of capability, energy-efficiency and responsiveness. Second, we evaluate the deadline and lifetime schedulers, and demonstrate their effectiveness in achieving their goals. Third, we focus on the performance of the profiler, and its accuracy when used with the forwarding surrogate. Fourth, we demonstrate how the Triage system is able to adapt itself to different QoS constraints at minimal energy consumption. Finally, we show the energy consumption of different independent components of the system and identify potential bottlenecks in the system.

TelosB Max. Power Consumption	100 mW
Stargate Bootup Energy	32 J
Stargate Bootup Time	15.6 s
Stargate Resume Energy	3.7 J
Stargate Resume Time	2.6 s
Stargate Suspension Power	60.5 mW
Stargate Idle Power (WiFi in PSM-idle mode)	912.3 mW

Table 4.1. Platform Measurements

We use a camera sensor network application in our evaluation. The application involves in-network processing, storage and forwarding and forms an adequate testbed for Triage. The experimental setup is shown in Figure 4.5. Motes emulate cameras in the network and feed images of variable sizes to the microserver. The other nodes in the network are client Motes attached to a device equipped with a 802.11b interface. Therefore, results of queries can be routed back to a client using a 802.15.4 or a 802.11b link. All power measurements were taken using a NI-PCI 6251 DAQ with a SC-2345 signal conditioning unit.

In our evaluation, we compare Triage with three other systems.

1. PSM-DVFS : This is a single-tiered dual radio system which uses WiFi PSM and DVFS (dynamic voltage frequency scaling) to save power on the Stargate. The system runs a DVFS algorithm which uses previously measured data to identify the lowest DVFS state on the Stargate where the given deadline can be satisfied. The wakeup interval in PSM mode for the 802.11 card is set to the maximum value supported by the card (33 secs). This provides an accurate comparison with a system which does not use the hardware architecture of Triage.

2. WoW* : The system is similar to Wake-on-Wireless [138]. The published Wake-on-Wireless system wakes up when it receives a network packet. WoW*, however, wakes up tier-1 whenever a task arrives at tier-0. Tasks are always executed on tier-1. WoW* is a system which uses the hardware architecture of Triage but does not use its software architecture.

3. Triage-Batch : The Triage-Batch system uses the same hardware as Triage. However, it does not use any online profiling, scheduling or caching. It batches a task as long as its deadline permits and then wakes up tier-1 to execute the task.

4.7.1 Static Energy Costs

In order to provide better intuition into the behavior of our prototype, we measure the static energy costs that directly impact Triage's performance. These values, shown in

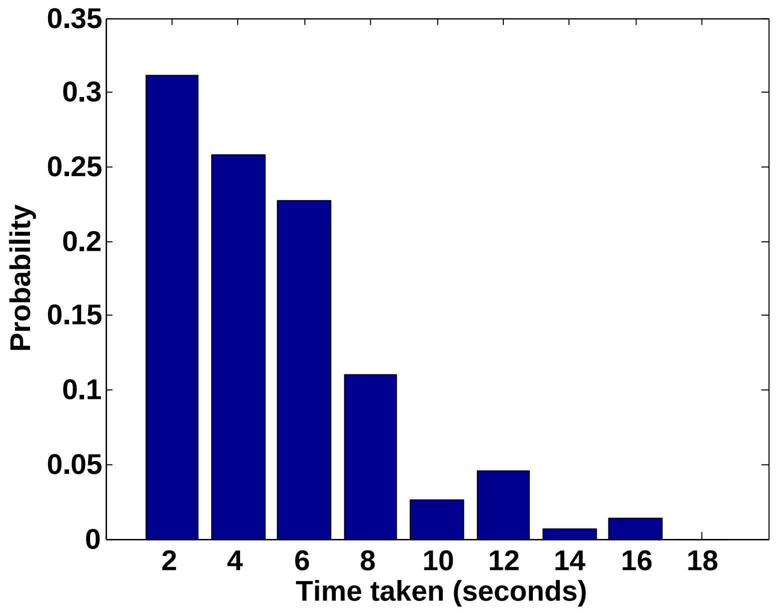


Figure 4.6. Histogram of the time taken to execute image classifier on Stargate

Figure 4.1, are the basis for the energy savings achieved by Triage. We observe that the idle cost of the Stargate platform with WiFi card in PSM mode is 6 times the sum of the suspend power of the Stargate and the maximum power consumed by the TelosB. Therefore, offloading tasks to tier-0 while keeping tier-1 asleep can lead to substantial energy savings. However, the Stargate's transition from suspend to active costs as much energy as 36 seconds of computation on tier-0. Transitioning from shutdown to active is equivalent to 319 seconds of active tier-0 computation. Therefore, replacing expensive state transitions at tier-1 with low-power, tier-0 computation as long as possible can lead to minimal energy costs for the system.

4.7.2 Soft-Realtime Scheduling

In order to evaluate the deadline scheduling algorithm, we observe how a Triage microserver performs in the presence of different latency constraints. In this experiment the microserver answers queries of client Motes for objects in images. Such a query is common in surveillance applications where a user might want to detect movements at a scene or extract important features of a scene. The objects in the image are computed using the `kmeans` classifier at the microserver and the classified image is sent back to the client.

The histogram of the amount of time taken to classify random 100-by-100 images using the `kmeans` algorithm is shown in Figure 4.6. The figure illustrates the large variance in the amount of time taken to classify an image. Moreover, it is clear that the time taken to classify an image does not follow a known probability distribution. Hence, accurate time profiling for the application is crucial to the success of the scheduler in meeting the deadline constraint at minimal energy consumption. The profiler uses a threshold of $p = 0.9$ in Equation 4.1 to determine the typical execution time of each task, i.e., at least 90% of the time the microserver will meet its deadline for tasks.

We evaluate the scheduler on two task arrival distributions—(i) when tasks arrive at a constant rate of one per 30 seconds (ii) when tasks arrive in bursts of three queries—the inter arrival time between bursts follow a Poisson distribution with mean 30 seconds. While the first scenario exhibits a constant load on the system which is easy to learn, the second task arrival distribution tests loads which are unpredictable. Therefore, the system has to accurately predict and adapt to the variable load patterns to perform well.

We varied the latency constraint on the task in the experiment and we measure the power consumed by the microserver and the percentage of tasks completed within the deadline. Each point on the x-axis represents experiments where the latency constraint is chosen uniformly at random within the interval $[x - 30, x + 30]$ seconds. 100-by-100 images are sent to the microserver at a rate of one per minute. The results are compared with the Triage-Batch system.

The results of the experiment are shown in Figure 4.7 and Figure 4.8. We first note (in Figure 4.8) that for all workloads the Triage deadline scheduler meets at least 90% of the latency constraints. Triage adapts to bursty and unpredictable task arrival distributions. This demonstrates both the accuracy of the non-parametric histogram profiler and deadline scheduler of Triage. The histogram-based profiler precisely determines the computational

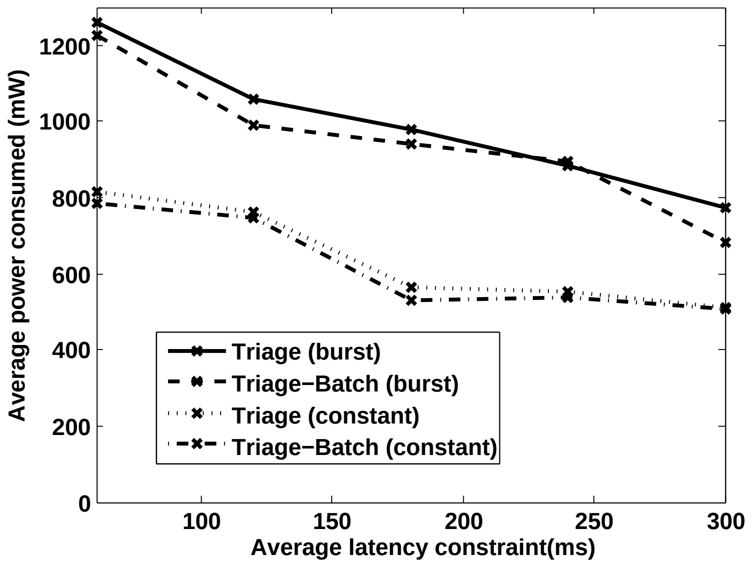


Figure 4.7. Average Power Consumption for the deadline scheduler. The deadline scheduler consumes slightly more power in order to meet more deadlines.

needs of queries. The deadline scheduler correctly schedules the wakeup of tier-1 to meet the desired latency constraints. Without profiling the execution time of tasks or the deadline scheduler, the Triage-Batch scheduler is unable to determine when to wake tier-1 and regularly misses deadlines, especially when latency constraints are small. This is because scheduling errors are more likely to occur at the beginning of each batch of tasks. Allowing longer latencies results in larger batches of tasks and more tasks that are processed ahead of their deadlines. Moreover, the Triage system consumes only slightly more power than the Triage-Batch system.

The second observation we make from the experiment is that the Triage system satisfies all constraints irrespective of the task arrival distribution. However, both systems consume more power for the bursty task arrival distribution. Even though the mean inter-arrival time between bursts is the same as the interval between arrival time of constant rate tasks, the number of tasks executed for bursty task arrival is more than for the constant task arrival distribution.

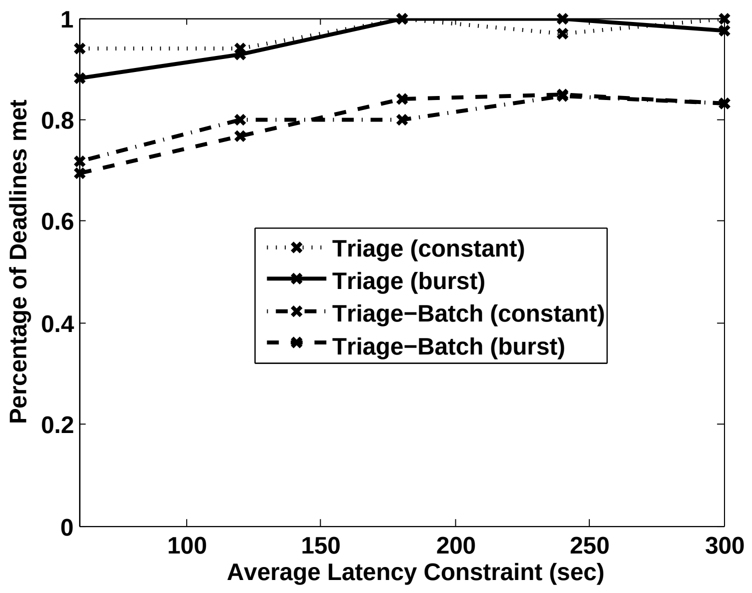


Figure 4.8. Percentage of Queries executed within deadline. Triage meets more than 95% of all deadlines while Triage-Batch is able to meet only 70% of all deadlines

4.7.3 Lifetime Scheduling

To show that the lifetime scheduler is able to meet a lifetime goal, we perform a similar experiment as before using a Query Processing application. We use simple queries for images stored by the Stargate storage. In order to expedite this experiment, we use a small battery capacity of 100 mAhr—enough energy for the microserver to operate at a maximum load for 9 minutes. We set the lifetime goal of the server to be 60 minutes. For the first 30 minutes, all queries arrive at a constant rate of one per 30 seconds with a latency constraint uniformly distributed over the interval $[150, 210]$ seconds. For the remaining time, the server sees a more intense load with queries arriving with latency constraints uniformly distributed over the interval $[5, 15]$ seconds. Figure 4.9 shows the results of this experiment. The slope of the straight line demonstrates the lifetime goal divided by the energy capacity of the server—this is the overall average power goal.

Recall that when using the lifetime scheduler, Triage prioritizes lifetime over latency constraints. It attempts to meet the latency constraint whenever it can, and the token-bucket algorithm allows for bursts of short, energy intensive workloads. For this algorithm to

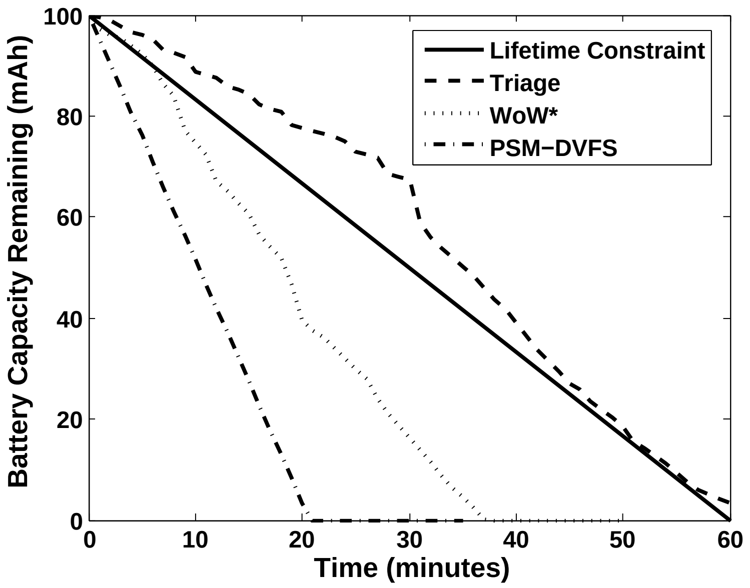


Figure 4.9. Lifetime Scheduler. This figure shows Triage's use of the lifetime scheduler for a goal of 60 minutes. For the first 30 minutes the load is light and the microserver accumulates an energy surplus. For the last thirty minutes it uses the surplus at the detriment of meeting deadlines.

operate correctly, Triage must accurately profile the energy use of tasks and track the overall energy consumption of the microserver to account for profiling errors. For the first 30 minutes, the server consumes less energy than is required to meet its lifetime goal. During this time the server's workload is insufficient to drain the bucket, so Triage is free to schedule all tasks and therefore meets its deadlines. After operating for 10 minutes under a more intense workload, Triage continues to meet its deadlines, but it begins to consume the surplus energy that has accrued. At 45 minutes, Triage runs out of surplus energy and begins to sacrifice latency constraints for conserving energy. Note that Triage meets the lifetime goal with an excess energy of about 3mAh. The WoW* and PSM-DVFS systems are unable to meet the lifetime constraint. Their batteries die out at 38th and 21st minutes respectively. Therefore, these systems are left without any battery for 37% and 65% of the time.

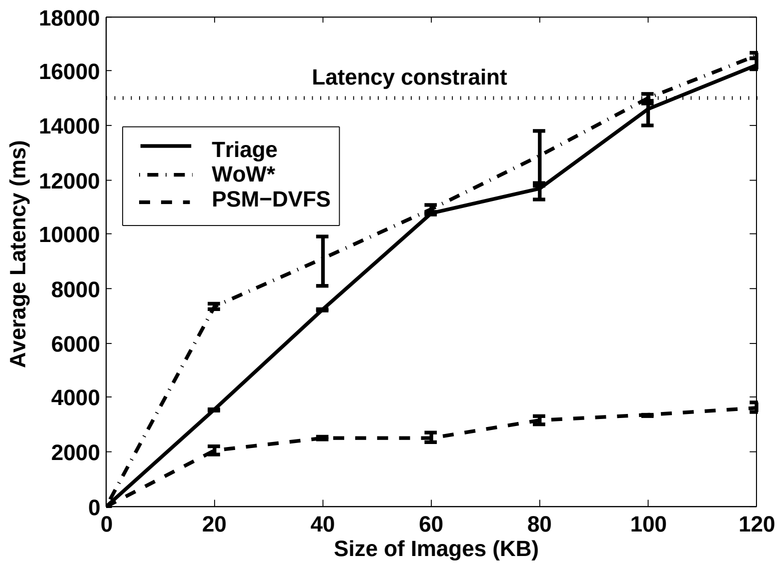


Figure 4.10. Time Profile Accuracy. This shows the forwarding surrogate choosing between two radios based on a latency constraint. For 80KB of data is switches from the 802.15.4 radio to the 802.11 radio to meet the constraint.

4.7.4 Task Profiling

The profiling function of the microserver is essential to providing soft-realtime guarantees. We use simple parametrized linear models for the forwarding surrogate and general histogram models for the more complex tasks like image processing where good models are not known apriori. The model parameters are learned by Triage over time. The efficacy of the histogram model was shown in Section 4.7.2. We present the efficacy of the parametrized model here. We use an experimental setup where variable sized images are sent from the camera Motes to the microserver at a fixed rate of one image per minute to be routed to some destination Mote. The destination could correspond to a central processing server or another node. Since the linear model for the amount of time taken to route data for the two radios is a function of the amount of data that Triage wants to send over the radio, we vary the size of the images sent during the experiment—this corresponds to images of different resolutions required by a client.

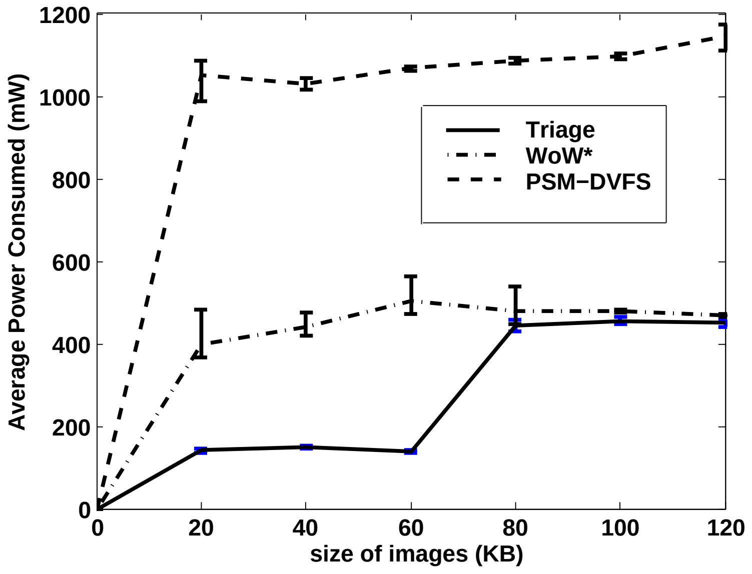


Figure 4.11. Time Profile Power Consumption. This shows the forwarding surrogate power consumption based on the amount of data it sends. At 80KB of data it must switch to the 802.11 radio to meet the latency constraint, thus using more power. The WoW* and PSM-DVFS solutions use more power as they only use the 802.11 radio to send data.

Each forwarding request has a fixed latency constraint of 15 seconds. We show the results of the experiments in Figures 4.10 and 4.11. We compare our results with the PSM-DVFS and WoW* systems. Comparing these systems we see several effects. First, Triage is able to profile the time required for forwarding tasks correctly and send them by their required latency constraint for data sizes less than 100KB. Triage uses the 802.15.4 radio at lower data rates. However above 80KB, Triage must wake the tier-1 system to meet the latency constraint. Second, as Triage can use the TelosB-node alone to transfer data it achieves 200% increase in lifetime over the WoW* system and 500% increase in lifetime over the PSM-DVFS system. The savings clearly demonstrate the benefit of using a tiered architecture. The architecture provides the microserver with the flexibility of using resources on either tier intelligently—leading to substantial energy savings. Third, neither WoW* and Triage are unable to meet the latency constraint for 120KB images. This is due to the USB data transfer bottleneck in our system. The PSM-DVFS system meets all deadlines at a much higher energy cost.

4.7.5 Scaling to QoS Constraints

The primary goal of Triage is to balance energy consumption of the server with a given QoS constraint. Triage uses a combination of task profiling, scheduling, caching, and idle state management to determine the minimal cost at which a QoS constraint can be satisfied.

We perform the following experiment to validate the above claim. Camera Motes feed 100-by-100 images to the microserver at a rate of one per minute. Client Motes request images from the microserver at a rate of one per 30 seconds. Each query request is for a single image taken T seconds ago, where T is exponentially distributed with a mean of 100 minutes. This represents applications where newer data is more valuable to the user than old data. We vary the latency constraint on the task in the experiment. We compare Triage with PSM-DVFS and WoW* systems. The results are shown in Figure 4.12. We see that Triage consumes 300% less power than WoW* and 600% less power than the PSM-DVFS system. Triage uses a combination of serving requests from cached data, scheduling and delayed execution to amortize the transition cost of the tier-1 platform over a long period of time and hence shows large power savings.

4.7.6 Component Power Consumption

Finally, we demonstrate the power consumed by independent components of Triage. We perform an experiment where 100-by-100 images are sent to the microserver at a rate of 1 per minute. The client Motes request classified images with a task arrival rate distributed in the interval $[30, 60]$ seconds. This application provides for a combination of storage, data transfer, processing and data forwarding. The power trace collected is shown in Figure 4.13. The break-down of the average power consumed by different independent components of the system are show in Figure 4.14. We find that the PSM-DVFS system suffers from a huge idle cost of keeping the Stargate platform awake with the WiFi card in PSM-idle mode. This problem is solved by the WoW* system by using the hardware architecture of Triage and duty-cycling the Stargate. However, the WoW* system suffers from a huge transition

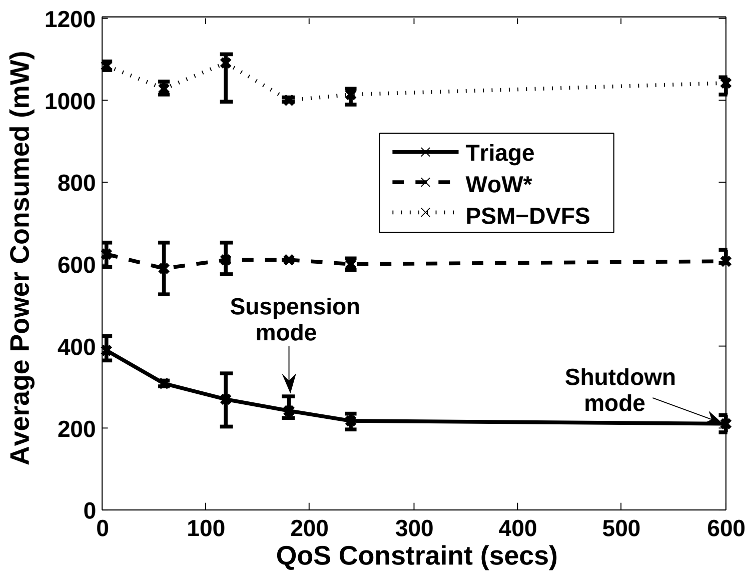


Figure 4.12. Scaling to QoS constraints. Triage finds the minimal energy cost at which a QoS constraint can be satisfied. The WoW* system and PSM-DVFS system consume 3x and 5x more power than Triage respectively.

energy cost, since it has to wake up the Stargate on every task arrival. The above problem is solved by Triage using its software architecture and amortizing the transition cost over a long interval of time. However, we find that the USB-transfer cost is a potential bottleneck for both the Triage and WoW* systems and the performance of Triage could be improved if the bottleneck is eliminated using low power DMA (direct memory access) between the Mote and the Stargate.

Till now we detailed the design, implementation, and evaluation of Triage: a HPM instantiation for designing sensor net microservers. We showed that by combining diverse hardware platforms into a single tightly integrated platform and intelligently sharing tasks among them, we can design systems which are highly available and energy-efficient at the same time. To provide further external validity to the efficacy of the HPM architecture, we motivate and describe Turducken: a HPM instantiation for designing highly energy efficient laptops.

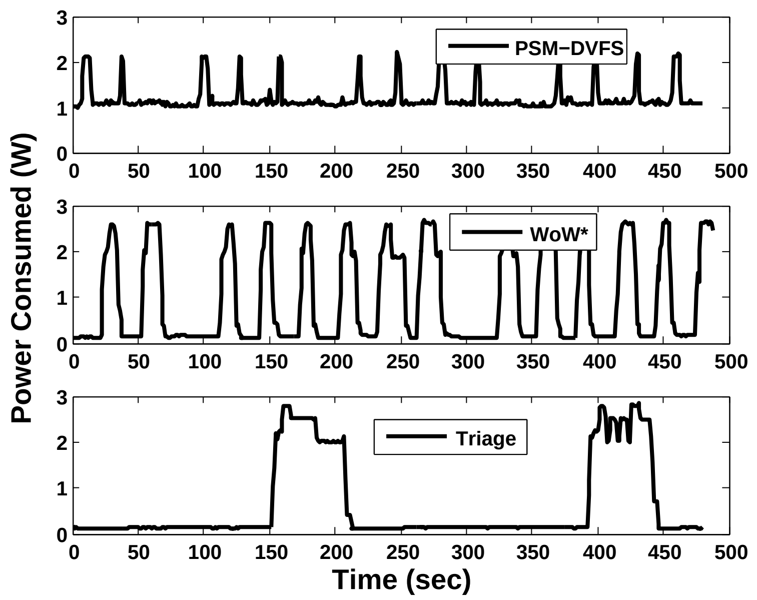


Figure 4.13. Power Traces for the Triage, WoW* and PSM-DVFS systems.

4.8 Turducken: A HPM Architecture for Laptops

Turducken is a HPM instantiation for designing highly energy-efficient laptops. Turducken combines three diverse tiers: a x86-based laptop, StrongARM-based PDA, and an ATmega-based Mote into a single integrated laptop. Through a proxy based software architecture for task sharing among tiers, Turducken achieves high levels of availability at minimal energy consumption. Below, we describe the components of a working Turducken system prototype. The prototype currently consists of a hardware implementation and three applications: time synchronization, web caching, and IMAP synchronization.

4.8.1 Hardware Implementation

The hardware prototype, shown in Figure 4.15, consists of three tiers: an x86-based IBM Thinkpad X31, a Compaq iPAQ 3870 StrongARM-based PDA, and a CrossBow Mica2Dot ATMega-based Mote. The Mote and iPAQ are directly connected via a serial interface and the iPAQ and the laptop are directly connected via a USB interface. The Mote can wake the iPAQ through the use of the serial DCD line, and the iPAQ can wake the laptop

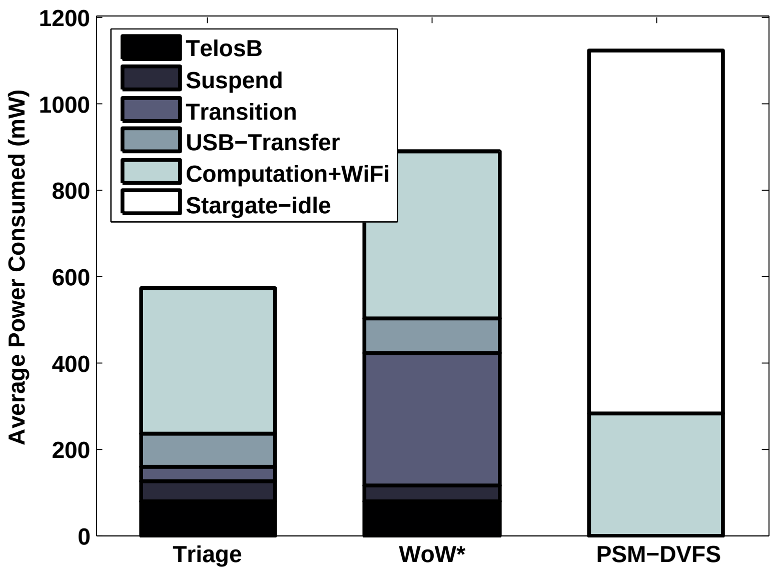
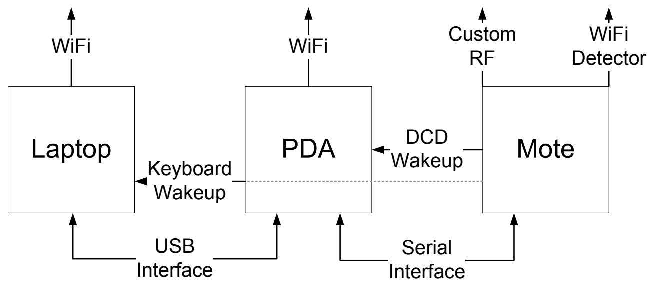


Figure 4.14. Break-down of the power consumed by independent components of Triage, WoW* and PSM-DVFS systems.

by sending a request to the Mote, which wakes the laptop by triggering a relay connected to the keyboard. Our prototype can currently be reconfigured as: x86, x86+ATMega, or x86+StrongARM+ATMega. Each tier also contains a real-time clock (RTC) that can generate a wake interrupt. If we reconfigure the system as x86 only, it can suspend itself and use its RTC to wake it at set intervals.

This prototype differs from our ideal HPM design in three significant ways. First, the hardware components are all physically separate—a deployed system would integrate all of the components into a laptop form-factor. The connections shown in the picture would all be internal to the system. Second, there is a plethora of extra parts in our prototype. An integrated implementation would eliminate much of the PDA, including its screen, sleeve, and buttons. Third, each tier is run from its own battery. The Turducken design assumes that there is only a single, shared battery. This has implications for how we evaluate the system.

In our implementation, there are two types of wireless interfaces: WiFi and the Mote's custom radio interface. There are both advantages and disadvantages to having access to multiple wireless standards. It does allow the system to take advantage of a broader range of



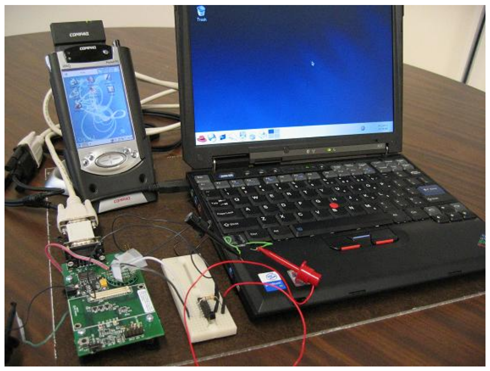


Figure 4.15. These figures show the prototype implementation of the Turducken System. The diagram on the left shows the logical connections between components and the photo on the right shows the current prototype.

services by allowing it to communicate with more devices; however, it makes system design more challenging since certain tasks may require a particular network interface and thus it cannot be accomplished by all tiers. To mitigate this disadvantage, we have attached a WiFi detector to the Mote. The detector can determine if WiFi signals are present, though it cannot communicate using WiFi or discover if an access point is open or closed.

Even though the x86 and StrongARM tiers each have WiFi interfaces, there is no reason to use them both in the Turducken system. In a configuration that includes both, we turn off the x86-tier's interface and route all traffic through the StrongARM-tier. This saves power, thus extending the battery lifetime of the system.

4.8.2 Applications

We have developed and deployed three applications that are representative of commonly-used mobile distributed services: time synchronization, web caching, and IMAP synchronization. Time synchronization is necessary for timestamping distributed updates and determining timeouts in soft-state protocols. Web caching on mobile devices allows the mobile node to serve pages during periods of disconnection and improves response time when connected. IMAP synchronization maintains a local mail cache that can serve mail

	Execution Tier	Incoming or Outgoing
Time Sync	$\geq$ATmega	Incoming
Web Cache	$\geq$StrongARM	Incoming
IMAP Sync	$\geq$StrongARM	Both

Table 4.2. This table shows a summary of the application characteristics. The execution tier denotes where the application is carried out, and Incoming or Outgoing describes the direction of updates.

during periods of disconnection and improves response time. In addition, a local IMAP store can buffer outgoing mail and send it when the node is connected.

These applications also represent three broader classes of applications. These classes are defined by the traits listed in Table 4.2. Time synchronization represents applications that require limited processing and limited transmission of incoming data updates. Web caching represents applications that require more significant processing and larger amounts of incoming data. This is similar to a variety of publish-subscribe systems. IMAP synchronization represents applications that require fairly significant processing and support for outgoing as well as incoming updates. This is similar to the requirements of distributed file and database systems, though the consistency requirements are not as strict.

4.8.2.1 Time Synchronization

The time synchronization application follows the system-aware programming model. The ATmega-tier runs a custom built Network Time Protocol (NTP) client that synchronizes its local clock with a known time server every t seconds. The StrongARM and x86 tiers can then request the current time from the ATmega-tier and update their local clocks. We define an explicit API for this communication. When the ATmega-tier is not present, the x86-tier uses its RTC to wake every t seconds and synchronize with the remote time server using the UNIX utility `ntpdate`.

4.8.2.2 Web Cache

The web cache application follows a proxy-based programming model. The ATmega-tier detects the presence of a WiFi signal; the StrongARM-tier runs a Squid proxy cache; and the x86-tier runs a web browser. Every t seconds, the ATMega determines whether a WiFi connection is available and, if so, wakes the StrongARM-tier. The StrongARM-tier remains awake for 30 seconds while the proxy continuously fetches expired cache items. Web requests originating from the web browser running on the x86-tier are routed through the StrongARM-tier. These requests can be transparently serviced by the proxy when no network connection is available.

When the StrongARM-tier is not present, the Squid proxy runs on the x86-tier and the cache is stored on the system's hard disk. The ATmega-tier or RTC wakes the x86-tier every t seconds. If a connection is present, it remains awake for 30 seconds while the Squid proxy fetches expired cache items. Again, the Squid proxy can transparently fulfill requests from a web browser.

4.8.2.3 IMAP Synchronization

The IMAP synchronization application also follows a proxy-based programming model. The ATmega-tier detects the presence of a WiFi signal and the StrongARM-tier runs a UNIX utility named `mailsync`, which performs synchronization between an IMAP server and a *secondary* mail store. The x86-tier maintains the *primary* mail store and uses `mailsync` to synchronize with the StrongARM-tier's secondary mail store. The x86-tier also runs the user's mail client. Every t seconds, the ATmega-tier determines whether a WiFi connection is available and, if so, wakes the StrongARM. The StrongARM-tier uses `mailsync` to retrieve incoming mail from and send outgoing updates to the user's mail server. Incoming mail is stored in the secondary mail store hosted on the StrongARM-tier.

When the user turns on the x86, it synchronizes its primary store with the secondary store on the StrongARM-tier. The user accesses mail by configuring the mail client to point to the primary mail store on the x86-tier. When the user suspends the x86-tier, any changes the user has made will be synchronized with the StrongARM-tier which will synchronize with the remote mail server when connected.

In some cases, the user may receive pieces of mail that are too large to be stored in the StrongARM-tier's flash memory. To accommodate this scenario, the primary mail store also synchronizes with the remote mail server when possible. In addition, we would like to modify the StrongARM-tier to wake the x86-tier when it detects this situation, though we have not yet implemented this feature.

If the StrongARM-tier is not present, the x86-tier synchronizes directly with the remote mail server when connected. Similar to the web cache, the ATmega-tier or RTC wakes the x86 every t seconds. If the x86-tier discovers that no connection is present, it goes back into a suspended mode without performing synchronization.

Both the IMAP synchronization and the web caching applications were implemented using standard components. Due to the distributed nature of these applications, recoding is not necessary in order to deploy them on our prototype Turducken system. Each component can simply be recompiled for both the x86 and StrongARM architectures.

4.9 Turducken Evaluation

The primary goal of Turducken is to extend the lifetime of a mobile computing device while allowing it to remain aware of its environment when not actively in use. In our evaluation of the Turducken system, we measure the lifetime of several Turducken configurations running the following three sample applications: time synchronization, web caching, and IMAP synchronization. For each application, we compare the system lifetimes of different configurations with respect to data consistency. Finally, we focus on the web caching application and compare system performance with respect to variable network and service availability.

4.9.1 Methodology

Our evaluation measures the lifetime of several system configurations running varied workloads. Measuring the lifetime of a Turducken system presents a number of interesting challenges. Explicitly measuring the lifetime of a single configuration running a single workload can take longer than a week. Collecting even a small number of data points using this method is impractical with only a single prototype system. We address this problem using time dilation. For each experiment we measure the energy consumed by the system under a small number of workloads. Using these measured values, we use extrapolation to project system lifetimes over a wide range of data points.

We calculate the lifetime of the system from the average energy that is required to run a particular application under a given workload for a set period of time. From this value we can estimate how long it will take the system to drain a battery of known capacity. We also make the assumption that the power draw of a full system will be no greater than the sum of the power draw of each tier. This estimate is conservative since an integrated system can use more power-efficient communication links between tiers.

For the experiments presented here, we measure the amount of energy consumed by each tier using the tier's native power management interface. Batteries used in modern mobile devices typically contain a gas gauge chip, such as the Texas Instruments BQ2011 chip used in the x86-tier's battery, which considers temperature, battery chemistry, and past usage to accurately compute the amount of energy remaining in the battery. This approach is sufficient to measure the average energy consumption over a particular period of time. This is similar to the method used to measure power consumption of the Odyssey System [59].

Using this method we measure the energy consumed by each tier over a fixed period of time, and calculate the amount of time it takes the entire system to drain a full battery. This calculation depends on several factors: the average power draw of each tier while active; the average power draw of each tier while suspended; the amount of time each tier spends active; and the amount of time each tier is suspended.

We measure the power draw of both the x86-tier and the StrongARM-tier in suspended mode over a 10 hour period of time. The energy in the battery is sampled immediately before and after the period of suspension in order to determine the total energy consumed. We divide this value by the total experiment time to obtain the power draw of each device in suspended mode. For the StrongARM-tier, we obtain the full battery capacity from the manufacturer's specification. For the x86-tier we use the estimated capacity specified by the device's battery.

To determine the power draw of the x86-tier and StrongARM-tier in active mode we run each application on each system configuration for a 24-hour period. During all experiments, we turn off both the screen and backlight of the two higher tiers in order to make a more fair comparison. For each device, we measure the amount of time it is active, t_A, the amount of time it is suspended, t_S, and the total energy, E, consumed by the tier. Using the total amount of time suspended and the suspended power draw, P_S, we calculate the energy consumed while suspended, E_S over the 24 hour period:

$$E_S = P_S t_S. \tag{4.2}$$

We then use the total energy, E, and the energy used while suspended, E_S, to compute the energy used while active:

$$E_A = E - E_S. \tag{4.3}$$

By dividing the energy used while active by the amount of time the system is active, we obtain the power draw, P_A, of each tier in the active state:

$$P_A = \frac{E_A}{t_A}. \tag{4.4}$$

The resulting power draws are shown in Tables 4.3, 4.4, and 4.5.

Mode	x86	ATmega
Active (mW)	11,600	26.4
Suspended (mW)	180	0.056

Table 4.3. The active and suspended power consumption of each tier running the time application. The active power consumption for the StrongARM-tier was not measured since it never synchronizes with the time server.

Mode	x86	StrongArm	ATmega
Active (mW)	10,955	740	26.4
Suspended (mW)	180	40	0.056

Table 4.4. The active and suspended power consumption of each tier running the web caching application.

For the ATmega-tier, we assume it will be always on and establish a generous upper bound on the power draw from the Crossbow datasheets. Even using this upper bound, the power draw of the ATmega-tier has very little impact on the lifetime of the system.

Using these individual measurements, we calculate the power draw of the full system as the sum of the power draw of each tier in the appropriate state. Using this value, we calculate the amount of time it takes the entire system to drain the entire battery of the x86-tier.

4.9.2 Consistency

The goal of our first set of experiments is to vary the level of consistency required and observe the consequent lifetimes of several system configurations. To accomplish this, we vary the interval at which the system wakes to perform synchronization from 0 (always on) to 0.5 hours. A wake interval of *i* minutes ensures that data is inconsistent for no longer than *i* minutes.

For each of these experiments, a wireless network is always present, the remote service is available, and new data updates are ready. For the time synchronization application, we assume that the time is synchronized whenever the system wakes. For the web caching

Mode	x86	StrongArm	ATmega
Active (mW)	11,720	810	26.4
Suspended (mW)	180	40	0.056

Table 4.5. The active and suspended power consumption of each tier running the IMAP synchronization application.

application, the system maintains a 5 MB cache consisting of 15 web sites. For the IMAP synchronization application, the Turducken system fetches data updates and sends any queued, local updates upon waking. For this experiment, the x86-tier wakes for 2 minutes of every hour to simulate a user creating modifications to the local mail store. This store initially contains 4MB of mail in four separate folders. The queued updates to the local store are sent to the remote IMAP server when the StrongARM-tier wakes to synchronize. In addition, new mail is sent to the inbox at a rate of 120KB per hour. During synchronization, the Turducken client fetches this mail.

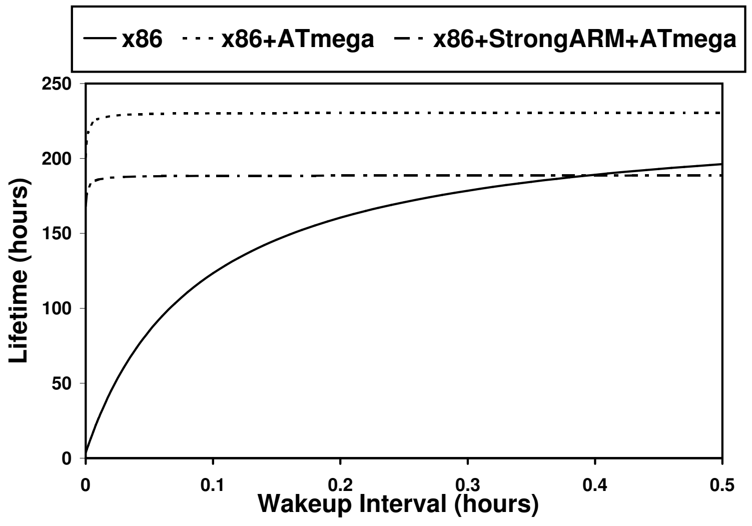


Figure 4.16. The lifetime of three system configurations running the time synchronization application. As the system wakes more frequently, Turducken provides a more significant gain in lifetime.

The results of the time synchronization experiment are shown in Figure 4.17. When the system synchronizes frequently, the lifetime of the x86-only system degrades drastically while both the x86+StrongARM+ATmega and x86+ATmega configurations maintain nearly constant lifetimes. This is a consequence of the fact that when using a Turducken system, the x86 and StrongARM tiers never need to come out of a suspended state. In this case, the x86+ATmega configuration has a lifetime of about 225 hours and the x86+StrongARM+ATmega has a lifetime of approximately 180 hours. The difference between these two configurations is a result of the energy draw of the StrongARM-tier in suspended mode.

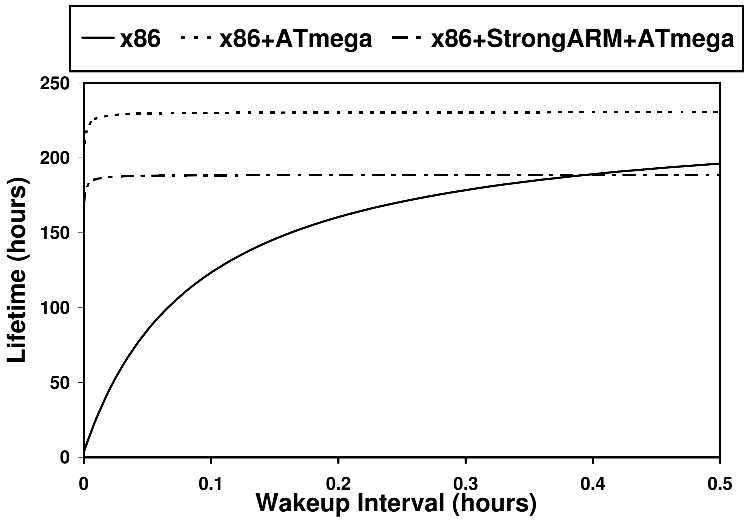


Figure 4.17. The lifetime of three system configurations running the time synchronization application. As the system wakes more frequently, Turducken provides a more significant gain in lifetime.

Figure 4.18 shows the results of the web caching experiment. We observe that the x86+StrongARM+ATmega consistently performs better than the other configurations, providing a ten times improvement for always on operation and a three times improvement when the system wakes up every six minutes. Additionally, we observe that as the wake interval grows, the lifetime gain lessens. This is a result of the energy required to power the

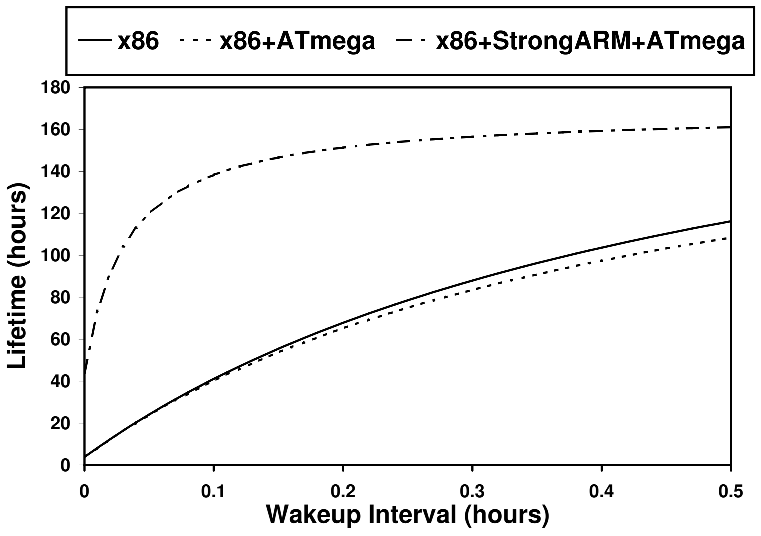


Figure 4.18. The lifetime of three system configurations running the web caching application. For this application, the full three-tiered Turducken system offers up to a 4 times longer lifetime and consistently performs better than the x86-only configuration.

StrongARM-tier in suspended mode. Similarly, the x86+ATmega system performs worse than the x86-only configuration for larger wake intervals because of the additional energy required to power the ATmega tier. Again, we can conclude from these observations that the higher the level of consistency required, the better the performance of Turducken.

Figure 4.19 shows the results of the IMAP synchronization experiment. The relative performance for IMAP synchronization is very similar to the web caching application, however, we observe that the absolute system lifetimes are significantly smaller. This is a result of the workload of IMAP synchronization. This particular experiment requires that the x86-tier wake periodically to simulate a user updating the local mail store, which costs additional energy. This application also introduces additional outgoing network traffic which impacts energy usage. However, we still observe that Turducken enjoys at least a 150% improvement in system lifetime for wakeup intervals less than six minutes. If the x86-tier does not perform periodic synchronization and only wakes up once an hour to send and receive updates its average lifetime is found to be 75 hours. However, the cost of this

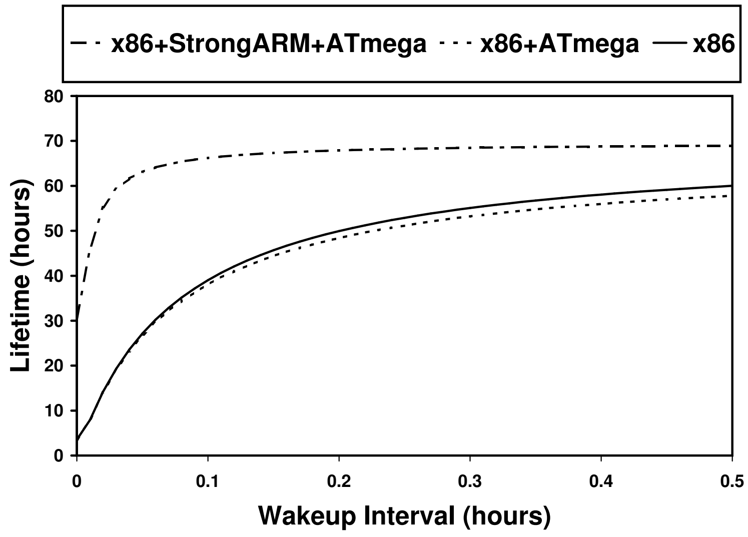


Figure 4.19. The lifetime of three system configurations running the IMAP synchronization application. For this application, the full Turducken system offers a 1.5 times longer lifetime and consistently performs better than the x86 only configuration.

gain in system lifetime is that the expected time to get an update is $\frac{1}{p}$ hours, where p is the probability of a network connection being available. Since this latency can be large for small values of p, it is reasonable to sacrifice 13% of the system's lifetime in exchange for one-tenth the expected latency.

Figure 6.4 shows the average power draw for each tier. Each bar represents the total average power consumed by a particular configuration running a particular application. A bar is composed of several components that show each tier's contribution to the average power draw of the entire system. We further decompose each tier's contribution into its active and suspended modes. For example, for the x86-only configuration running the time application, the graph shows that the x86-tier spends most of its time suspended and a small amount of time in its active mode. Similarly, when it is augmented with an ATmega-tier, it spends all of its time suspended and the ATmega-tier expends a negligible amount of power. In the web caching experiment, the x86+StrongARM+ATmega configuration is able to replace the active power of the x86-tier with the StrongARM-tier. The mail experiment

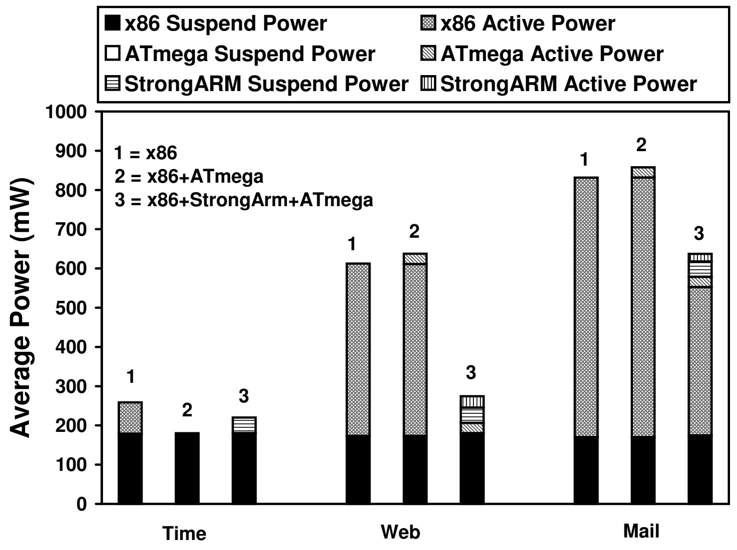


Figure 4.20. This figure shows how each tier, in different states, contributes to the average power draw of the system as a whole. We observe that Turducken systems achieve battery lifetime gains by replacing active power consumption in less efficient tiers with more efficient ones.

sees a similar gain; however, because the x86-tier spends more time in active mode, the resulting active power draw is larger. We observe that Turducken systems achieve lower average power consumption by replacing active power consumption in less efficient tiers with more efficient ones.

4.9.3 Network and Service Availability

The goal of our second set of experiments is to vary the availability of a wireless network and the availability of the required service, and observe the consequent lifetimes of several system configurations. For this set of experiments, we look exclusively at the web caching application and fix the wake interval at 12 minutes. In the first experiment, we vary the probability that a wireless network is available from 0 (network never available) to 1 (network always available). In the second experiment, we fix the probability of wireless network availability at 1 and vary the probability that a set of web servers is reachable from 0 (web servers never reachable) to 1 (web servers always reachable). For this experiment,

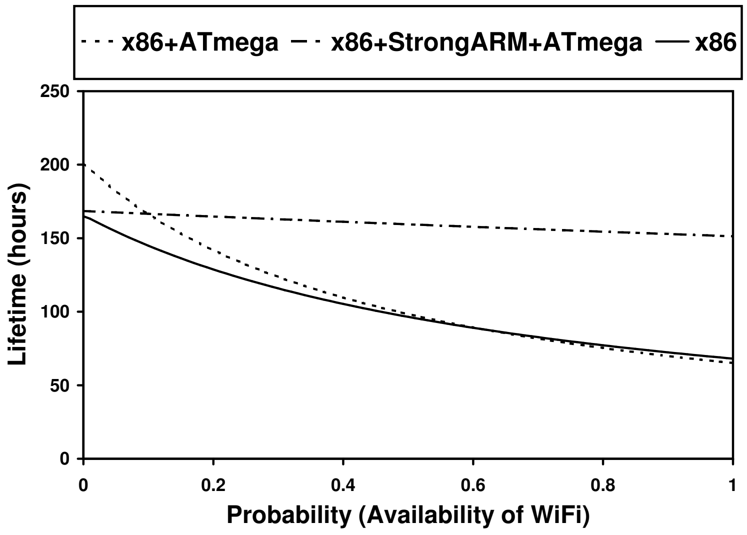


Figure 4.21. This figure shows the battery lifetime of different configurations with respect to varying the probability of availability of WiFi. As network coverage increases, Turducken provides a greater benefit.

we assume that either all web servers are reachable or no web servers are reachable and we assume that it takes a trivial amount of time to determine reachability for all servers.

The results of varying the network availability are shown in Figure 4.21. When the probability of WiFi is low, the x86+ATmega system performs best. This is because it can avoid waking the x86-tier if no signal is present. The x86+StrongARM+ATmega system enjoys the same benefit, but incurs the cost of powering the StrongARM-tier in suspended mode. Interestingly, the x86-only configuration performs similar to the x86+StrongARM+ATmega for low probabilities. This implies that the cost to periodically wake the x86 to discover that no network is present is roughly equivalent to the cost of powering the StrongARM and ATmega tiers in suspended mode. As the probability of a network connection increases, the x86+StrongARM+ATmega system remains nearly constant, outperforming the other configurations by up to a factor of 2. This is a result of the energy saved fetching web pages using the StrongARM-tier without waking the x86-tier. We can conclude that Turducken

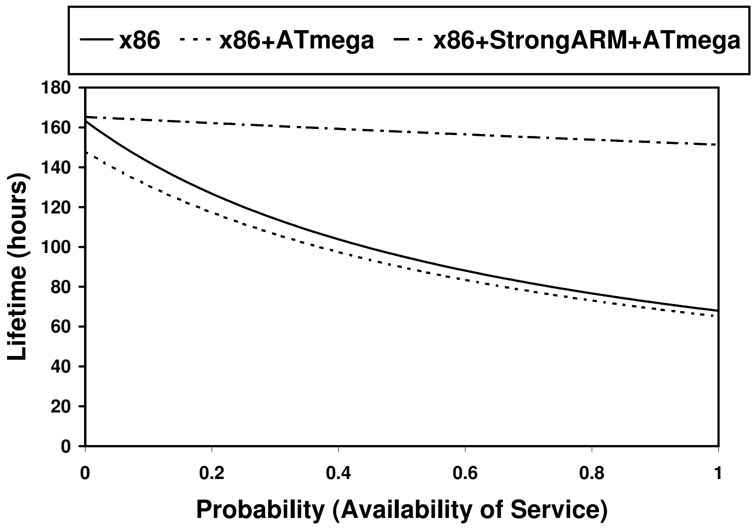


Figure 4.22. This figure shows the battery lifetime of different configurations with respect to varying probability that a set of web servers is reachable. The benefit of Turducken is evident as the probability that the servers are available increases.

provides a greater benefit as network coverage increases, and performs no worse than an x86 alone as coverage decreases.

The results of varying the availability of web servers is shown in Figure 4.22. The results for this experiment are similar to the previous experiment with the exception of the x86+ATmega configuration. While the ATmega-tier can determine the presence of WiFi, it cannot determine the reachability of a web server. Therefore, the ATmega-tier must always wake the x86-tier to determine if the web servers are reachable. This costs the x86+ATmega configuration up to 40 hours of lifetime. However, as the probability of service increases, the benefit of Turducken increases.

4.9.4 Observations

Our primary observation is simple: for many common distributed applications, a Turducken system can maintain a high level of consistency at a fraction of the power cost of a conventional laptop. This allows system behavior which has traditionally been ruled out in favor of conserving battery power. Naturally, there is a cost incurred when powering additional devices. This cost becomes noticeable when the system wakes up less frequently, reducing the benefit and retaining the cost of the additional hardware. Fortunately, even if the system never wakes up, the x86+StrongARM+ATmega configuration will last 82% as long as the x86-only system.

Our experiments have also shown that the main limiting factor of the system's battery lifetime is the suspended power draw of the x86-tier. Our proposed solution to this is to use hibernation, which involves saving the machine's state to disk and powering it down. When the system is restored, it boots to the previously saved state. Clearly, it will cost more in both energy and latency to wake a device out of hibernation; however, during times of little or no activity (e.g. at night), using hibernation could result in significant power savings, potentially extending the system's lifetime to *over a month* on a single charge.

Additionally, it is clear that the benefit achieved is highly application dependent. For example, in the case of very simple applications, like time synchronization, the x86+ATmega configuration achieves the best performance. The best set of tiers for a particular Turducken system depends on the target applications that the system will host.

4.10 Discussion

In this chapter we presented a new paradigm for energy management in mobile and sensor systems called Hierarchical Power Management. HPM can provide high levels of consistency in mobile and sensor systems at minimal energy consumption. We presented two instantiations of HPM—Turducken and Triage. While Turducken is a system architecture for combining a x86 with a StrongARM with a ATMega platform to build highly energy efficient laptops, Triage is a clean slate software and hardware design for combining a PDA class device with a Mote for sensor network microserver design. While HPM can help build highly energy-efficient mobile and sensor client devices, it is a solution to only half the problem of providing *improved network consistency and connectivity in mobile and sensor* *systems*. In the next chapter we argue that without proper support and enhancements to the associated network infrastructure, the vision of improved performance for networked applications and improved connectivity will remain a myth.

CHAPTER 5
ENHANCING MOBILE NETWORKS WITH INFRASTRUCTURE

Energy-Efficient mobile and sensor clients can provide improved network availability at minimal energy consumption. However, without proper support from the network infrastructure, providing improved network connectivity at clients will always remain a myth. For example, consider a wildlife tracking application [147]. If the mobile client devices are energy-efficient they can collect large amounts of data. However, if the network is sparse and mobile node contacts are meager, packet delivery delays to base stations could be of the order of days. As another example consider a laptop user who accesses the web using open WiFi connections when mobile. If the laptop runs the Turducken system (see Chapter 4), it will have improved lifetime and always-on availability. However, if connectivity using open WiFi is intermittent there is no way an always-on client can provide high levels of consistency to network applications.

One way to improve connectivity in mobile networks is by adding infrastructure/stationary resources [20, 38, 66, 79]. This includes using open WiFi base stations [20, 38, 78], mesh networks [32], and relays [79]. While building and nomadically deploying these alternate infrastructure enhancements presents a significant systems challenge, the first step towards using relays, base stations or mesh nodes to enhance mobile networks is to understand their relative benefits. The study of different types of infrastructure enhancement under one unified framework is extremely important to understanding their relative cost and benefits—an element lacking in the present literature. In fact, there are several research questions related to alternative infrastructure enhancements which are still poorly understood. For example, if opportunistic base stations are available, is mobile-to-mobile routing necessary?

If opportunistic access to base stations is unavailable, should one deploy relays, base stations or mesh nodes to reduce delays? How do these trade-offs scale as the network grows in size or density?

To answer these questions, in this chapter we present results from a set of field experiments and analysis that compares the benefits of each kind of hybrid mobile network. While the deployment study demonstrates many of the practical issues involved, including node placement, real world propagation, and dynamic routing schemes the results heavily depend on the particular system, underlying technology, and mobility patterns. To address this shortcoming, we use observations from the deployment study to develop analytical models for large scale hybrid networks based on ordinary differential equations (ODE).

5.1 Vehicular Network Deployment

We used the DOME testbed (see Chapter 3) to deploy alternative infrastructure enhancements. To examine different kinds of infrastructure, we deployed three different kinds of networks:

Relays: We placed six stationary relay nodes in the network for a period of 20 days. The nodes consist of a PDA-class Stargate device equipped with a WiFi CF card and 64 MB of storage (see Figure 5.1). When a vehicle comes within WiFi range of a relay, the two exchange data over a TCP connection until the contact opportunity ends. The relay node stores those packets, waits for another vehicle, and then exchanges packets with that vehicle, propagating packets from mobile node to mobile node.

Mesh Nodes: These same stationary nodes can be configured as a mesh network. Using an additional radio, in this case a 900MHz Digi-XTend radio, the nodes form a mesh network. When one of the mesh nodes receives data from a vehicle over WiFi it propagates those packets to other nodes in the mesh network.

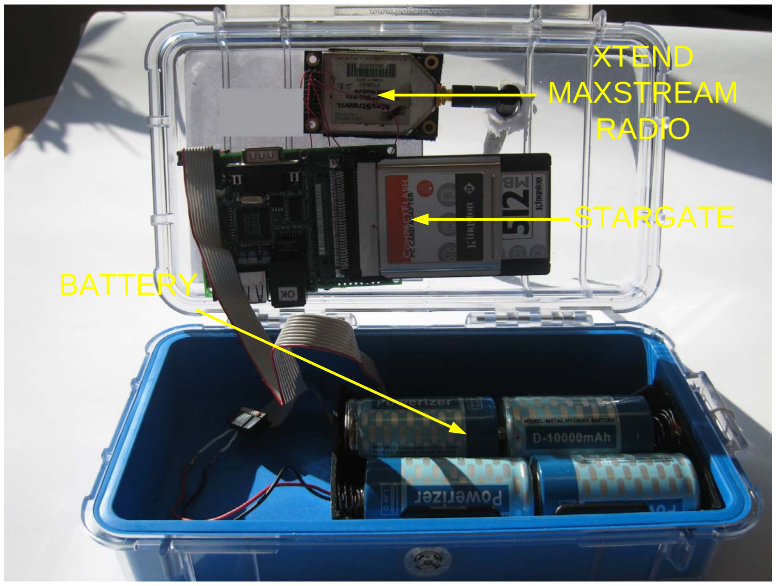


Figure 5.1. Prototype for a relay/mesh node.

Characteristics	Network-core	Network-periphery
average pairwise inter-contact time	434 min	506 min
average contact duration	8590 ms	2802 ms
number of contacts (per day)	145	7

Table 5.1. The above table shows the characteristics of the two regions in the network.

Base Stations: For base stations, we used a combination of APs that we deployed and a set of open-access points set up by third parties. When a vehicle passes an AP it exchanges data with a central server through the AP, reachable by all of the base stations.

We collected only data inherent to the network and type of infrastructure so that we could later apply the traces to a variety of scenarios. Each node, both mobile and stationary, recorded its contact with other nodes and their durations, as well as the amount of data transferred during the contacts. For the mesh nodes, we collected data on the WiFi-based vehicle-to-mesh and the XTend-based mesh-to-mesh connections. The mobile nodes collected detailed mobility traces from GPS units.

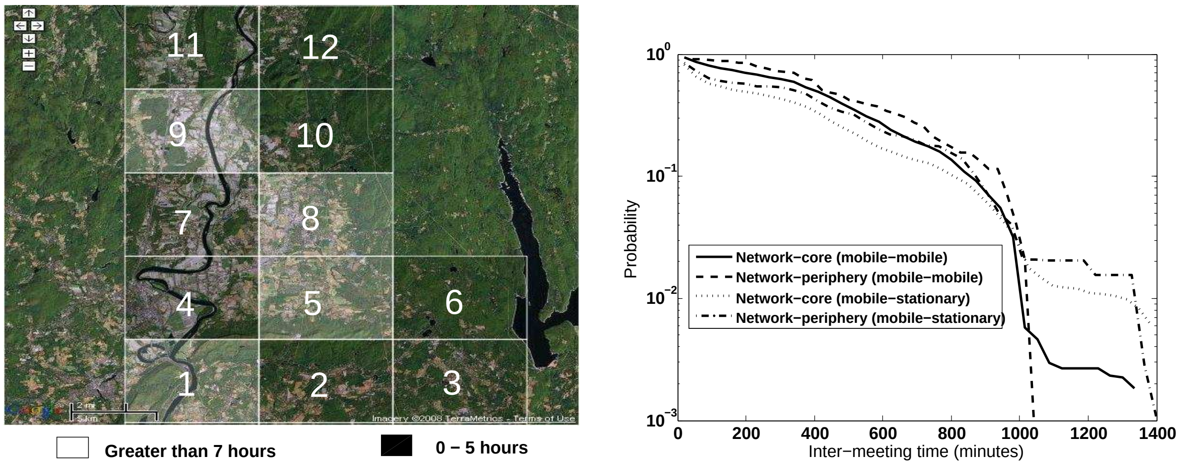


Figure 5.2. Heatmap showing the amount of time vehicles spend in different regions of the network as well as the CCDF of the aggregate pairwise inter-meeting times between contacts for mobile nodes and mobile and stationary nodes.

5.1.1 Network Characteristics

To understand how placement and mobility effects hybrid network performance, we first look at the geographic characteristics of our mobile network testbed. Figure 5.2 shows a heatmap of the amount of time vehicles spend in different parts of the network. The darker squares are areas where vehicles spend the least amount of time while lighter regions are areas where vehicles spend the greatest amount of time. From the figure, we can divide the network into two disjoint regions: (1) a *network core*, where nodes reside for a large fraction of time (squares 1, 5, 8, and 9 in Figure 5.2); and (2) a *network periphery*, where nodes spend the rest of the time. Figure 5.1 shows the characteristics of these two regions. Note that the pairwise contact durations, the pairwise inter-contact times and the overall number of contacts is different for both regions.

We later use this dichotomy of regions to develop accurate mathematical models for mobile networks with infrastructure. Further, this feature is not unique to this kind of deployment—a dichotomy of regions is inherent in many other mobile networks. For example, in a network of humans [76], nodes are likely to spend more time in certain parts of the network as compared to others. The partition of the mobile network also provides insight on how stationary nodes should be placed in a mobile network to maximize performance.

While the two regions exhibit different absolute contact statistics, the pairwise inter-contact times between mobile nodes and mobile and stationary nodes for both regions have similarly *shaped* distributions. Figure 5.2 shows the CCDF of aggregated inter-contact time in each region. From the graph we see that 90% of all the contacts approximately follows an exponential distribution. The noisy behavior in the tail of the distribution is due to end-of-day effects, called *partially observed contacts*—these are contacts that have a start time but not an end time recorded in the logs. We calculate the inter-meeting time for such contacts using the end of the day as the last contact between the node pair (similar to previous work in vehicular networks [169]).

5.1.2 Trace-driven Simulator

Using the traces taken from our deployment, we built a trace-driven simulator to evaluate the performance benefits of alternatives to enhancing a mobile network. The simulator can be used to examine three key factors that affect performance: the type of infrastructure enhancement, the placement of that infrastructure, and the choice of the routing protocol. Examining each type of infrastructure enhancement (relay, mesh, and base stations) is straightforward, we modify the simulator to simulate each communication path—relays can only communicate with mobile nodes, meshes and base stations with other infrastructure nodes and mobile nodes. Modifying the simulator for routing protocols is also possible: the packets exchanged in the simulator between nodes is determined by the particular algorithm we choose.

Placement requires restricting infrastructure to a set of feasible locations. Relays can be placed anywhere in the network. Using solar cells and batteries, relay nodes can be placed independent of power and network infrastructure, as well as independent of one another. This does add an additional complexity of energy management on the nodes—something we discuss in the next chapter. Mesh nodes are more constrained, in that they must be placed within range of one another (in the case of our mesh nodes, this range is 1650 meters). We have built these mesh nodes from the same solar-powered boxes as the relays, and the collection of connected nodes can be placed anywhere subject to the maximum distance constraint. The base stations can only be placed where there is wired connectivity. However, they do not need to be proximate to one another. A pair of base stations can communicate across the full geographic diameter of the network. To approximate the placement constraint for base stations, we assume that the base stations can only be placed at current access point locations.

The remaining parameters come from the traces themselves. We use the vehicle GPS traces to calculate the time and duration of contacts between mobile and stationary nodes. We use measured values for the WiFi radio (100 meters) and the XTend Maxstream radio (1650 meters) from our six node deployment. The simulator uses the vehicle-to-relay and vehicle-to-AP bandwidth distributions measured from our deployment for the amount of data transferred during each contact. For the mobile-to-mobile contacts, we use the vehicle-vehicle connections logged by the nodes in our testbed. The data includes the time, location, duration of contacts, and the amount of data transferred during connection events. The movement of vehicles is taken directly from the GPS traces.

5.1.3 Placement

We use two strategies for stationary node placement. (1) uniform: place nodes uniformly across the entire network (2) non-uniform: place more nodes in regions where the vehicles spend a larger fraction of time (see Figure 5.3).

We evaluate the effect of each placement strategy using our trace-driven simulator. As a starting point, we use RAPID [17] as our routing protocol. We chose RAPID since it has been shown to perform close to optimal. We present results from experiments using other routing schemes in Section 5.1.4. We compare performance using two metrics (1) *average*

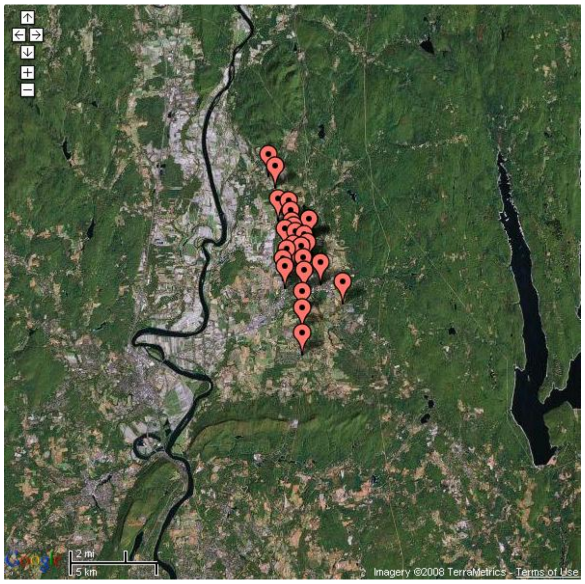


Figure 5.3. Placement of 25 base stations non-uniform across the mobile network.

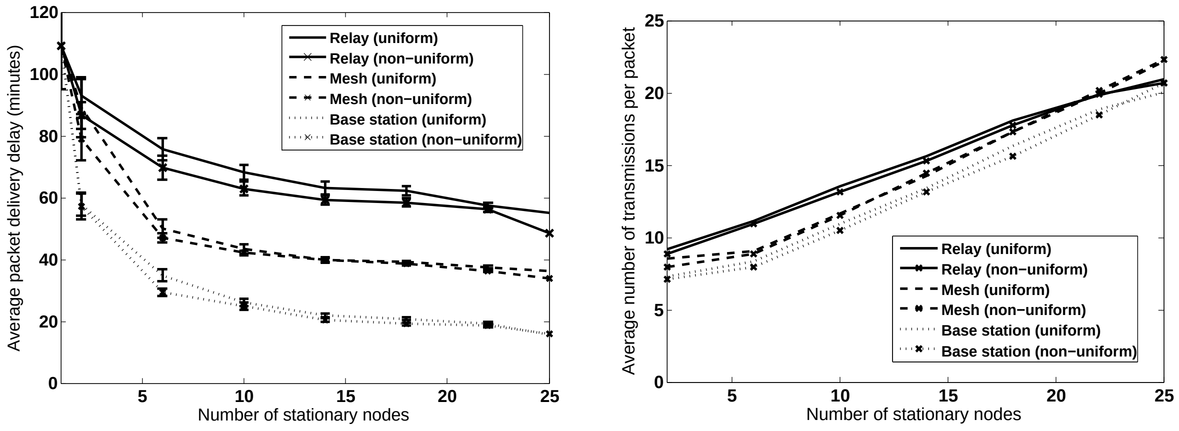


Figure 5.4. The average packet delivery delay and the average number of transmissions per packet with a varying number of stationary nodes for uniform and non-uniform placement

packet delivery delay: the lifetime of a packet from its creation to the first time it is delivered to the destination; (2) *average number of transmissions per packet*: the number of copies made by the time the packet is first delivered to its destination. This can be used to estimate energy and bandwidth as both are proportional to the number of packet copies.

The simulation generates 1KB packets at a uniform rate of 5 packets per destination per hour. This load is intentionally small so that it does not exceed the network capacity. The small load also isolates the effect of delay from bandwidth. The destination for each packet is divided into two equal classes: one half of the traffic is destined for a sink node; otherwise it is equally likely to be destined for any other mobile node. The sink node for the relay and mesh case is a randomly chosen node, and in the base station case it is any base station node.

Figure 5.4 shows the average packet delivery delay for different infrastructures for both uniform and non-uniform placement across a range of numbers of infrastructure nodes. The error bars represent the 95% confidence interval around the sample mean. Figure 5.4 shows the number of transmissions per packet, including replication. From the figures, we make the following observations about our deployment. (1) We need only 5–7 times as many relays or 2–3 times as many mesh nodes as base stations to reduce delay by a factor of two. Given that the cost of deploying base station networks can be greater than 7 times the cost of relays or a mesh network, either relays or meshes are better choices. (2) The non-uniform placement of nodes leads to 10% lower delays as compared to uniform placement. Therefore, placing infrastructure in popular regions can lead to better performance. (3) The average number of transmissions per packet for different infrastructures is similar. In terms of resource consumption, all infrastructure networks are similar.

5.1.4 Dynamic Routing Protocols

To determine the effect of different routing schemes on different kinds of infrastructure, we perform a similar experiment with three routing protocols: (1) RAPID, (2) two-hop: a protocol that does not use any mobile-to-mobile routing—instead the mobile source only passes the packet to the infrastructure, and the destination can only receive the packet from it as well, and (3) Random-epidemic: a base-line protocol that creates copies of a packet randomly, i.e., when two nodes meet the packets they transfer are chosen randomly from their buffers.

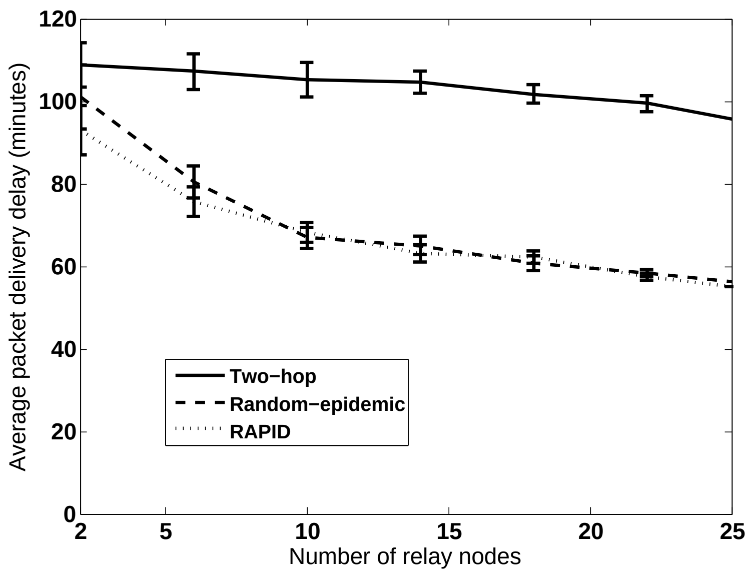


Figure 5.5. The average packet delivery delay with a varying number of relay nodes for the three routing protocols.

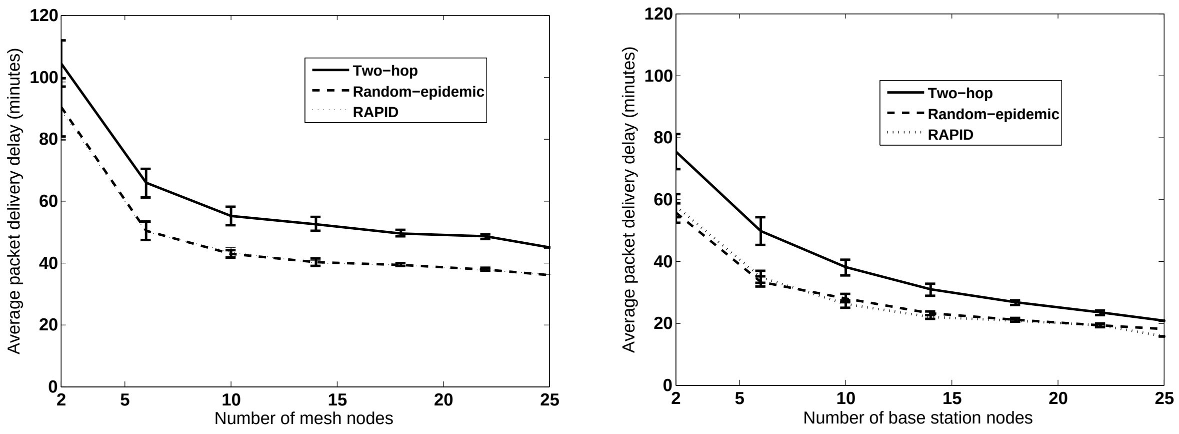


Figure 5.6. The average packet delivery delay by varying the number of mesh nodes and base stations for different routing protocols.

Figures 5.5, and Figure 5.6 compare the average packet delivery delay for random, RAPID, and two-hop routing for relay, mesh, and base station networks. From these graphs, comparing two-hop routing with RAPID, we find that the improvement yielded by mobile-to-mobile routing is marginal for a mesh and base station infrastructure. *Base stations and meshes substantially help reduce delays. However, the additional benefit of mobile-*

to-mobile routing over a two-copy approach yields little additional benefit. Somewhat intuitively, mobile-to-mobile routing helps in the relay case, as the destination must meet one of the same relay nodes that the source saw. Moreover, from the figures, we find that Random-epidemic performs close to RAPID. From this we can conclude that even for relay infrastructure, simple epidemic schemes like random provide excellent performance. This is because the increase in capacity of the network through additional infrastructure overwhelms the benefits of priority schemes to select packets for replication, as used by RAPID, over Random-epidemic.

Since most of our results are for low traffic loads, we also experimented with higher packet generation rates for RAPID and Random-epidemic. We found that increasing the packet generation load from 5 packets per hour per destination to 120 packets per hour per destination (a factor of 24) leads to a 29%, 12%, and 28% increase in average packet delivery delay for base stations, meshes, and relays, respectively. Moreover, even for higher loads the performance of Random-epidemic is close to RAPID. From the experiment, we conclude that the addition of infrastructure manages increasing network load well.

5.2 Analytical Model

Although our deployment study presents interesting trade-offs between the three type of infrastructure it suffers from the following limitation. The number of mobile nodes are fixed, hence we cannot validate our results for a large number of mobile and stationary nodes. To address this limitation, we develop detailed analytical models for large-scale networks in the presence of infrastructure.

We model the mobile network as $N + 1$ mobile nodes and M stationary nodes. The area covered by the mobile network is divided into k disjoint *regions*. We observed that $k = 2$ for our deployment. The mobile nodes spend time in each of the k regions. The stationary nodes can either be placed uniformly— every region has the same number of stationary nodes or non-uniformly— some regions have more nodes than other regions. The stationary nodes are placed uniformly within a region.

We assume that the *pairwise meeting times* between mobile nodes within region i are represented by exponentially distributed random variables with mean inter-meeting time $1/\beta_i$, and the *pairwise meeting times* between mobile and stationary nodes are represented by exponentially distributed random variables with mean inter-meeting time $1/\gamma_i$. The time at which mobile nodes move from region i to j are assumed to be represented by exponentially distributed random variables with mean rate μ_{ij}. The exponential assumptions follows from the observation made in our deployment regarding the mobile-to-mobile and mobile-stationary contacts (see Figure 5.1.1).

We assume a very general traffic model: *(i)* traffic in the network is unicast; *(ii)* traffic sources and destinations are uniformly random; and *(iii)* stationary nodes route data and can serve as sinks and not sources. For traffic destined to a sink, all base stations, one of the mesh access points, and one of the relays are connected to the Internet.

Our analytical model evaluates a general epidemic routing protocol where every node forwards packets to every other node it meets. Two nodes meet when they are within transmission range of each other. Every node has an large buffer and every transfer opportunity is long enough such that all packets in a node's buffer can be transferred to its peer. Hence, every packet can be considered independent of all other packets. Our analytical model differs from previous work in two major aspects. (1) Previous works use Markov chains to model infrastructure enhanced mobile networks [70, 79, 150]. The use of ODEs as fluid limits of Markov chains is limited to purely mobile networks [170]. (2) Prior work treats the entire network as one single region and models the inter-meeting times between peers as one exponential distribution [70, 79, 150] which we found yields inaccurate results for our mobile network.

5.2.1 Epidemic Spread

We model the spread of a packet and its replicas as an *epidemic infection* among nodes in the network. When nodes meet one another they exchange packets, infecting each other with the packets they possess until the packet infects the eventual destination.

The packet infection model consists of a system of non-linear differential equations with two variables for every region i: $x_i(t)$ and $y_i(t)$. The number of mobile nodes infected with a packet at time t within region i is $x_i(t)$, and the number of infected stationary nodes at time t within region i is $y_i(t)$. The network delivers a packet once the destination is infected with the packet, and our goal is to determine $P(t) = Pr[T < t]$, the probability that the time to deliver a packet is less than t. The expected delivery delay of a packet is given by $\int_0^\infty (1 - P(t))dt$.

Similar to our deployment study, our other goal is to estimate the amount of resources used in the network, including bandwidth, storage, and energy costs. All are strongly correlated with the number of transmissions/copies per packet. The expected number of copies per packet, $E[C]$, is given by the following. $E[C] = \int_0^\infty (\sum_{i=1}^{i=k}(x_i(t) + y_i(t)))dP(t)$. We assume in our analysis that all replicas of a packet are removed once the packet is delivered to its destination. In this section, we describe the set of differential equations for each of the three cases: relays, base stations, and meshes.

5.2.2 Relays

For peer-to-peer traffic, the set of non-linear differential equations governing the packet spread in the network is given by the following.

$$\begin{aligned}
x_i'(t) &= (\beta_i x_i(t) + \gamma_i y_i(t))(n_i(t) - x_i(t)) \\
&\quad - \sum_{\forall j \neq i} \mu_{ij} x_i(t) + \sum_{\forall j \neq i} \mu_{ji} x_j(t), \\
y_i'(t) &= \gamma_i x_i(t)(M_i - y_i(t)), \\
n_i'(t) &= \sum_{\forall j \neq i} \mu_{ji} n_j(t) - \sum_{\forall j \neq i} \mu_{ij} n_i(t), \\
x_1(0) = 1, &\quad \forall i \neq 1, x_i(0) = 0, \forall i, y_i(0) = 0, \\
&\quad n_1(0) = N + 1.
\end{aligned} \tag{5.1}$$

The rate of change of $P(t)$, $P'(t)$, is given by the following:

$$\begin{aligned}
P'(t) &= \frac{\sum_{i=1}^{i=k} x_i'(t)}{N}, \\
&\quad P(0) = 0.
\end{aligned} \tag{5.2}$$

The above set of differential equations assume that the destination is always a mobile node. However, in many cases data is destined towards a sink (e.g., the Internet). To account for node to sink traffic we assume that one of the relay nodes in region 1 is a sink. Such a node could be located close to an open access point and can act as a gateway to the Internet. We only need a minor modification to $P'(t)$ to account for the source-sink traffic. For packets destined to the sink, $P'(t)$ is the following.

$$\begin{aligned}
P'(t) &= \frac{y_1'(t)}{M_1}, \\
P(0) &= 0.
\end{aligned} \tag{5.3}$$

5.2.3 Base stations

We model the M base stations as one node. We assume that the pairwise meeting times between mobile nodes and the single node in region i is represented by exponentially distributed random variables with mean inter-meeting time $1/(M_i\gamma_i)$. For peer-to-peer traffic, the differential equations governing the dynamics of the packet spread in the network is given in Equation 5.4. The rate of change of $P(t)$ is given in Equation 5.5.

$$\begin{aligned}
x_i'(t) &= (\beta_i x_i(t) + \gamma_i M_i y(t))(n_i(t) - x_i(t)) \\
&\quad - \sum_{\forall j \neq i} \mu_{ij} x_i(t) + \sum_{\forall j \neq i} \mu_{ji} x_j(t), \\
y'(t) &= (\sum_{i=1}^{i=k} M_i \gamma_i x_i(t))(1 - y(t)), \\
n_i'(t) &= \sum_{\forall j \neq i} \mu_{ji} n_j(t) - \sum_{\forall j \neq i} \mu_{ij} n_i(t), \\
x_1(0) = 1, &\quad \forall i \neq 1, x_i(0) = 0, y(0) = 0 \\
&\quad n_1(0) = N + 1.
\end{aligned} \tag{5.4}$$

$$\begin{aligned}
P'(t) &= \frac{\sum_{i=1}^{i=k} x_i'(t)}{N}, \\
&\quad P(0) = 0.
\end{aligned} \tag{5.5}$$

For mobile node to sink traffic any base station can act as a sink since all of them are connected to a wired infrastructure. We make a simple modification to $P'(t)$ to accommodate traffic to the sink.

$$\begin{aligned} P'(t) &= y'(t), \\ P(0) &= 0. \end{aligned} \tag{5.6}$$

5.2.4 Mesh

We can model the wireless mesh as a special case of a relay network. The difference between a relay network and a mesh network is that once a mesh node is infected by the packet, all nodes in a mesh can be infected by the packet using the mesh radio. This rate of infection among mesh nodes, denoted by α, depends on the bit rate of the mesh radio and the number of packets transferred simultaneously over a mesh link. The ODEs governing the rate of change of $x_i(t)$, $P(t)$, and $n_i(t)$ are the same as the relay case. The difference lies in the differential equations governing the rate of infection among the stationary mesh nodes in region i ($y_i'(t)$).

$$y_i'(t) = \gamma_i x_i(t)(M_i - y_i(t)) + f(\alpha, y_1(t), ..., y_k(t), M_i) \tag{5.7}$$

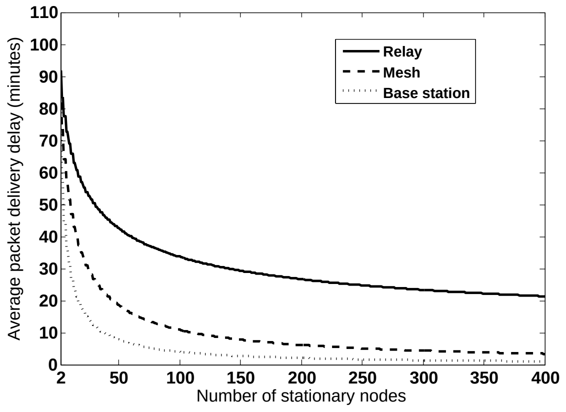
The function f depends on the topology of the mesh. For example, if we consider a clique, f is the following.

$$f(\alpha, y_1(t), ..., y_k(t), M_i) = \alpha(\sum_{i=1}^{i=k} y_i(t))(M_i - y_i(t)) \tag{5.8}$$

The above definition of f means that the rate of infection among mesh nodes in region i is equal to the rate at which the infected mesh nodes (in the entire network) infect the nodes in region i.

5.2.5 Impact of Network size

We validated the results from the model by comparing it with results from our simulation on data from the deployment—please refer to the associated research publication [23]. We



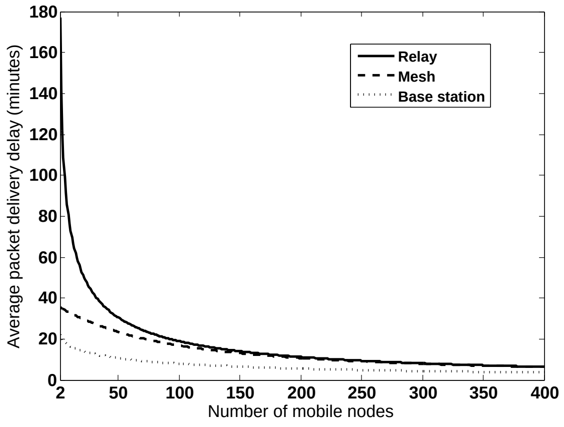


Figure 5.7. The average packet delivery delay as a function of the number of stationary nodes and mobile nodes. The number of mobile nodes is fixed at 20 for the first experiment and the number of stationary nodes are fixed at 25. The network parameters are taken from our deployment.

then use the model to evaluate the performance of hybrid networks with large numbers of stationary and mobile nodes. We perform two experiments with the model. We fix the number of mobile nodes in the first experiment to 20 and vary the number of stationary nodes. In the second experiment the number of stationary nodes is fixed at 25 and we vary the number of mobile nodes. The model parameters are taken from the uniform placement of stationary nodes in our deployment and the fraction of traffic to a sink is taken as one-half.

Figures 5.7 shows the average packet delivery delay for the first and second experiment. From the figures, we draw the following conclusions. (1) With a very large number of stationary nodes, we obtain very low delays. However, if we desire delays of the order of seconds (e.g. for an instant messaging application) we need a very large number of base station nodes. (2) To reduce packet delivery delay by a factor of two we need around 5–7 times as many relays and 2–3 times as many mesh nodes as base stations. (3) Adding mobile nodes to a network produces substantial improvement in performance of a relay infrastructure but only a small improvement in the performance of either a mesh or base station infrastructure. This result confirms that mobile-to-mobile routing leads to marginal improvement in the presence of base stations or a mesh.

5.3 Discussion

This chapter presented several interesting trade-offs between base station, mesh, and relay hybrid networks. We found that while base stations provide better performance in terms of packet delivery delay, their high costs often makes mesh and relay networks a better choice. However, base station-enhanced mobile networks with a large number of stationary nodes is required to support applications such as instant messaging which can tolerate latency of the order of seconds. Moreover, we saw that addition of infrastructure often obviates the need for mobile-to-mobile routing. However, there is an important aspect of relay and mesh node design which we ignored in our analysis. We assumed that mesh and relay nodes are solar powered yet always available. This assumption is true only if we have intelligent *power management* on relay nodes which provide maximal performance and *near-perpetual* operation using energy scavenged from the sun. In the next chapter, we use the idea of Hierarchical Power Management (described in Chapter 4) to design a multi-tiered, scalable solar powered relay node called Throwbox.

CHAPTER 6

ENHANCING SPARSE MOBILE NETWORKS WITH THROWBOXES

We showed in the previous chapter that the addition of stationary resources in the form of base stations, relays, and mesh nodes can improve the performance of mobile networks. Especially in sparse disruption-tolerant mobile networks [82, 147], addition of stationary resources in the form of nomadic relays can substantially improve performance. However, building and deploying relay nodes in mobile networks face unique system challenges. For example, one of the chief impediments to the deployment and use of relays for enhancing mobile networks is energy management on the stationary nodes. To understand why consider this: for relay nodes to be maximally effective they may have to be placed *anywhere* in the network. Such a nomadic placement requires solar or battery powered devices. Hence, the effectiveness of these untethered nodes without intelligent energy management is minimal. Therefore, the actual design of these relays opens up several interesting networking and embedded systems questions. For instance, what is the ideal hardware platform for such relays? How do we manage energy on these nodes?

The high power consumption of always-on nodes results in a shorter lifetime; conversely nodes powering on and off intermittently without regard to node mobility may miss numerous connection opportunities, increasing lifetime without necessarily improving performance. A key goal is to provide energy efficiency without sacrificing network performance. Moreover, the relays are constrained to consume power at a rate dictated by a lifetime requirement or the availability of scavenged energy, such as solar power. In this chapter we address these issues in the context of the following challenge: *how can a relay in a mobile network meet*

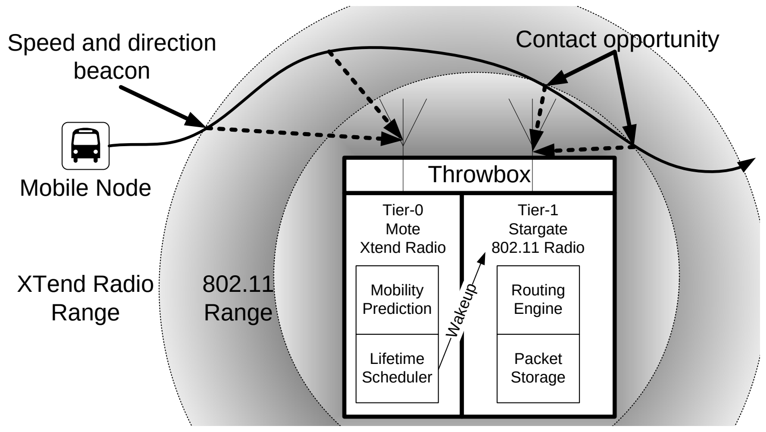


Figure 6.1. Overview of our Throwbox architecture.

an average power constraint and simultaneously maximize the number of packets forwarded by it?

Our approach is a new relay architecture called *Throwbox*. Throwboxes use a novel paradigm for energy management in mobile DTNs that provides more efficient neighbor discovery by detecting the mobility of other nodes at a minimum cost and predicting the cost and opportunity of each possible contact. Using these predictions, the throwbox can intelligently choose the most fruitful contact opportunities, and can limit the number of opportunities to meet energy constraints. In our architecture, we pair a *tier-1* platform—a PDA-like Stargate and 802.11 radio—with a low-power *tier-0* platform—a Mote and Digi-XTend radio. The XTend radio is a *long-range, low-bitrate* radio for neighbor and mobility discovery. It has a range of about a mile which is more than ten times the range of WiFi radios. By coupling the XTend radio with a low-power Mote, we reduce the total energy cost of the throwbox platform. The system builds on ideas from other multi-radio systems such as CoolSpots [127] and Wake-on-Wireless [138].

Figure 6.1 presents an overview of our approach and the problems we address in this chapter. Mobile nodes (shown as a bus in the diagram) beacon their position, direction, and speed using the long-distance radio. While listening for other nodes, the throwbox needs very little computational power and memory, and the tier-0 platform is sufficient to process the beacons. If a throwbox hears a beacon, tier-0 predicts if, when, and for how long the mobile node will come in contact with the tier-1, 802.11 radio. Using a constrained single-objective optimization to meet a power consumption target, the node decides whether to take the transfer opportunity. If it does, tier-0 wakes the Stargate and WiFi radio in advance of the mobile node entering radio range, which then transfers DTN data. After the contact opportunity, the tier-1 platform returns to a sleep state. For this scenario, we develop a method for predicting the contact opportunities based on the observed mobility of remote nodes (§ 6.1) and an algorithm for choosing which contacts to take (§ 6.3).

6.1 Mobility Prediction Engine

If a mobile network is sparse (e.g., TurtleNet [147], ZebraNet [168]), throwboxes must expend significant energy searching for contact opportunities. Because of node mobility, throwboxes have a limited amount of time to exchange data with peers. Searching for contacts efficiently requires waking and sleeping rapidly (called *duty cycling*), and conversely, data transfer requires high-bandwidth connections. Our design philosophy is to separate neighbor discovery from data transfer, and to divide these tasks across the hardware that is best suited to each task. The long-distance radio and tier-0 platform can duty-cycle, listen, and idle efficiently, while the high-bandwidth radio and tier-1 platform can transfer large amounts of data during limited connection opportunities. While the tier-0 platform is listening for contacts, the tier-1 platform remains asleep.

The decision to wake the tier-1 platform and radio depends on an engine that detects, and predicts the mobility of nodes in the network, based on information gathered by the tier-0 platform and radio. By predicting the trajectory of an on-coming mobile node, the tier-0 platform can determine if, when, and for how long it will enter the range of the data transfer radio. If the throwbox determines that the mobile node will not come in contact with its data transfer radio, it can safely ignore that node. If it determines that the mobile node will enter at a certain time, it must make sure that the throwbox can make the most

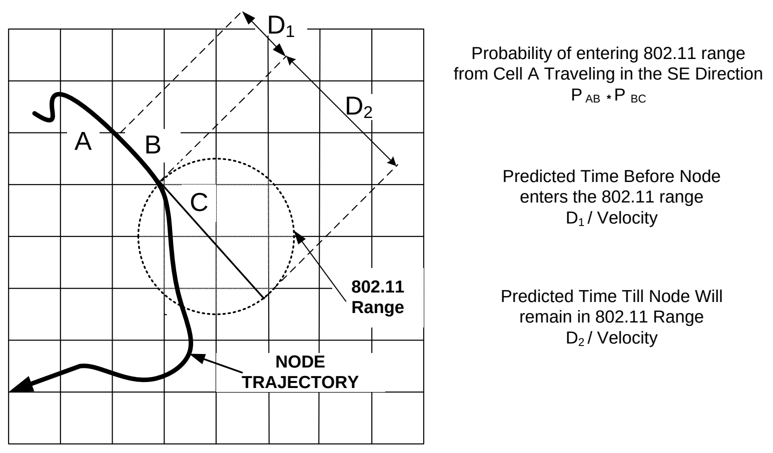

D1
D2
A
B
C
802.11 Range
NODE TRAJECTORY
Probability of entering 802.11 range from Cell A Traveling in the SE Direction $P_{AB} * P_{BC}$
Predicted Time Before Node enters the 802.11 range D_1 / Velocity
Predicted Time Till Node Will remain in 802.11 Range D_2 / Velocity

Figure 6.2. The figure depicts the working of the Mobility Prediction Engine

of the contact opportunity. When the throwbox estimates the time that the mobile node will enter 802.11 range, it wakes the tier-1 node in advance of the mobile node arriving—a process that can take several seconds. The predicted length of the contact opportunity is then used to determine which contact opportunities to take.

To determine trajectories, we require mobile nodes to periodically transmit location, speed, and direction over the long-distance radio as a beaconing message.

6.1.1 Prediction Algorithm

Mobility prediction algorithms in the past have been used for predicting future network topology [153], accurately predicting hand-offs and bandwidth provisioning in cellular networks [43, 142] and location tracking in wireless ATM networks [99]. Such algorithms either require coordinated information from more than one base station [142, 144], or they are computationally too expensive to implement on a resource constrained Mote like device [144], or are built assuming very specific mobility patterns of nodes [142]. We want an uncoordinated, base station independent mobility prediction engine which works on constrained platforms like a Mote. Therefore, unlike previous work, we present a lightweight simple mobility prediction engine.

We model the movement pattern of the mobile nodes as a Markovian process. Hence, we assume that the location of a node at time t_i depends on its location at time t_{i-1}. The algorithm models the problem as a virtual square of width $2r$, where r is the radius of the long-range radio. The throwbox is located at the center of the square in Figure 6.2. The large square is divided into square cells numbered from 0 to k. The algorithm makes use of a probability transition matrix T for mobile nodes in the network. Entry T_{ij} of the matrix is the probability that a node will transition to cell j given that it is in cell i.

Mobile nodes beacon data in tuples of the form $< p, v, t, d >$, where p is the current GPS location of the mobile node, v is its speed, t is the time at which the beacon was sent, and d is the direction of motion of the node. The direction of motion is determined using a GPS. When a node n approaches, the throwboxes collect a set of tuples $\{< p_1, v_1, t_1, d_1 >, ..., < p_k, v_k, t_k, d_k >\}$ over time. Using this set, as well as historic information, the algorithm estimates :

- The probability that the mobile node n would be in any cell within the data transfer radio range after time ΔT, where ΔT is the time required to wakeup tier-1 and start transferring application data. The time to wakeup tier-1 can be typically on the order of seconds

- The predicted time when the node will be in range of the data transfer radio and, subsequently, the amount of data it is likely to transfer

Figure 6.2 illustrates this process. At time $t = 0$, the mobile node's beacon from cell A is received. By dead-reckoning along a straight line of motion, the prediction engine estimates D_1, the distance until the node is within 802.11 range, and D_2, the distance for which the node would be within range. Accordingly, the predicted length of time until the mobile node reaches the range of the data transfer radio is D_1/v; if this time is greater than the transition time to wakeup tier-1, then the mobile node is ignored. Otherwise, the throwbox calculates P$_r$, the probability that the node will enter the range of the data transfer radio. In the simple example presented in the figure, Pr is the product of the probability of transition from cell A to B and from B to C; i.e., $\mathrm{P}_r = T_{AB} \cdot T_{BC}$. The expected duration of time the mobile node will stay in data radio range is $Pr \cdot (D_2/v)$. Below, we show how we calculate Pr under Markovian assumptions.

6.1.2 Estimating the Probability of Entering Data Radio Range

The prediction engine assumes that the node moves in the direction that it was last moving in. Therefore, it constructs a straight line in the current direction of motion and finds the intersection with the range of the data-transfer radio range, idealized as a circle. Let p be the cell at which the node enters the data radio range; let D_{p,p_k} be the Euclidean distance between the present location of the node and the location of the point where it enters the data transfer radio range (assuming the present direction of motion of the node). Let v_k be the velocity of node n. Let $c_i, \ldots, c_{j+i}$ be a sequence of cells in the predicted path of motion of the node. Let $\{d_1, \ldots, d_i\}$ be the set of i possible directions of motion. For example, a direction could be north, south, east or west. The probability that a node enters the data transfer radio range at position p after time ΔT is given by:

$$Pr[X_{t_k+\Delta T} = p | (X_{t_k} = p_k, d_{t_k} = d_k), \ldots, (X_1 = p_1, d_{t_1} = d_1)],$$

where X_t is the position of the node at time t. The above probability can be approximated as

$$P_{appr} = Pr[X_{t_k+\Delta T} = p | X_{t_k} = p_k, d_{t_k} = d_k] \tag{6.1}$$

assuming that the node movement is modeled as a Markovian process. P_{appr} is evaluated by the algorithm as

$$P_{appr} = \begin{cases} T_{c_i,c_{i+1}} \cdot \ldots \cdot T_{c_{i+j-1}c_{i+j}}, & \frac{D_{p,p_k}}{v_k} \leq \Delta T \\ 0, & \frac{D_{p,p_k}}{v_k} > \Delta T. \end{cases} \tag{6.2}$$

Equation 6.2 shows that the probability of a node entering a cell of the data transfer radio after time ΔT equals the transition probability of the node from its present cell to cell c_{i+j} provided the node is traveling fast enough to cover the shortest distance between the two cells in time ΔT. Although we assume that that the node is going to continue movement in the direction that it was last seen, this information is periodically recalculated with every incoming beacon. Hence, if the beaconing rate is fast enough (once per second in our prototype implementation) the actual trajectory of the mobile node is approximated as a combination of piecewise linear functions.

The transition probability T_{ij} is learned/updated by each throwbox node using historical data. Every time a mobile node passes through the long range radio's range the beacons captured by the throwbox is used to build the transition probability matrix. If a throwbox finds that a mobile node in a path of motion has moved from cell A to B, it updates the transition probability entries P_{Ai}, where i represents the eight neighboring cells of A including B.

We also derive the running time and space complexity bounds for the prediction algorithm—please refer to the associated research paper [22].

We also note that while searching for mobile nodes, the long-distance radio does not need to be constantly powered; it needs only to wake-up often enough to predict if, when, and for how long the mobile node will be within range of the data transfer radio. Our design uses a Digi-XTend radio that support several very efficient duty-cycling modes built into the hardware and the MAC layer. We have implemented an adaptive controller for duty cycling the radio which we describe in the next section.

6.2 Discovery Radio Duty-Cycle Controller

The radio used in our design has k software configurable cyclic sleep modes $\{S_1, S_2, ..., S_k\}$. In sleep mode S_k, the radio wakes up every t_k seconds for time τ ($\tau << t_k$) looking for RF activity. If any RF activity is discovered and the packet received has a special header, the radio is switched on to receive the entire packet.

We use these cyclic sleep modes to efficiently duty-cycle the Digi-XTend radio. If the mobility prediction engine requires m data points to accurately predict the movement pattern of a node, the aim of the heuristic is to gather these m points at minimal energy cost. The number of points the mobility predictor needs must be determined through experimentation. In our experience we found that a small number of points, such as four, is sufficient to make accurate predictions in our network.

The algorithm constructs m virtual concentric circles of radii $r_n = R + \frac{n \cdot (r-R)}{m}$ $(1 \leq n \leq m)$, where r and R are the ranges of the discovery radio and the data transfer radio. The number of circles, m, constructed depends on the number of data points required by the mobility prediction engine to accurately predict when the node will enter the data transfer radio range. The distance $r_1 - R$ is such that the time before a node enters the data transfer range is less than the time to wakeup tier-1 from suspension or shutdown. The virtual concentric circles help the throwbox determine how close it is to a mobile node. The sleep mode S_k corresponding to the shortest sleep interval is taken as the default sleep mode. Therefore, when there are no nodes in range the radio is duty-cycled in sleep mode S_k. When the throwbox receives a beacon from a mobile node it determines the circles of radii r_i and r_{i-1} within which the node is presently located. The algorithm uses the velocity $\overline{v}$ to find the sleep mode S_j with the longest t_j such that $t_j \leq (r_i - r_{i-1})/\overline{v}$. Consequently, the duty-cycle controller puts the discovery radio to sleep in sleep mode S_j. If the discovery radio does not find any data after time t_k, the node is put to sleep again in the default sleep mode. Assuming that the variance in the speed of the mobile nodes is not very large, the algorithm would not miss many beacons on the long range radio and would remain asleep as long as possible. If the throwbox records beacons from multiple nodes, the heuristic calculates the shortest time after which a node will enter the adjacent concentric circle.

6.3 Token Bucket Lifetime Scheduler

Mobile nodes may present a Throwbox with many transfer opportunities. However, because they have limited energy, they may not participate in every opportunity. Additionally, mobile nodes may only skirt the outside range of the data transfer radio, and by the time the tier-1 platform is awake, the contact may have moved out of range.

A limited energy supply can be viewed as a constraint on the *average power* that the system consumes in two different scenarios. First, the throwbox may be designed to last for a certain period of time—computing the average power from the capacity of the batteries is straightforward. Second, throwboxes may be capable of scavenging energy, such as the solar panels used in our prototype system. In that case, we assume the average power constraint to be the average power produced by the solar cells. This is a simplification, as there is great variance in radiant solar energy and the battery must be large enough to smooth fluctuations.

We show that computing the optimal subset of opportunities to participate in given an energy budget is NP-Hard (see the related publication [22]). Here we present our sub-optimal solution, which is linear in the number of connections and requires constant memory. Note that optimization criteria other than energy can also be considered concurrently. For example, mobile nodes may transmit the priority of packets that they are carrying, or the throwbox can decide which opportunities to take based on the likelihood of being able to route the packet to its final destination. However, we leave this extension to future work (see Chapter 8).

6.3.0.1 A Token-Bucket Approach

An approximate online solution to choosing the appropriate set of connection events would follow the following intuition: since an online algorithm has no knowledge of the future, at any time t it should take a contact opportunity only if the energy cost does not lead to a violation of the average power constraint till time t. In other words, the heuristic should regulate energy flow based on the average power constraint. Moreover, since contact opportunities may occur in bursts, the scheduler should also be capable of handling bursts of energy consumption.

Token buckets have been used in a similar scenario in networking to regulate network traffic [154]. Token buckets allow bursty traffic to continue transmitting while there are tokens in the bucket, up to a user-configurable threshold thereby accommodating traffic flows with bursty characteristics [56].

Algorithm 1 Token Bucket Scheduler

```
Average power constraint = $P$
Generate energy tokens at the rate of $P$/sec
Energy cost of present connection event = $E_p$
Size of present connection event = $S_p$
Number of tokens accumulated till present = $m$
if $m \cdot P < E_p$ then
    Do not wakeup Tier-1
else
    Mean size of the last $k$ connection events = $S_k$
    Mean energy of the last $k$ connection events = $E_k$
    Mean inter-arrival time of connection events = $T_m$
    if $m \cdot P - E_p + T_m \cdot P < E_k$ and $S_p < S_k$ then
        Do not wakeup Tier-1
    end if
end if
```

Our scheme, shown in Algorithm 1, is based on a token bucket scheduler. It generates energy tokens at the rate of the average power constraint. For any given connection event, if the number of tokens accumulated is less than the amount of energy required, the event is ignored. However, if the number of tokens accumulated is greater than the energy required for a connection event the system decides between the following simple choices: should it take this connection event, the next connection event, or both? First, the algorithm estimates the size and energy cost of the current connection event based on the mobility prediction engine. It then estimates the size and energy cost of the next connection event as the mean of the last k connection events and assumes that the connection event arrives at the mean inter-arrival time of the last k connections. Taking into account the number of tokens that would accumulate between now and the next connection event, if it estimates that it can take

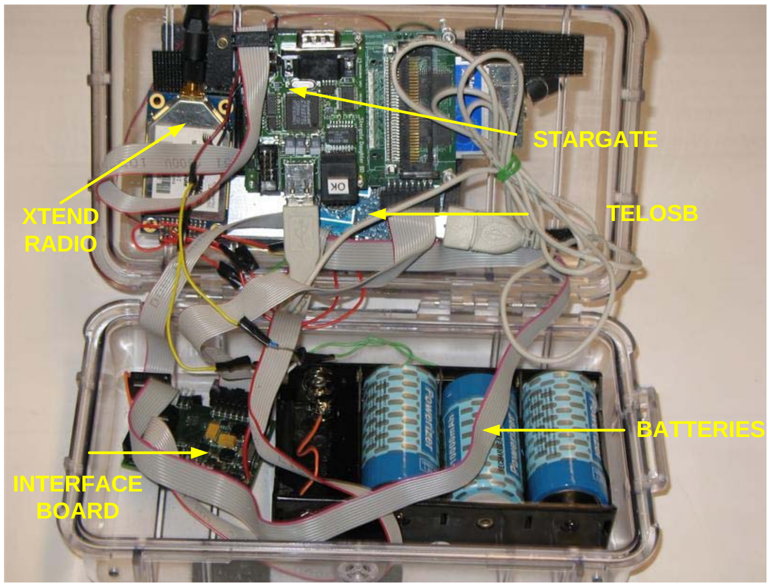


Figure 6.3. Prototype for a Throwbox node

Characteristic	XTend	Dlink-Air
Ranges	1000 m	150 m
Receive Power	360 mW	1000 mW
Transmit Power (max)	1 W	1.2 W
Sleep Power	10 mW	50 mW

Table 6.1. Characteristics of the two radios

both connection events it takes the current one. Otherwise it chooses the larger of the two. This process repeats for the next contact opportunity.

The algorithm runs in time linear in the number of connection events and requires $O(1)$ space. Therefore, the algorithm is space and time efficient and is easily implementable on a low power tier-0 platform.

6.4 Throwbox Prototype Implementation and Deployment

We constructed prototype throwboxes, shown in Figure 6.3, using a Crossbow Stargate (tier-1) [160] and a TelosB Mote (tier-0) [129]. We chose these hardware platforms because they handle the two mobile DTN activities, data transfer and neighbor discovery, efficiently.

The Stargate platform runs Linux, allowing us to run the same Java-based routing software used in our mobile testbed [33] (see Chapter 3).

The Stargate has a 32-bit, 400 MHz PXA255 XScale processor, 64 MB of RAM, 32 MB of internal flash, and a DLink-Air 802.11b interface. The TelosB Mote has a 8-bit, 8 MHz micro-controller, 10 KB of RAM, and 1 MB of external flash. The Mote is attached to a Digi-XTend 900 MHz OEM module. The prototype also employs two 5V PowerFilm solar panel as additional energy source. The characteristics of the two radios are summarized in Table 6.1.

We fabricated a power supply board for attaching both platforms to a single battery and support charging from solar energy. This board provides an additional hardware element necessary to throwboxes, a Maxim DS2770 fuel gauge chip. The fuel gauge chip provides accurate data on the amount of energy left in the battery, similar to those found in laptops. This accounts for energy consumed by the platforms and radio, as well as energy produced by the solar panels. The TelosB Mote reads this value periodically and corrects the token bucket to account for the actual energy used and produced.

To support a real-world test of the throwbox, we used the mobile component of our DOME testbed, UMass DieselNet [33] (described in Chapter 3). We deployed three always-on throwbox prototypes at fixed locations on DieselNet bus routes and collected data for a period of three weeks. The throwboxes were placed according to a placement algorithm in previous work [174].

6.5 Evaluation

We evaluated the Throwbox system using two techniques: trace-driven simulations and prototype experimentation. The simulations use traces collected from three always-on throwboxes placed in DieselNet for three weeks, changing batteries manually. These prototypes logged connection events. In the simulations, we compare our system with the following systems.

- Optimal (Dual platform): The optimal system uses the two-radio, two-platform system, but with a perfect mobility prediction algorithm and an optimal scheduler. The optimal system assumes knowledge of the exact time and length of every contact opportunity. The system uses a dynamic programming algorithm for the knapsack problem to select the exact set of connection events that maximizes throughput while meeting an energy constraint [45].

- PSM*: This system is a single tiered single radio system that periodically powers off its wireless interface to save energy. This is similar to PSM (Power Saving Modes) used on WiFi cards. PSM* wakes up its wireless interface and scans for connection events and goes back to sleep if it finds none. When the WiFi card is switched off, the platform is in its idle state. We exhaustively searched the state space to set parameters that use the minimum energy with equivalent data transferred. This provides a comparison to the best system that does not use any extra hardware.

- WoW*: This system is adapted from Wake-on-Wireless [138] which uses a second radio as a discovery radio. The published WoW system always wakes up when it sees data addressed to it on the discovery radio and it is assumes that the discovery radio has the same range as the data transfer radio. For fair comparison, we adapt WoW to a DTN environment. The WoW* system uses our mobility prediction engine to intelligently decide when to wakeup the data transfer radio. Without this modification, the WoW* system would wakeup the higher power tier on every spurious contact over the long-range radio. However, it does not use the scheduling algorithm to decide which opportunities to take, nor does it duty-cycle the long-range radio.

- Always-on: This throwbox remains on all of the time, and thus does not use the second radio, and it is not penalized by the additional energy needed for the long-distance radio and Mote.

- No throwbox: Without a throwbox the DTN delivers data normally.

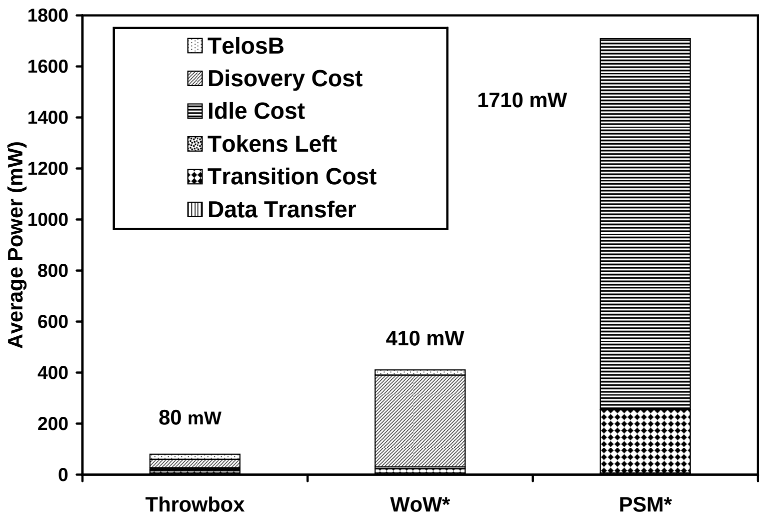


Figure 6.4. The breakup of the energy consumed by different components for the Throwbox, PSM*, and WoW* systems.

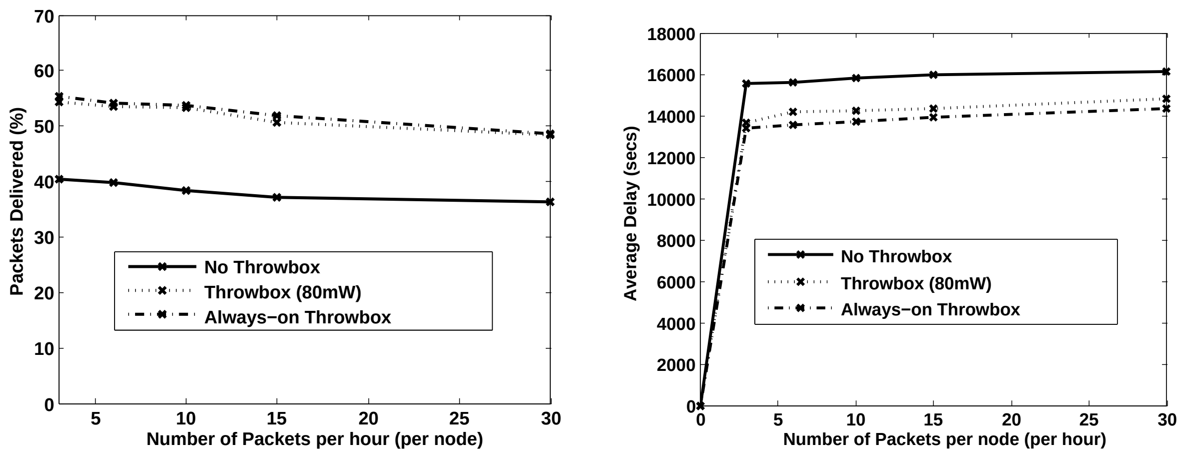


Figure 6.5. The improvement in the delivery rate and the latency of packets transferred using a Throwbox with a 80mW power constraint

6.5.1 Trace-Driven Simulation Results

System Power Consumption

We first present the energy consumption for different systems when they forward the same amount of data. Figure 6.4 shows the average power of each system broken down by their component power consumption. We have divided the bars into the following parts:

energy for useful work (Data Transfer), the number of tokens left over by the scheduler at the end of the simulation (Tokens Left), the energy used by the long distance radio (Discovery Radio), the cost of turning the tier-1 system on (Transition Cost), the cost of idling tier-1 while it is searching for contacts (Idle Cost), and the power consumed by the tier-0, Mote system (TelosB).

The power consumption of the PSM* system plainly illustrates the value of using the tier-0 platform and radio. PSM* devotes 99.5% of its energy turning the platform on and off and idling while searching for contacts. The WoW* system virtually eliminates this idle cost as it only turns on the tier-1 system when there is a valid contact. However, it does devote a large amount of energy to the long-distance radio as it does not use the duty-cycling employed by the Throwbox architecture. Additionally, the WoW* system suffers from a high transition overhead as it takes all contacts—short and long—and hence incurs a overall high transition energy cost.

Average Power Constraint Versus Network Performance Boost

We next study the performance boost provided by placing a single Throwbox in our mobile testbed with a given power constraint. We use the MaxProp [33] routing protocol to exchange data among nodes in the network. Each node uses a 1GB buffer and the packets transferred were 1KB in size. We varied the number of packets generated per node per hour and calculated the delivery rate and average latency of delivered packets. We ran the experiments for three weeks of simulated time—the total period of our deployment. Figure 6.5 shows the improvement in packet delivery percentage and decrease in packet delivery time with a single Throwbox deployed with an average power constraint of 80 mW.

From the results in figures, we see that a Throwbox with an average power constraint of 80 mW performs close to an always-on Throwbox and leads to an increase in packet delivery percentage of more than 37% and decrease of at least 10% in the packet delivery time. When compared to an always-on throwbox, the power constraint of 80 mW yields a

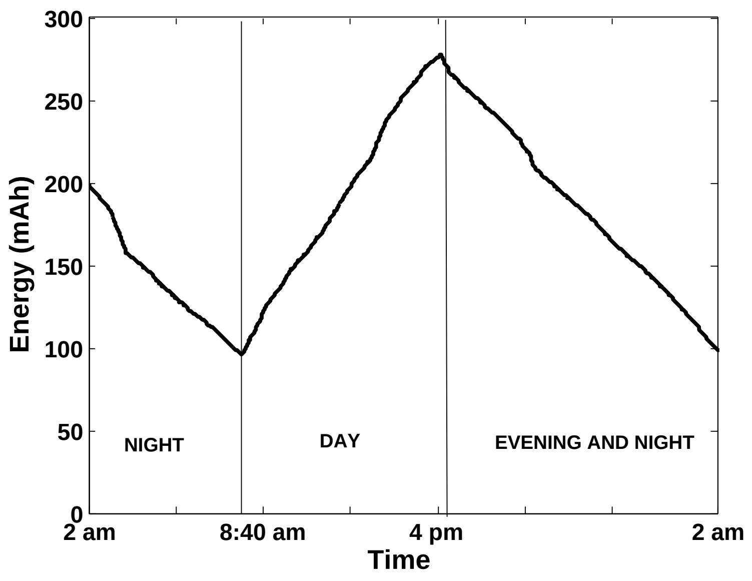


Figure 6.6. The consumption of energy for the throwbox prototype over a period 24 hours

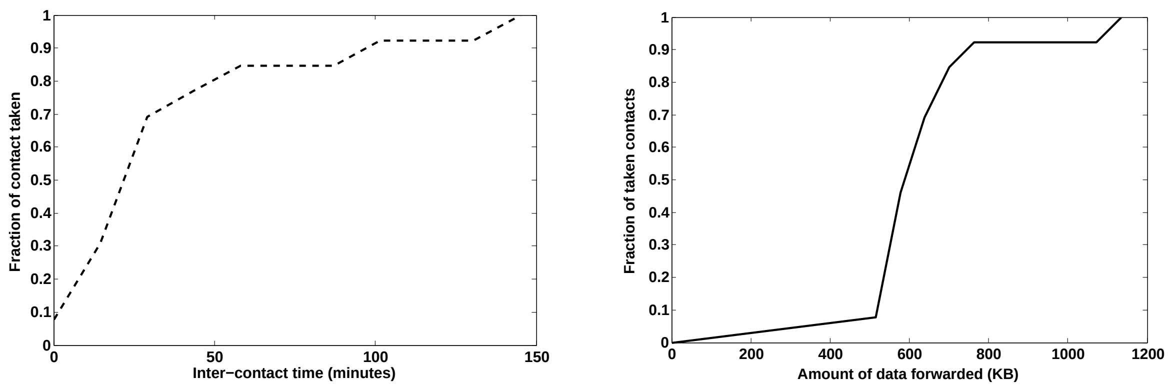


Figure 6.7. The distribution of the amount of data transferred during contacts and the inter-contact time for the Throwbox prototype.

system that can last 31 times longer on the same battery while delivering almost as many packets.

6.5.2 Prototype Evaluation

We deployed a two-platform, two-radio prototype with a power constraint of 80 mW using the mobility prediction, scheduling, and duty-cycling algorithms described in this

Chapter. The system was equipped with solar panels 220 cm^2 in area and a 1 Ah battery that was 20% full. The box was deployed in our mobile testbed for a day and it logged the capacity remaining in the battery as a function of time. The results of the experiments are shown in Figure 6.6 , Figure 6.7. In Figure 6.6, we see that the Throwbox stores excess energy during the day and during nighttime it spends the excess accumulated energy. We determined through this experiment that the solar panels produce an average of 65 mW over 24 hours. This is about 15 mW less than the amount of power we predicted using the data sheets from the solar panel. During the period of the entire day, the throwbox was able to transfer 8.3 MBytes of data with bus nodes. On an average, when an Always-on throwbox was deployed in our testbed, it could transfer 22 MBytes of data with the bus nodes. Therefore, the throwboxes at 65 mW power consumption can transfer one-third the amount of data compared to the best case. Figure 6.7 shows the distribution of the time in between contacts taken by the throwbox prototype. We find that in most cases the inter-contact time is less than one hour. This shows that the Throwbox's scheduling algorithm is able to take bursts of contacts successfully. Moreover, the throwbox takes contacts equally during the day (when energy is being scavenged from the sun) and during the night (when it uses the stored energy in the battery). The same figure shows the distribution of the amount of data transferred during contacts. From the figure, we find that the Throwbox always takes contacts which are long and where substantial amount of data can be transferred.

6.6 Discussion

In this chapter we presented a hardware and software architecture of solar powered Throwboxes. Through extensive trace-driven simulations and deployment evaluation, we showed that Throwboxes can operate near-perpetually while substantially improving network performance. However, since Throwbox relay can only create virtual contact opportunities between mobile nodes by storing and forwarding data for mobile nodes, they can only enhance sparse disruption tolerant mobile networks where traffic is peer-to-peer [33, 82].

However, in urban and semi-urban regions where the primary destination from mobile nodes is an Internet sink, we require a radically different solution. In the next chapter, we briefly outline the challenges with connectivity in open WiFi based mobile networks and present a solution which uses additional mesh nodes to provide enhanced connectivity to mobile users.

CHAPTER 7
ENHANCING WIFI MOBILE NETWORKS

Throwboxes can enhance sparse mobile networks. They are especially useful when traffic is peer-to-peer [82, 147]. However, in urban and semi-urban areas, mobile users are most concerned with transferring data to an Internet server. For example, popular mobile phone applications such as email, web browsing, and instant messaging require an active Internet connection.

There are several methods of providing reliable, ubiquitous connectivity for mobile devices in urban and semi-urban areas. Cellular deployments, including 3G, EVDO, and GPRS, offer commercial, fee-based coverage of large areas. Unfortunately, such infrastructure is costly and difficult to manage, and its installation and operation is reserved for a handful of large carriers. And due to the costs involved—a recurring fee of about US$50 per month per device—its use is limited to persons that can afford it, and usually for only one mobile device.

At the same time, open WiFi access points (APs) are gaining in popularity [136] and are present in cities large [54, 136] and small [20]. The major advantage of these organically deployed APs is cost — while 3G price plans will vary, open WiFi is by definition free to mobile users. Accordingly, open WiFi removes a significant impediment to pervasive computing. Figure 7.1 shows the availability of open WiFi APs in a section of our city. From Aug 07–Oct 08, at least 75% of 62500 regions each 0.01 km^2, supported open WiFi Internet access.

The disadvantage of WiFi access is robustness. Although WiFi links can have higher peak downstream bandwidths than 3G, it is a shorter-range radio, which leads to both coverage

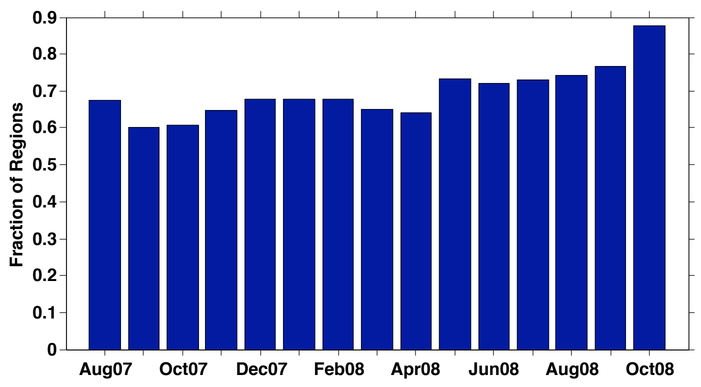


Figure 7.1. The fraction of 100x100 m^2 regions in our city where vehicles have open WiFi access points.

holes and areas of high loss rates, even in networks planned for blanket coverage [18, 107]. In networks where mobile users are subject to longer periods without connectivity, a myriad of ad hoc and disruption tolerant networking (DTN) protocols have been proposed that leverage the mobile nodes to deliver data [55]. While ad doc networks and DTNs provide connectivity where there was none, delivery delays depend on the mobility of other users. Typically, popular interactive, delay-sensitive applications cannot be reliably supported.

In this chapter we first briefly analyze the problems with open WiFi networks. We show that such networks are rampant with coverage holes which can have an adverse effect on the performance of network transport. Next, we propose a system called *Epsilon* which uses a novel approach of placing very low-bandwidth, long-range radios wherever holes are present. The low-rate backbone acts as a bridge to a 802.11 AP. Connections from the mobile user are striped across both channels to smooth hand-off. Combined with an Internet proxy, clients can then maintain TCP connections across open APs, making applications more predictable. Because the radios are longer range than WiFi, fewer devices and a smaller cost is required to cover a large area. We show that while existing solutions [54] can increase TCP throughput by a factor of 2x when a node is associated to an access point

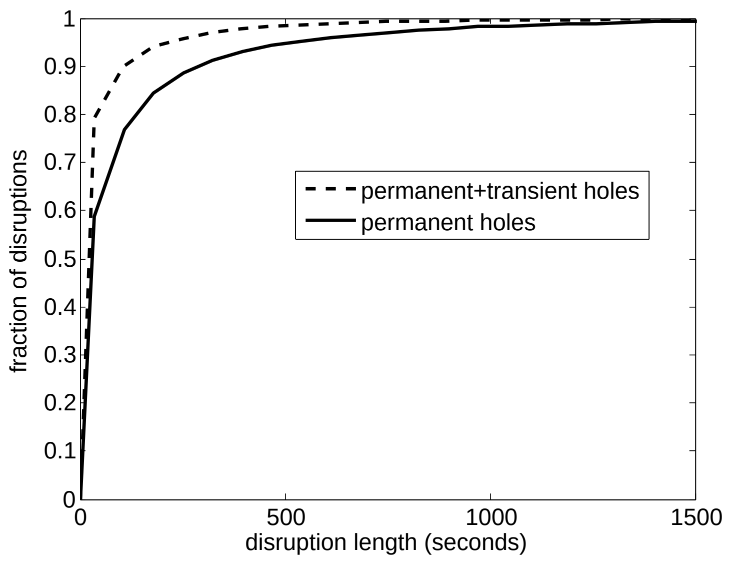


Figure 7.2. The cumulative distribution function of the disruption lengths observed by buses (from a moving node) in DOME.

(effectively a gain of 15% when there are disruptions), using Epsilon, TCP throughput can increase for mobile users by 1.2x to 13.0x.

7.1 Coverage Holes: Presence and Impact

To analyze the performance of mobile systems in open WiFi networks, we conducted a measurement study of disruptions in our mobile testbed (see Chapter 3). We performed both link-layer and application layer connectivity measurements from buses using open WiFi [148]. We found that coverage holes are rampant even in WiFi networks planned for blanket coverage. Consider Figure 7.2 as an example. The figure shows the CDF of the duration of coverage holes as observed by buses in DOME. Permanent holes correspond to areas where there is no AP coverage over a period of time and transient holes are areas where there is access point coverage but there is no connectivity due to transient factors such as failure to obtain DHCP leases or other factors such as interference and obstacles. The absolute lengths of the holes is high: the median length is 50 seconds with a 90th

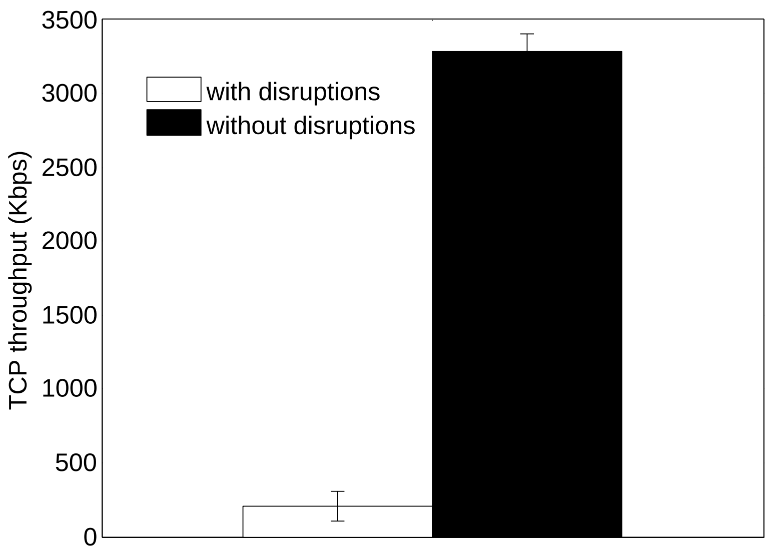


Figure 7.3. The decrease in TCP throughput for four minute TCP sessions in Measurement Set I. For this figure the disruptions correspond to times in between associations (successful or otherwise)

percentile of 500 seconds. This figure clearly indicates the presence of disruptions—large and small—in WiFi based mobile networks.

TCP and UDP streams suffer differently from disruptions in connectivity. Because of the TCP congestion control mechanism, a mild disruption can engage slow start, strangling throughput. UDP simply suffers the minimum of the available capacity and offered packet rate. Here, we quantify the loss of throughput for TCP based on traces of mobility and coverage holes in our environment. We performed a series of trace-driven experiments using traces collected from DOME. To provide repeatable experimentation we conducted the experiments in an indoor environment with a stationary node associated to a WiFi AP. We implemented a proxy at the client that uses `ip_queue` (an `ip_table` utility to queue packets in the linux kernel) to drop packets whenever the trace recorded a disruption. We then started a large TCP transfer to download data from a host on the network.

The results of the experiment are shown in Figure 7.3. As a point of reference, we plot the bars corresponding to TCP performance when there are no disruptions. We see that due to constant TCP timeouts and congestion control management TCP throughput decreases by a factor of 16 in the presence of disruptions, even though the disruptions would ideally decrease throughput by a factor of 2. The drop in TCP throughput is due to TCP's inability to adjust its timeout values and RTT after a connection returns. TCP stalls for a period of length t_w due to exponentially increasing timeout values during disruptions. t_w can grow linearly in T_d for certain values of T_d [148]. Therefore, if there is a period of connectivity followed by a equivalent period of disruption, and this trend repeats itself, TCP will have near-zero throughput.

7.2 Covering Holes with a Low Bandwidth Bridge

In the last section we saw that coverage holes (small and large) are rampant in WiFi based mobile networks. Such holes can have an adverse effect on the performance of TCP. A large suite of applications such as web browsing, web search, instant messaging, and voice is likely to perform very poorly in such environments. The performance degradation comes primarily from long timeouts and small congestion window sizes after the disruption occurs.

7.2.1 Patching Options

Patching coverage holes is challenging and non-trivial, although there are a number of existing proposals. For example, Vi-Fi [19] addresses transient holes by coordinating retransmissions by APs; it is unlikely that such a solution can be easily deployed in an unplanned WiFi network where APs are unlikely to coordinate, and it does not address permanent holes. Similarly, client-based solutions that leverage diversity, such as FatVAP [86], can only be applied to areas where multiple access points are available and do not address permanent holes.

Another option is to avoid WiFi and use very long range cellular/3G access; such networks impose a recurring cost for each device that the user carries, and are unavailable to a town or campus. For example, a municipality cannot offer free 3G access within its

Range	600 m
Max Data Rate	115.2 Kbps
UDP Throughput	47 Kbps
Lowest Transmit Power	1 mW
Highest Transmit Power	1 W
Receive Power	360 mW

Table 7.1. Characteristics of the Digi-XTend radios.

downtown, commercial area, nor can a University offer free 3G access on its campus; it's simply not an option. WiMax installations to cover large areas require a license, large towers, and carrier grade hardware (and will not be devoid of coverage holes).

Finally, the most obvious way to cover WiFi holes is to add WiFi infrastructure, but that has its own challenges [23, 77]. In general, covering a large area with medium range WiFi APs is much less expensive than *completely* covering the area so that no performance problems are present [107].

To summarize, we desire an inexpensive, long-range solution — off-the-shelf, unlicensed 900 MHz radios offer both characteristics. Moreover, they have a small energy profile and are appropriate for nomadic deployments. What such radios lack is bandwidth. A summary of the characteristics of the 900 MHz Digi-XTend radio is given in Table 7.1.

At first thought, it seems counter-intuitive to augment a WiFi network with a radio with a data rate that is 1/20 as large. This approach is explained by the following intuition: one of the primary factors affecting TCP performance is the exponential growth of timeouts. Timeouts result in a stalled period after connections are re-established, a period that can grow linearly with the disruption length. Moreover, TCP has to restart from a window size of one (slow start) after this period. Hence, our goal for the low-bandwidth bridge is to avoid this wasted period by keeping TCP in congestion avoidance through the period of WiFi disconnection and into a reconnection. As we show in the system evaluation, the throughput gains are substantial. Moreover, the bandwidth of 44 kbps offered for TCP and 47 Kbps for UDP (see Table 7.1) is significant, especially in the presence of larger coverage holes. VoIP codecs such as G.726 and G.728 require bit rates of less than 50 Kbps and would work well using just the Digi-XTends.

The other benefit of the Digi-XTend radio is that it supports several efficient cyclic sleep modes and has low-power transmit modes. The receive power of the radio is 360 mW (one third of WiFi and 3G) while the average transmit power is around 500 mW (also one third of WiFi and 3G). We are working on porting our system to solar powered Mote class devices as future work (see Chapter 8).

The primary decision that has to be made by the mobile node is when to switch between the two radios. Multiplexing or striping schemes for managing the single TCP stream over both radios will result in performance that is governed by the low bandwidth channel and will be affected greatly by the fluid connectivity changes. We expect the systems to be available simultaneously only at times. Epsilon, in its present implementation detects disruptions *in situ* and switches to the 900MHz radio whenever a disruption is detected. Disruption in WiFi connectivity can be detected by the absence of end-to-end connectivity (losing consecutive pings) or through lack of link-layer acknowledgments. We expect that the loss of connectivity using link-layer information can be detected quickly (on the order of tens of milliseconds).

7.3 Implementation

The main components of the Epsilon architecture are shown in Figure 7.4. The core architecture is proxy-based, similar to the MAR system [134]. There are three types of nodes in network: (1) the mobile node router, (2) bridge nodes, (3) and an Internet proxy. In this section, we describe the design and implementation of the software modules.

7.3.1 Mobile node router

Each mobile node in the network is assumed to have two radios: a WiFi radio and a Digi-XTend radio. Epsilon uses a user-space proxy at each mobile client. When a disruption

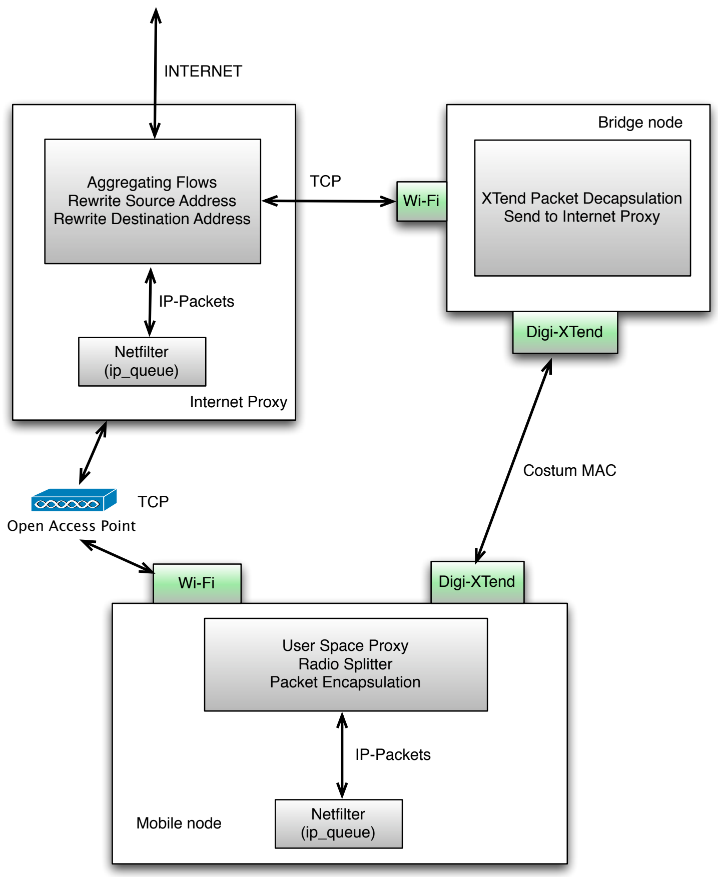


Figure 7.4. An overview of the Epsilon system architecture.

is detected, the proxy sends packets over the low bandwidth channel; otherwise it forwards them on WiFi.

As the mobile node associates with different access points, the IP address associated with its outbound wireless interface changes. To mask the effect of these changes, Epsilon uses a virtual dummy interface (`eth2`) and forces all application packets to be routed through this interface. The interface is assigned a static IP address which is unique for every mobile node. However, a single static route is set to the Internet proxy through the actual outbound interface (`wlan0`). The proxy uses `Netfilter` and `ip_queue` to capture IP packets from the kernel. If the mobile node proxy decides to route packets over WiFi, it encapsulates the captured IP packet in another IP packet and sends it to the Internet proxy over a TCP connection. Otherwise, the IP packet is encapsulated in a Digi-XTend packet and transferred over the 900 MHz link to the bridge.

7.3.2 Bridge Nodes

The bridge node is a Linux box equipped with a 900 MHz Digi-XTend radio as well as a WiFi radio. It acts as an intermediary between the mobile nodes and the Internet proxy. When receiving a packet over the Digi-XTend radio, the bridge first decapsulates it to get the raw IP packet, and then encapsulates it again in another IP packet. The bridge will then forward the packet over a TCP connection to the proxy. The reverse of the process occurs when it receives a packet from the Internet proxy.

7.3.3 Internet proxy

The Internet proxy acts as an aggregation point for packets being received from or sent to the bridge or mobile node. The following example demonstrates how the Internet proxy is used in Epsilon. Consider a mobile user sending a query to `google.com`. The IP packets sent from the bridge and the mobile node (encapsulated in other IP packets) have the address of the virtual interface of the mobile node as their source and `google.com` as their destination addresses. The Internet proxy rewrites the source address of the packets with its own address, recalculates the IP/TCP checksum and then sends the packets to the destination (here, `google.com`) using a raw socket. On the other hand, the packets received from `google.com` are queued using a `iptable` hook and trapped at the Internet proxy using `ip_queue`. The destination address of the packets is then rewritten with the destination address of the virtual interface of the mobile node. The packets are then sent over a previously established TCP connection with the mobile node or to the bridge from where they are sent to the mobile node over the Digi-XTends. The packets are decapsulated at the mobile node and then sent to the application using raw sockets. The mobile node re-initiates TCP connection with the Internet proxy every time it associates with a new open access point.

7.4 System Evaluation

In this section we evaluate the improvement in TCP throughput and application performance (HTTP transfers) with Epsilon. For evaluating these factors, we use transport-layer disruption traces from our mobile testbed. Moreover, to extend our results to other testbeds, we evaluate more general scenarios using synthetic workloads.

7.4.1 Evaluation Methodology

We have built a prototype of the Epsilon bridge node using the same hardware deployed on our mobile nodes, described in Chapter 3. In our evaluation, we compare two systems: *Epsilon*: our proposed dual-radio and bridge system; and *WiFi*: a system that uses only the WiFi radio.

For our experimental evaluation, we use traces of connectivity durations collected from our testbed to carry out trace-driven emulations on an indoor experimental setup. The computer representing the mobile node begins a TCP transfer to a server on the Internet using the WiFi network. Based on the traces of connectivity, the user-space proxy drops outgoing and incoming packets on the WiFi interface. If the mobile node decides to use the XTend radio, it connects and transfers data to an implementation of an Epsilon bridge node, also placed in our building.

Although the experimental setup is static and indoors, the effects of isolated disruptions are captured in the real world traces we have collected from our testbed in February 2009. Accordingly, the experimentation using trace-driven emulation has two major advantages. First, it provides us the flexibility to produce repeatable results and to try out various approaches without redeployment in the testbed; and second, using a full implementation on real hardware provides a comparison platform that produces accurate system delays.

7.4.2 TCP performance

We measured TCP performance using an `iperf` server running on a remote Internet host, and an `iperf` client running on the emulated mobile node. All experiments are

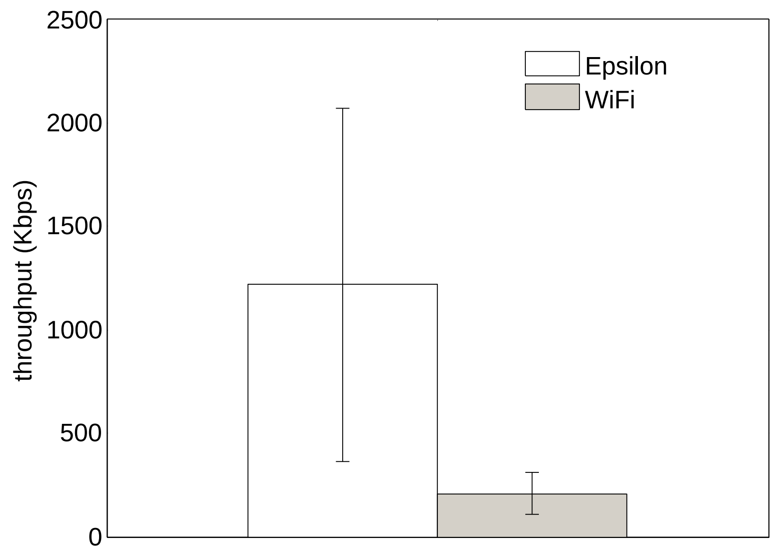


Figure 7.5. The TCP improvement produced by Epsilon. Epsilon produces an improvement of 6x in TCP throughput over a system which does not use the Digi-XTend radio.

based on 4-minute long sessions initiated continuously during the experiment by the client's `iperf` software. The proxy on the emulated mobile node drops packets when connectivity is absent.

The results in Figure 7.5 show a 6x improvement in average TCP throughput for Epsilon over just WiFi, i.e., from 200 Kbps to 1200 Kbps. Note that the additional TCP throughput that the Digi-XTends can provide is limited to less than 50 Kbps —-hence the extra 950 Kbps is an outcome of the 900MHz radio preventing TCP timeouts (hence minimizing the wasted time t_w), preventing TCP from moving into slow start, and keeping TCP in congestion avoidance. However, there is large variance in the Epsilon bar (shown by the error bars). Since we run 4-minute TCP sessions and in our mobile testbed we have disruptions of greater than 4 minutes, some TCP sessions fall into regimes where there is only 900MHz connectivity—gaining throughput of less than 50 Kbps.

Metric	WiFi	Epsilon
Web-pages downloaded	796	1187
HTTP timeouts	203	61

Table 7.2. Improvement produced by Epsilon for web transfers. Epsilon downloads 40% more web-pages and has less than 3x timeouts.

7.4.3 Application Performance

Our evaluation of throughput of TCP streams shows an order of magnitude improvement using Epsilon. Here we capture the performance of specific applications using Epsilon. We evaluated HTTP performance since it is the basis of popular Web browsing, Web email, and Web search applications. Instant messaging, file transfers, and other applications that rely on continuous, long-running TCP streams, will achieve improvements at least as good or better than HTTP-based applications using short-lived TCP connections.

In our web transfer experiments, we chose 10 popular web-pages and fetched their content using `wget`. The web-pages were spread across sources in U.S., Europe, and Asia. This two-hour experiment was based on two key performance metrics: the number of web-pages successfully downloaded; and the number of HTTP timeouts. The number of timeouts is proportional to the Web browsing experience a user would have. This is because timeouts lead to broken images, unloaded web-pages, and error messages.

Table 7.2 shows the relative benefits of using the Epsilon system for many Web applications. Epsilon is able to download more than 40% extra pages and experience 3x less timeouts. While web transfers involve short TCP transfers and are probably the worst case for Epsilon as compared to WiFi, we still see a substantial improvement in performance using the additional radio.

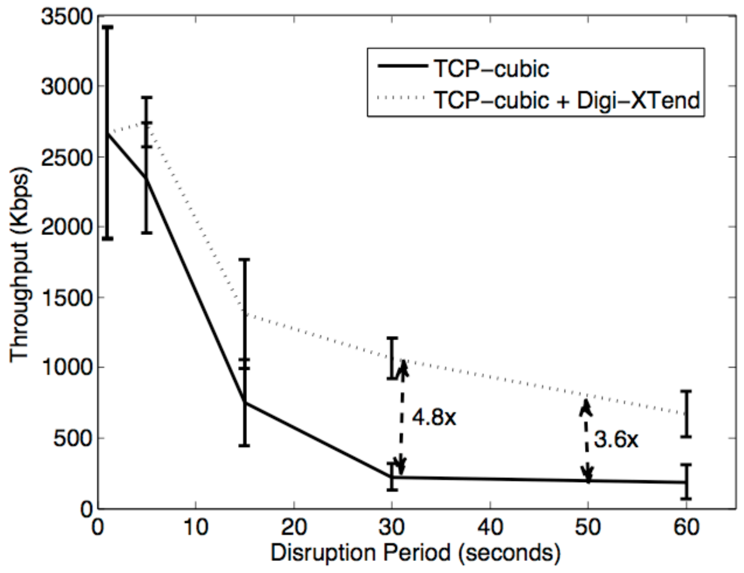


Figure 7.6. The figure shows TCP throughput as a function of the Off period with and without Epsilon. The disruption duration is varied and the ON period is chosen uniformly at random between 0-30 seconds.

7.4.4 Synthetic Workloads

While these performance improvements in TCP and application performance demonstrate Epsilon's benefits, the results are specific to our testbed. To generalize the results to other scenarios, we evaluated our system using a parametrized, synthetic connectivity trace.

The synthetic trace is characteristic of two parameters. The *on* period is when connectivity is not disrupted, and the *off* period is when connectivity is disrupted. We performed the experiments using the same `iperf`-based sessions from our emulated client node to an Internet host. Figure 7.6 shows TCP performance as a function of the *off* period statistics. The *on* period is chosen uniformly at random between 0–30 seconds. As the disruption periods increase the benefits of using Epsilon also increase and the *off* period is varied in the varied. When the length of the *off* period is comparable to the *on* period we see a 4.6x improvement using Epsilon. Figure 7.7 shows the benefits of Epsilon as the *on* period varies. The right hand side of the graph represents a very dense deployment of APs while the left hand corner represents a sparse distribution of access points. Cabernet [54] reports a median

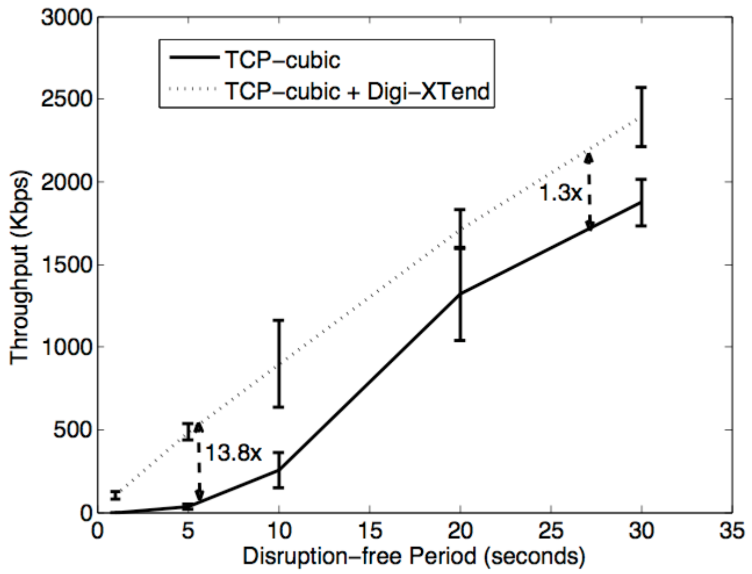


Figure 7.7. The figure shows TCP throughput as a function of the disruption-free/connected period with and without Epsilon. The disruption period is chosen uniformly at random between 0-30 seconds.

duration of 4 seconds for the *on* period and an *off* period (i.e., the median time between WiFi encounters) as 32 seconds. For these values, the results in Figure 7.6 show that Epsilon can provide a 13.8x improvement in TCP throughput. While the best known system [54] reports 2x improvement to TCP when associated with access points (effectively a 15% improvement when there are disruptions) Epsilon produces more than an order of magnitude improvement to TCP.

7.5 Discussion

In this chapter, we presented a system which uses a 900MHz bridge to patch coverage holes in organic and managed WiFi networks. The system can stripe TCP flows across the 900MHz and 2.4GHz channels to provide near ubiquitous connectivity in WiFi mobile networks. The 900MHz bridge helps keep TCP in congestion avoidance when WiFi connectivity reappears after a period of disruption/disconnection. While Throwboxes can provide enhanced connectivity in sparse mobile networks, Epsilon bridges can help provide (near) ubiquitous connectivity in WiFi based dense mobile networks. Hence, using a variety of alternate enhancements such as relays, mesh nodes (bridges), and base stations performance of diverse mobile networks can be enhanced.

CHAPTER 8
CONCLUSION AND FUTURE DIRECTIONS

This dissertation addresses the problem of providing improved consistency and network connectivity in mobile and sensor systems. We identify two fundamental problems which are primary deterrents to pervasive connectivity and high level of consistency in mobile and sensor systems. First, we attribute poor consistency and availability of clients to energy scarcity at end systems. Second, we observe that unreliable and poor connectivity offered by available infrastructure in mobile networks can also cause poor performance at clients.

To address the first problem, the thesis presents a novel energy management architecture called Hierarchical Power Management (HPM). HPM combines platforms with diverse energy needs and capability into one integrated system which provides high levels of consistency at minimal energy consumption. We address the second problem by alternative enhancement to mobile networks in the form of relays, mesh nodes, and base stations. We first study the relative performance benefits of each type of enhancement. Next, we present the design, implementation, and evaluation of relay and mesh based systems for enhancing sparse and dense mobile networks.

While the dissertation presents a convincing solution for improved connectivity and consistency in mobile and sensor systems, it also opens up several avenues for future research. First, a problem we have ignored in the deployment of mesh and relay enhancements is the placement of the nodes. Moreover, in the design of the 900MHz bridge to patch coverage holes we assumed that the nodes are tethered. It is unlikely to be the case in practical deployments, especially if coverage holes are large. To patch large coverage holes, we require multi-hop connectivity to a bridge node. This would require multiple solar powered 900MHz nodes deployed at various locations of the network. Such a nomadic deployment requires a highly energy-efficient design of nodes—something we leave as future work. Another interesting direction of research lies in the algorithmic issues with the Throwbox control algorithm. While the present incarnation of the algorithm considers local forwarding of packets, an ideal algorithm should consider end-to-end metrics such as end-to-end throughput and delay under the constraint of energy scavenged from solar. This presents an interesting direction where local information at the Throwboxes can be used to optimize for network wide global metrics.

BIBLIOGRAPHY

[1] Apple iPhone. `http://www.apple.com/iphone/`.

[2] Dodgeball Social Networking. `http://dodgeball.com/`.

[3] Loopt. `http://loopt.com`.

[4] Marker Focus: Bluetooth in Mobile Devices, WorldWide, 2004-2009. *Gartner*.

[5] Mobile Device - An Important Marker Segment for WLAN. *doc:IEEE*.

[6] Mobile Phone Sales. *Gartner Newsroom*.

[7] Nokia Smartphones. `http://nokia.com`.

[8] Pantopic Social Networking. `http://pantopic.com/`.

[9] Rummble Social Networking. `http://rummble.com/`.

[10] Zigbee Standard. *http://www.digi.com/technology/rf-articles/wireless-zigbee.jsp*.

[11] Afanasyev, M., Chen, T., Voelker, G. M., and Snoeren, A. C. Analysis of a Mixed-Use Urban WiFi Network: When Metropolitan becomes Neapolitan. In *IMC* (October 2008), pp. 85–98.

[12] Agarwal, Y., Chandra, R., Wolman, A., Bahl, V., Chin, K., and Gupta, R. Wireless Wakeups Revisited: Energy Management for VoIP over Wi-Fi Smartphones. In *Proceedings of Mobisys* (Puerto Rico, USA, June 2007).

[13] Anand, M., Nightingale, E. B., and Flinn, J. Self-Tuning Wireless Network Power Management. In *Proceedings of MobiCom* (San Diego, CA, September 2003).

[14] Balakrishnan, H., Padmanabhan, V., and Katz, R. The Effects of Asymmetry on TCP Performance. *ACM Mobile Networks and Applications (MONET)* (1999).

[15] Balakrishnan, H., Padmanabhan, V. N., Seshan, S., and Katz, R. H. A Comparison of Mechanisms for Improving TCP Performance over Wireless Links. In *Proceedings of Sigcomm* (1996), pp. 256–269.

[16] Balani, R., Han, S., Raghunathan, V., and M.B.Srivastava. Remote Storage for Sensor Networks. UCLA-NESL-200504-09, UCLA, 2005.

[17] Balasubramanian, A., Levine, B. N., and Venkataramani, A. DTN Routing as a Resource Allocation Problem. In *Proceedings of ACM Sigcomm* (August 2007).

[18] Balasubramanian, A., Levine, B. N., and Venkataramani, A. Enabling Interactive Applications in Hybrid Networks. In *Proceedings of Mobicom* (September 2008).

[19] Balasubramanian, A., Mahajan, R., Venkataramani, A., Levine, B., and Zahorjan, J. Interactive WiFi Connectivity for Moving Vehicles. In *Proceedings of ACM Sigcomm* (August 2008).

[20] Balasubramanian, A., Zhou, Y., Croft, W. B., Levine, B. N., and Venkataramani, A. Web Search From a Bus. In *Proceedings of ACM Workshop on Challenged Networks* (September 2007).

[21] Banerjee, N., Agarwal, S., Chandra, R., Bahl, P., Wolman, A., and Corner, M. Virtual Compass: Relative Positioning to Sense Mobile Social Interactions. *Under Submission, Sensys* (2009).

[22] Banerjee, N., Corner, M. D., and Levine, B. N. An Energy-Efficient Architecture for DTN Throwboxes. In *Proceedings of IEEE Infocom* (May 2007).

[23] Banerjee, N., Corner, M. D., Towsley, D., and Levine, B. N. Relays, Base Stations, and Meshes: Enhancing Mobile Networks with Infrastructure. In *Proceedings of ACM MobiCom* (San Francisco, CA, USA, September 2008).

[24] Banerjee, N., Rahmati, A., Corner, M. D., Rollins, S., and Zhong, L. Users and Batteries: Interactions and Adaptive Energy Management in Mobile Systems. In *Proceedings of Ubicomp* (Innsbruck, Austria, September 2007).

[25] Banerjee, N., Wei, W., and Das, S. K. Mobility Support in Wireless Internet. *IEEE Wireless Communications* (2009).

[26] Batalin1, M. A., and Sukhatme1, G. S. Coverage, Exploration and Deployment by a Mobile Robot and Communication Network. *Telecommunications Systems* (2004).

[27] Bicket, J., Aguayo, D., Biswas, S., and Morris, Robert. Architecture and Evaluation of an Unplanned 802.11b Mesh Network. In *Proceedings of MobiCom* (August 2005), pp. 31–42.

[28] Blinn, D. P., Henderson, T., and Kotz, D. Analysis of a Wi-Fi Hotspot Network. In *Proceedings of WiTMeMo* (2005), pp. 1–6.

[29] Brakmo, L. S., Wallach, D. A., and Viredaz, M. A. microSleep: A Technique for Reducing Energy Consumption in Handheld Devices. In *Proceedings MobiSys* (Boston, MA, June 2004).

[30] Brekhovskikh, Leonid, and Lysanov, Yuri. *Fundamentals of Ocean Acoustics*, 3rd ed. Springer-Verlag, 2003.

[31] Brooks, D., Tiwari, V., and Martonosi, M. Wattch: A Framework for Architectural-level Power analysis and Optimizations. In *Proceedings of ICSA* (2000), pp. 83–94.

[32] Bruno, R., Conti, M., and Gregori, E. Mesh Networks: Commodity Multihop Ad-hoc Networks. *IEEE Communications Magazine 43*, 3 (March 2005).

[33] Burgess, John, Gallagher, Brian, Jensen, David, and Levine, Brian Neil. MaxProp: Routing for Vehicle-Based Disruption-Tolerant Networks. In *Proc. IEEE INFOCOM* (April 2006).

[34] Burns, B., Brock, O., and Levine, B. N. *MV* routing and capacity building in disruption tolerant networks. In *Proceedings of IEEE INFOCOM* (March 2005), pp. 398–408.

[35] Burns, B., Brock, O., and Levine, B. N. MV routing and capacity building in disruption tolerant networks. In *Proceedings of IEEE Infocom 2005* (March 2005).

[36] Burns, B., Brock, O., and Levine, B. N. Autonomous Enhancement of Disruption Tolerant Networks. In *Proceedings of IEEE ICRA* (May 2006).

[37] Burns, B., Brock, O., and Levine, B. N. MORA Routing and Capacity Building in Disruption-Tolerant Networks. *Elsevier Ad hoc Networks Journal* (2008). To appear.

[38] Bychkovsky, V., Hull, B., Miu, A. K., Balakrishnan, H., and Madden, S. A Measurement Study of Vehicular Internet Access Using in situ 802.11 Networks. In *Proceedings of ACM Mobicom* (Sept 2006), pp. 50–61.

[39] Caceres, R., Carter, C., Narayanaswami, C., and Raghunath, M. T. Reincarnating PCs with Portable SoulPads. In *Proceedings of MobiSys* (2005).

[40] Camp, J., Robinson, J., Steger, C., and Knightly, E. Measurement Driven Deployment of a two-tier Urban Mesh Access Network. In *Proceedings of ACM MobiSys* (2006), pp. 96–109.

[41] Chaintreau, Augustin, Hui, Pan, Crowcroft, Jon, Diot, Christophe, Gass, Richard, and Scott, James. Impact of Human Mobility on the Design of Opportunistic Forwarding Algorithms. In *Proceedings of IEEE INFOCOM* (Apr 2006).

[42] Chandra, R., Bahl, P., and Bahl, P. MultiNet: Connecting to Multiple IEEE 802.11 Networks Using a Single Wireless Card. In *Proceedings of IEEE Infocom* (2004).

[43] Choi, S., and Shin, K. G. Predictive and Adaptive Bandwidth Reservation for Handoffs in QoS-sensitive Cellular Networks. In *Proceedings of ACM Sigcomm* (1998).

[44] Cole, R. G., and Rosenbluth, J. H. Voice over IP Performance Monitoring. *CCR* (2001).

[45] Cormen, T. H., Lieserson, C. E., Rivest, R. L., and Stein, C. *Introduction to Algorithms*. MIT Press, 2001.

[46] Cortes, J., Martinez, S., Karatas, T., and Bullo, F. Coverage Control for Mobile Sensing Networks. *Proceedings of IEEE ICRA* (2002).

[47] Crossbow Technology Inc. *Stargate Developer's Guide*, Rev. A ed. San Jose, CA, September 2004. 7430-0317-12.

[48] D. McIntire, K. Ho, Yip, B., Singh, A., Wu, W., and Kaiser, W. J. The Low Power Energy Aware Processing (LEAP) Embedded Networked Sensor System. Tech. rep., UCLA, Los Angeles, CA, 2005.

[49] Davis, J., Fagg, A., and Levine, B. N. Wearable Computers and Packet Transport Mechanisms in Highly Partitioned Ad hoc Networks. In *Proc. IEEE Intl. Symp on Wearable Computers (ISWC)* (October 2001), pp. 141–148.

[50] Douglis, F., Krishnan, P., and Bershad, B. N. Adaptive Disk Spin-down Policies for Mobile Computers. In *Proceedings of the 2nd Symposium on Mobile and Location-Independent Computing* (April 1995).

[51] Douglis, F., Krishnan, P., and Marsh, B. Thwarting the Power-hungry Disk. In *Proceedings of The USENIX Winter 1994 Technical Conference* (San Francisco, CA, 1994).

[52] Dutta, P., and Culler, D. Practical Asynchronous Neighbor Discovery and Rendezvous for Mobile Sensing Applications. *Proceedings of Sensys* (2008).

[53] Eagle, N., and Pentland, A. Reality Mining: Sensing Complex Social Systems. *Personal and Ubiquitous Computing 10*, 4 (2006), 255–268.

[54] Eriksson, J., Balakrishnan, H., and Madden, S. Cabernet: Vehicular Content Delivery Using WiFi. In *Proceedings of ACM MobiCom* (San Francisco, CA, September 2008).

[55] Fall, K. A Delay-Tolerant Network Architecture for Challenged Internets. In *Proceedings of ACM Sigcomm* (2003), pp. 27–34.

[56] Ferguson, P., and Huston, G. *Quality of Service: Delivering QoS on the Internet and in Corporate Networks*. Wiley and Sons, Inc, 1998.

[57] Fioravanti-Score, A., Mitchell, Sarah V., and Williamson, J. Michael. Use of Satellite Telemetry Technology to Enhance Research and Education in the Protection of Loggerhead Sea Turtles. In *19th Annual Symposium on Sea Turtle Biology and Conservation* (1999).

[58] Flautner, K., Reinhardt, S., and Mudge, T. Automatic Performance-setting for Dynamic Voltage Scaling. In *Proceedings ACM MobiCom* (Rome, Italy, July 2001).

[59] Flinn, J., and Satyanarayanan, M. Managing Battery Lifetime with Energy-Aware Adaptation. *ACM Transactions on Computer Systems (TOCS) 22*, 2 (May 2004).

[60] Forman, G.H., and Zahorjan, J. The Challenges of Mobile Computing. *IEEE Computers* (1994).

[61] Froehlich, J., Chen, M. Y., Consolvo, S., Harrison, B. L., and Landay, J. A. MyExperience: A system for in situ Tracing and Capturing of User Feedback on Mobile Phones. In *Proceedings of MobiSys* (2007).

[62] Gaonkar, S., Li, J., Choudhary, R. R., Cox, L., and Schmidt, A. Micro-Blog: Sharing and Querying Content Through Mobile Phones and Social Participation. In *Proceedings of MobiSys* (2008).

[63] Gass, R., Scott, J., and Diot, C. Measurements of In-Motion 802.11 Networking. In *Proceedings of WMCSA* (2006), pp. 69–74.

[64] Gay, D., Levis, P., Behren, R. V., Welsh, M., Brewer, E., and Culler, D. The nesC Language: A Holistic Approach to Networked Embedded Systems. In *Proceedings of PLDI* (June 2003).

[65] Girod, L., Stathopoulos, T., Ramanathan, N., Elson, J., Estrin, D., Osterweil, E., and Schoellhammer, T. A System for Simulation, Emulation, and Deployment of Heterogeneous Sensor Networks. In *Proceedings of SenSys* (Baltimore, MD, November 2004).

[66] Goodman, D. J., Borras, J., Mandayam, N. B., and Yates, R. D. INFOSTATIONS: A New System Model for Data and Messaging Services. In *Proceedings of Vehicular Technology Conference* (Phoenix, AZ, May 1997), pp. 969–973.

[67] Govil, K., Chan, E., and Wasserman, H. Comparing Algorithms for Dynamic Speed-setting of a low-power CPU. In *Proceedings of MobiCom* (Berkeley, CA, November 1995).

[68] Gray, B. Soldiers, Agents, and Wireless Networks: A Report on a Military Application. *Technical Report* (2000).

[69] Grieco, L. A., and Mascolo, S. Performance evaluation and comparison of Westwood+, New Reno and Vegas TCP congestion control. In *ACM CCR* (2004), pp. 69–74.

[70] Groenevelt, R., Nain, P., and Koole, G. The Message Delay in Mobile Ad hoc Networks. In *Proceedings of Performance* (2006).

[71] Gumstix Inc. *http://www.gumstix.com/*, ed. San Jose, CA, September 2004. 7430-0317-12.

[72] Helmbold, D. P., Long, D. D. E., and Sherrod, B. A Dynamic Disk Spin-down Technique for Mobile Computing. In *Proceedings of the MobiCom* (Rye, NY, November 1996).

[73] Horst, G. R., Hoagland, D. B., and Kilpatrick, C. W. The mongoose in the west indies: The biogeography and population biology of an introduced species. *Biogeography of the West Indies: Patterns and Perspectives* (2003), 409–424.

[74] Howard, A., Mataric, M. J., and Sukhatme, G. S. Mobile Sensor Network Deployment using Potential Fields: A Distributed, Scalable Solution to the Area Coverage Problem. *Proceedings of DARS* (2002).

[75] Huang, H., Pillai, P., and Shin, K. G. Design and Implementation of Power-Aware Virtual Memory. In *Proceedings of USENIX Technical Conference* (San Antonio, TX, June 2003).

[76] Hui, P., Chaintreau, A., Scott, J., Gass, R., Crowcroft, J., and Diot, C. Pocket Switched Networks and Human Mobility in Conference Environments. In *Proceedings of ACM Workshop on Delay-Tolerant Networking* (August 2005), pp. 244–251.

[77] Hui, P., Lindgren, A., and Crowcroft, J. Empirical Evaluation of Hybrid Opportunistic Networks. In *Proceedings of COMSNETS* (Bangalore, India, January 2009).

[78] Hull, B., Bychkovsky, V., Zhang, Y., Chen, K., Goraczko, M., Miu, A., Shih, E., Balakrishnan, H., and Madden, S. CarTel: A Distributed Mobile Sensor Computing System. In *Proceedings of ACM SenSys* (October 2006), pp. 125–138.

[79] Ibrahim, M., Hanbali, A. Al, and Nain, P. Delay and Resource Analysis in MANETs in Presence of Throwboxes. In *Proceedings of Performance* (October 2007).

[80] Jain, S., Fall, K., and Patra, R. Routing in a Delay Tolerant Network. In *Proceedings of ACM Sigcomm* (August 2004).

[81] Joseph, A. D., and Kaashoek, M. F. Building Reliable Mobile-Aware Applications using the Rover Toolkit. In *Proceedings of MobiCom* (White Plains, NY, November 1996).

[82] Juang, P., Oki, H., Wang, Y., Martonosi, M., Peh, L. S., and Rubenstein, D. Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet. *SIGOPS Operating Systems Review 36*, 5 (2002), 96–107.

[83] Jun, H., Ammar, M. H., Corner, M. D., and Zegura, E. Hierarchical Power Management in Disruption Tolerant Networks with Traffic-Aware Optimization. In *Proc. ACM SIGCOMM Workshop on Challenged Networks (CHANTS)* (September 2006).

[84] Jun, H., Ammar, M. H., and Zegura, E. W. Power Management in Delay Tolerant Networks: A Framework and Knowledge-Based Mechanisms. In *Proceedings of IEEE SECON)* (2005).

[85] Kamijoh, N., Inoue, T., Olsen, C. M., Raghunath, M. T., and Narayanaswami, C. Energy trade-offs in the IBM Wristwatch Computer. In *Proceedings of International Symposium on Wearable Computers* (Zurich, Switzerland, October 2001).

[86] Kandula, S., Lin, K. Ching-Ju, Badirkhanli, T., and Katabi, D. FatVAP: Aggregating AP Backhaul Capacity to Maximize Throughput. In *Proceedings of NSDI* (San Francisco, CA, April 2008).

[87] Kravets, R., and Krishnan, P. Power Management Techniques for Mobile Communication. In *Proceedings of MobiCom* (Dallas, TX, October 1998).

[88] Kubach, U., and Rothermel, K. Exploiting Location Information for Infostation-based Hoarding. In *In Proceedings of MobiCom* (2004), pp. 217–230.

[89] Kulkarni, P., Ganesan, D., and Shenoy, P. Senseye: A multi-tier camera sensor network. In *ACM Multimedia* (2005).

[90] LaMarca, A., and de Lara, E. Location Systems: An Introduction to the Technology Behind Location Awareness. *Synthesis Lectures on Mobile and Pervasive Computing* (2008).

[91] Lebeck, A. R., Fan, X., Zeng, H., and Ellis, C. S. Power Aware Page Allocation. In *Proceedings of ASPLOS* (Cambridge, MA, November 2000).

[92] Lester, J., Choudhury, T., and Borriello, G. A Practical Approach to Recognizing Physical Activities. In *Proceedings of Pervasive* (Dublin, Ireland, May 2006).

[93] Li, K., Sohn, T., Huang, S., and Griswold, W. PeopleTones: A System for the Detection and Notification of Buddy Proximity on Mobile Phones. In *Proceedings of MobiSys* (2007).

[94] Li, M., Ganesan, D., and Shenoy, P. PRESTO: Feedback-driven data management in sensor networks. In *Proceedings of the 3nd USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI 2006)* (May 2006).

[95] Li, Q., and Rus, D. Sending Messages to Mobile Users in Disconnected Ad hoc Wireless Networks. In *Proceedings of MobiCom* (August 2000), pp. 44–55.

[96] Lin, Kris, Hsu, Jason, Zahedi, Sadaf, Lee, David C, Friedman, Jonathan, Kansal, Aman, Raghunathan, Vijay, and Srivastava, Mani B. Heliomote: Enabling long-lived sensor networks through solar energy harvesting. In *Proceedings of ACM Sensys* (November 2005).

[97] Lindgren, A., Diot, C., and Scott, Ja. Impact of communication infrastructure on forwarding in pocket switched networks. In *Proceedings of CHANTS* (Pisa, Italy, September 2006), pp. 261 – 268.

[98] Litzkow, M., Livny, M., and Mutka, M. Condor: A Hunter of idle Workstations. In *Proceedings of International Conference on Distributed Computing Systems* (June 1988).

[99] Liu, T., Bahl, P., and Chlamtac, I. Mobility Modeling, Location Tracking, and Trajectory Prediction in Wireless ATM networks. *IEEE JSAC 16*, 6 (1998), 922–936.

[100] Liu, Xiaotao, Shenoy, Prashant, and Corner, Mark D. Chameleon: Application Controlled Power Management with Performance Isolation. In *Proc. ACM Multimedia* (Singapore, November 2005).

[101] Lorch, J. R., and Smith, A. J. Reducing Processor Power Consumption by Improving Processor Time management in a Single-user Operating System. In *Proceedings of MobiCom* (Rye, NY, Novemeber 1996).

[102] Lorincz, K., Chen, Bor-rong, Waterman, J., Werner-Allen, G., and Welsh, M. Resource Aware Programming in the Pixie OS. In *Proceedings of SenSys* (New York, NY, USA, 2008), ACM, pp. 211–224.

[103] Luz, V. De La, Kandemir, M., and Kolcu, I. Automatic Data Migration for Reducing Energy Consumption in multi-bank Memory Systems. In *Proceedings of the 39th Conference on Design automation* (New Orleans, LA, June 2002).

[104] Lymberopoulos, D., Priyantha, N. B., and Zhao, F. mplatform: A Reconfigurable Architecture and Efficient Data Sharing Mechanism for Modular Sensor Nodes. In *Proceedings of IPSN* (Los Angeles, CA, April 2007).

[105] Lymberopoulos, D., and Savvides, A. XYZ: A Motion-Enabled, Power Aware Sensor Node Platform for Distributed Sensor Network Applications. In *Proceedings of ISPN* (Los Angeles, CA, April 2005).

[106] M. Anand, E. B. Nightingale, and Flinn, J. Ghosts in the Machine: Interfaces for Better Power Management. In *Proceedings of MobiSys* (Boston, MA, June 2004).

[107] Mahajan, R., Zahorjan, J., and Zill, B. Understanding WiFi-based Connectivity From Moving Vehicles. In *Proceedings of IMC* (2007).

[108] Mahesri, A., and Vardhan, V. Power Consumption Breakdown on a Modern Laptop. *Proceedings of PACS*.

[109] Mascolo, S., Torino, P. Di, Gerla, M., Sanadidi, M. Y., and Wang, R. TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links. In *Proceedings of ACM MobiCom* (2001), pp. 287–297.

[110] Mayo, R., and Ranganathan, P. Energy consumption in mobile devices: Why future systems need requirements-aware energy scale-down. *Lecture Notes in Computer Science* (2003). Special Issue on Power Management.

[111] Merugu, S., Ammar, M., and Zegura, E. Space-Time Routing in Wireless Networks with Preictable Mobility. Tech. Rep. GIT-CC-04-07, College of Computing, Georgia Institute of Technology, March 2004.

[112] Military. Rapid Deployment of Secure Mobile Communications. *Technical Report*.

[113] Miluzzo, E., Lane, N. D., Fodor, K., Peterson, R. A., Lu, H., Musolesi, M., Eisenman, S. B., Zheng, X., and Campbell, A. T. Sensing Meets Mobile Social Networks: The Design, Implementation and Evaluation of the CenceMe Application. In *Proceedings of SenSys* (2008).

[114] Musoll, E., Lang, T., and Cortadella, L. Exploiting the Locality of Memory References to Reduce the address Bus Energy. In *Proceedings of the 1997 International Symposium on Low Power Electronics and Design* (Monterey, CA, August 1997).

[115] Neugebauer, R., and McAuley, D. Energy is just another resource: Energy accounting and energy pricing in the Nemesis OS. In *Proceedings of HotOS* (Schloss Elmau, Germany, May 2001).

[116] Nicholson, A. J., Chawathe, Y., Chen, M. Y., Noble, B. D., and Wetherall, D. Improved Access Point Selection. In *Proceedings of MobiSys* (2006), pp. 233–245.

[117] Nicholson, A. J., Wolchok, S., and Noble, B. D. Juggler: Virtual Networks for Fun and Profit. *IEEE Transactions on Mobile Computing* (April 2008).

[118] Noble, B., Satyanarayanan, M., Narayanan, D., Tilton, J.E., Flinn, J., and Walker, K. Agile Application-Aware Adaptation for Mobility. In *Proceedings of the 16th ACM Symposium on Operating System Principles* (St. Malo, France, October 1997).

[119] Olsen, C. M., and Morrow, L. Alex. Multi-processor Computer System Having Low Power Consumption. In *Proceedings of the Second International Workshop on Power-Aware Computer Systems* (Cambridge, MA, February 2002).

[120] Onnela, J. P., Saramki, J., Hyvnen, J., Szab, G., Lazer, D., Kaski, K., Kertsz, J., and Barabsi, A. L. Structure and Tie Strengths in Mobile Communication Networks. *Proceedings of National Academy of Science*.

[121] Ott, J., and Kutscher, D. Drive-Thru Internet: IEEE 802.11b for "Automobile" Users. In *Proceedings of IEEE Infocom* (March 2004), pp. 362–373.

[122] Papathanasiou, A. E., and Scott, M. L. Energy Efficiency through Burstiness. In *Proceedings of WMCSA* (Monterey, CA, October 2003).

[123] Peleato, B., and Stojanovic, M. A MAC Protocol for Ad hoc Underwater Acoustic Sensor Networks. In *Proceedings of WUWNet* (2006), pp. 113–115.

[124] Peng, C., Shen, G., Zhang, Y., Li, Y., and Tan, K. Beep Beep: A High Accuracy Acoustic Ranging System Using COTS Mobile Devices. In *Proceedings of SenSys* (2007).

[125] Pering, T., Agarwal, Y., Gupta, R., and Want, R. CoolSpots: Reducing the Power Consumption of Wireless Mobile Devices with Multiple Radio Interfaces. In *Proceedings of ACM Mobisys* (June 2006).

[126] Pering, T., Burd, T., and Brodersen, R. The Simulation and Evaluation of Dynamic Voltage Scaling Algorithms. In *Proceeedings of the International Symposium on Low Power Electronics and Design* (Monterey, CA, August 1998).

[127] Pering, Trevor, Agarwal, Yuvraj, Gupta, Rajesh, and Want, Roy. CoolSpots: Reducing the Power Consumption of Wireless Mobile Devices with Multiple Radio Interfaces. In *Proc. ACM MobiSys* (June 2006), pp. 220–232.

[128] Plank, J., Beck, M., Kingsley, G., and Li, K. Libckpt: Transparent checkpointing under Unix. In *Proceedings of the USENIX Winter 1995 Technical Conference* (January 1995).

[129] Polastre, J., Szewczyk, R., and Culler, D. Telos: Enabling Ultra-low Power Wireless Research. In *Proceedings of IPSN/SPOTS* (April 2005).

[130] Polastre, J., Szewczyk, R., Sharp, C., , and Culler, D. The Mote Revolution: Low power wireless sensor networks. In *Proceedings of HotChips* (August 2004).

[131] Pompili, D., Melodia, T., and Akyildiz, I. F. Routing algorithms for Delay-insensitive and Delay-sensitive Applications in Underwater Sensor Networks. In *Proceedings of MobiCom* (2006), pp. 298–309.

[132] Rahmati, A., and Zhong, L. Context-for-Wireless: Context-Sensitive Energy-Efficient Wireless Data Transfer. In *Proceedings of Mobisys* (Puerto Rico, USA, June 2007).

[133] Robinson, J., Swaminathan, R., and Knightly, E. Assessment of Urban-Scale Wireless Networks with a Small Number of Measurements. In *Proceedings of ACM Mobicom* (2008).

[134] Rodriguez, P., Pratt, I., Chakravorty, R., and Banerjee, S. MAR: A Commuter Router Infrastructure for the Mobile Internet. In *Proceedings of ACM MobiSys* (2004), pp. 217–230.

[135] Sarafijanovic-Djukic, N., and Grossglauser, M. Last Encounter Routing under Random Waypoint Mobility. In *Proceedings of IFIP Networking* (May 2004).

[136] Sastry, N., Crowcroft, J., and Sollins, K. Architecting Citywide Ubiquitous Wi-Fi Access. In *Proceedings of HotNets 2007* (Atlanta, Georgia, November 2007).

[137] Schott, B., Bajura, M., Czarnaski, J., Flidr, J., Tho, T., and Wang, L. A Modular Power-Aware Microsensor with 1000X Dynamic Power Range. In *Proceedings of IPSN* (Los Angeles, CA, April 2005).

[138] Shih, E., Bahl, P., and Sinclair, M. J. Wake on Wireless: An Event Driven Energy Saving Strategy for Battery Operated Devices. In *Proceedings of ACM Mobicom* (Atlanta, GA, September 2002).

[139] Shin, M., Hong, S., and Rhee, I. DTN Routing Strategies using Optimal Search Patterns. In *Proceedings of ACM MobiCom Workshop on Challenged Networks* (September 2008).

[140] Shin, M. H., Mishra, A., and Arbaugh, W. Improving the Latency of 802.11 hand-offs using neighbor graphs. In *Proceedings of Mobisys* (2004), pp. 69–74.

[141] Sibley, G.T., Rahimi, M.H., and Sukhatme, G.S. Robomote: A Tiny Mobile Robot Platform for large-scale Ad-hoc Sensor Networks. *Proceedings of IEEE ICRA* (2002).

[142] Soh, W., and Kim, H. S. Dynamic Bandwidth Reservation in Cellular Networks Using Road Topology Based Mobility Predictions. In *Proceedings of IEEE Infocom* (March 2004).

[143] Son, D., Krishnamachari, B., and Heidemann, J. Experimental Study of Concurrent Transmission in Wireless Sensor Networks. In *Proceedings of SenSys* (2006), pp. 237–250.

[144] Song, L., Deshpande, U., Kozat, U. C., Kotz, D., and Jain, R. Predictability of WLAN Mobility and its Effects on Bandwidth Provisioning. In *Proceedings of IEEE Infocom* (2006).

[145] Sorber, J., Banerjee, N., Corner, M. D., and Rollins, S. Turducken: Hierarchical Power Management for Mobile Devices. In *Proceedings of MobiSys* (Seattle, WA, June 2005).

[146] Sorber, J., Banerjee, N., Corner, M. D., and Rollins, S. Turducken: Hierarchical power management for mobile devices. In *Proceedings of ACM MobiSys* (Seattle, WA, 2005).

[147] Sorber, J., Kostadinov, A., Garber, M., Brennan, M., Corner, M. D., and Berger, E. D. Eon: A Language and Runtime System for Perpetual Systems. In *Proceedings ACM SenSys* (Syndey, Australia, November 2007).

[148] Soroush, H., Banerjee, N., Corner, M. D., and Levine, B. N. Epsilon: Patching WiFi Mobile Networks. *Technical Report* (May 2009).

[149] Soroush, H., Banerjee, N., Corner, M. D., Levine, B. N., and Lynn, B. DOME: A Diverse Outdoor Mobile Testbed. Department of Computer Science Technical Report UM-CS-2009-23, Univ. of Massachusetts Amherst, May 2009.

[150] Spyropoulos, T., Psounis, K., and Raghavendra, C. S. Spray and Wait: An Efficient Routing Scheme for Intermittently Connected Mobile Networks. In *Proceedings of ACM Workshop on Delay Tolerant Networking* (2005).

[151] Stathopoulos, T., Lukac, M., McIntire, D., Heidemann, J., Estrin, D., and Kaiser, W. End-to-end Routing for Dual-Radio Sensor Networks. In *Proceedings of the IEEE Infocom 2007* (Anchorage, Alaska, USA, May 2007).

[152] Stellner, G. CoCheck: Checkpointing and Process Migration for MPI. In *Proceedings of International Parallel Processing Symposium* (April 1996).

[153] Su, W., Lee, S., and Gerla, M. Mobility Prediction and Routing in Ad hoc Wireless Networks. *International Journal of Network Management* (2001).

[154] Tanenbaum, A. S. *Computer Networks*, 3rd ed. Prentice-Hall, 2003.

[155] Theimer, M. M., Lantz, K. A., and Cheriton, D. R. Preemptable Remote Execution Facilities for the V System. In *Proceedings of SOSP* (Orcas Island, WA, December 1985).

[156] Tiwari, V., Malik, S., and Wolfe, A. Compilation Techniques for Low Energy: An Overview. In *Proceedings of IEEE Symposium on Low Power Electronics* (October 1994).

[157] Trifa, V. M., Girod, L., Collier, T., Blumstein, D. T., and Taylor, C. E. Automated Wildlife Monitoring Using Self-Configuring Sensor Networks Deployed in Natural Habits. *Proceedings of AROB* (2007).

[158] Vahdat, A., and Becker, D. Epidemic routing for partially-connected ad hoc networks. Tech. rep., Duke University, 2000.

[159] Varshavsky, A., de Lara, E., Hightower, J., LaMarca, A., and Otsason, V. GSM indoor localization. In *Pervasive and Mobile Computing journal* (Dec. 2007).

[160] Want, R., Pering, T., Danneels, G., Kumar, M., Sundar, M., and Light, J. The Personal Server - Changing the Way We Think about Ubiquitous Computing. In *Proceedings of Ubicomp* (Goteborg, Sweden, September 2002).

[161] Weiser, M., Welch, B., Demers, A., and Shenker, S. Scheduling for Reduced CPU Energy. In *Proceedings of OSDI* (Monterey, CA, November 1994).

[162] Whitaker, A., Cox, R. S., Shaw, M., and Gribble, S. D. Constructing Services with Interposable Virtual Hardware. In *Proceedings of NSDI* (San Francisco, CA, March 2004).

[163] Ye, W., Heidemann, J., and Estrin, D. An Energy-Efficient MAC Protocol for Wireless Sensor Networks. In *Proceedings of IEEE Infocom* (New York, NY, USA, June 2002), pp. 1567–1576.

[164] Yin, L., Li, B., Zhang, Z., and Lin, Yi-Bing. Performance Analysis of a Dual-threshold Reservation (DTR) Scheme for voice/data Integrated Mobile Wireless Networks. *Proceedings of IEEE WCNC* (2000).

[165] Yu, X. Improving TCP Performance over Mobile ad hoc Networks by Exploiting Cross-layer Information Awareness. In *Proceedings of MobiCom* (New York, NY, USA, 2004), ACM, pp. 231–244.

[166] Zeng, H., Ellis, C. S., Lebeck, A. R., and Vahdat, A. ECOSystem: Managing Energy as a first class Operating System Resource. In *Proceedings of ASPLOS* (San Jose, CA, October 2002).

[167] Zhang, P., Sadler, C. M., Lyon, S. A., and Martonosi, M. Hardware Design Experiences in ZebraNet. In *Proceedings of SenSys* (New York, NY, USA, 2004), ACM, pp. 227–238.

[168] Zhang, P., Sadler, C. M., Lyon, S. A., and Martonosi, M. Hardware Design Experiences in ZebraNet. In *Proceedings of SenSys* (New York, NY, USA, 2004), ACM, pp. 227–238.

[169] Zhang, X., Kurose, J., Levine, B. N., Towsley, D., and Zhang, H. Study of a Bus-Based Disruption Tolerant Network: Mobility Modeling and Impact on Routing. In *Proceedings of ACM Mobicom* (September 2007).

[170] Zhang, X., Neglia, G., Kurose, J., and Towsley, D. Performance Modeling of Epidemic Routing. In *Proceedings of IFIP Networking* (2006).

[171] Zhao, W., and Ammar, M. Message Ferrying: Proactive Routing In Highly Partitioned Wireless Ad hoc Networks. In *Proceedings of IEEE Workshop on Future Trends in Distributed Computing Systems* (May 2003).

[172] Zhao, W., Ammar, M., and Zegura, E. A Message Ferrying Approach for Data Delivery in Sparse Mobile Ad Hoc Networks. In *Proceedings of MobiHoc* (May 2004).

[173] Zhao, W., Ammar, M., and Zegura, E. Controlling the Mobility of Multiple Data Transport Ferries in a Delay-Tolerant Network. In *Proceedings of IEEE INFOCOM* (2005).

[174] Zhao, W., Chen, Y., Ammar, M., Corner, M. D., Levine, B. N., and Zegura, E. Capacity Enhancement using Throwboxes in DTNs. In *Proceedings of IEEE MASS* (Oct 2006), pp. 31–40.

[175] Zhu, J., Hung, Ka-Lok, and Bensaou, B. Tradeoff between Network Lifetime and Fair rate allocation in Wireless Sensor Networks with Multi-path Routing. In *Proceedings of MSWiM* (New York, NY, USA, 2006), ACM, pp. 301–308.