Implementation of a Power-Saving Protocol for Ad Hoc Wireless Networks

by

Kyle Jamieson

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

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2002

Author . . . .
Department of Electrical Engineering and Computer Science
February 4, 2002

Certified by . . . .
Hari Balakrishnan
Assistant Professor
Thesis Supervisor

Accepted by . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Implementation of a Power-Saving Protocol for Ad Hoc Wireless Networks

by

Kyle Jamieson

Submitted to the Department of Electrical Engineering and Computer Science
on February 4, 2002, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

We describe the design and implementation of a power-saving protocol for ad hoc wireless networks. We present the *Span* power-saving protocol and discuss its implementation in the context of the Linux operating system. We address the issues of ad hoc routing, link layer design, and integration with the Linux networking stack using the 802.11b wireless link technology. From this thesis, we conclude that Span can be implemented on an 802.11b network with reasonable performance for most networking applications. Furthermore, our implementation of Span yields a lifetime improvement of between 12% and 29% at each node in an ad hoc network. We argue that with additional hardware, Span can outperform conventional 802.11 ad hoc networks in terms of capacity, latency, and power savings.

Thesis Supervisor: Hari Balakrishnan
Title: Assistant Professor

Acknowledgments

I would first like to thank my research advisor, Hari Balakrishnan. His guidance has been an invaluable resource throughout all stages of this thesis. My discussions with him have opened up new insights, and on more than one occasion when I was deeply engrossed in intricate implementation details, discussion with him has enabled me to see the larger problem at hand.

My parents have given me love and support over the nineteen years I have been in school, and without them this thesis would not have been possible. I thank them for the sacrifices they made so that I could grow up in a learning environment. I also thank Kirsty, my sister, for lots of fun times.

The Span protocol is joint work with Benjie Chen, Robert Morris, and Hari Balakrishnan. I thank them for many interesting discussions about Span, ad hoc networks, and wireless radio in general.

I thank Barbara Liskov for the time I spent in the Programming Methodology Group that led to my coming to graduate school. Talking with her and her students inspired me to work on my own research. I also thank Chandra Boyapati and Miguel Castro from the Programming Methodology Group for many interesting discussions.

John Ankcorn provided many insights about radio design. He also taught me a bit about how the sales industry works. Dorothy Curtis was always available when I needed to purchase yet another wireless NIC for the testbed. Douglas De Couto and I shared our knowledge between the Grid and Span network testbeds. I would like to thank Laura Feeney from the Swedish Institute of Computer Science (SICS) for interesting discussions regarding 802.11 radios and power measurement techniques. Her work helped us fully understand what was happening in commercial 802.11 radios. Jamey Hicks from the Compaq Research Lab created the handhelds.org community and answered my questions about his ARM-Linux kernel ports. Many thanks to Eddie Kohler from ACIRI for the Click software, and for conducting Click/iPaq debugging sessions with me, over email. Thanks to Alex Snoeren for sharing experimental data on 802.11 and Bluetooth power consumption with me. Thanks also to Ken Steele for explaining compact flash on the iPaq to me in intricate detail.

My officemates Magdalena Balazinska, Allen Miu, and Godfrey Tan deserve thanks for putting up with complaints about the latest bug, and for helping me debug.

Finally, I would like to thank Matthew Burnside, Laura Diao, Michael Qin, and Yan Zhang for everything other than my thesis.

Contents

1 Introduction **11**
1.1 Motivating the use of Ad Hoc Networking 12
1.1.1 Applications for Ad Hoc Networks 12
1.2 The Case for Implementation 13
1.2.1 Wireless Network Simulation 13
1.3 Contributions 14

2 Reducing Power Consumption in an Ad Hoc Network **16**
2.1 The Problem 16
2.1.1 Problem Requirements 17
2.2 Wireless Radio Power Experiments 20
2.2.1 Methodology 20
2.2.2 Steady-State Power Consumption Experiments 21
2.2.3 State-Transition Experiments 21
2.3 The Span Protocol 23
2.3.1 Coordinator Announcement 24
2.3.2 Coordinator Withdrawal 26
2.4 Chapter Summary 27

3 The Span Implementation **29**
3.1 Architecture 29
3.1.1 Data Path 29
3.2 Span Layer 31
3.2.1 Coordinator Election 31
3.2.2 Span Encapsulation Header 33
3.2.3 Span Interface 33
3.3 Routing Architecture 33
3.3.1 Routing Layer Interface 33
3.3.2 Dynamically-Sequenced Distance Vector (DSDV) 35
3.4 Link Layer Design 37
3.4.1 IEEE 802.11 Distributed Coordination Function 39
3.4.2 IEEE 802.11 Ad Hoc Power-Saving Mode 40
3.4.3 Improving the 802.11 Ad Hoc Power-Saving MAC 41
3.5 Chapter Summary 43

4 Experimental Evaluation **44**
4.1 Coordinator Election Experiments 44
4.1.1 Test Topologies 44
4.1.2 Simulation Results 45
4.2 Latency Experiments 46
4.2.1 One-hop Latency 46
4.3 Energy Experiments 49
4.4 Capacity Experiments 50
4.5 Capacity of 802.11 Networks 52
4.6 Chapter Summary 53

5 Conclusions **54**
5.1 Related Work 54
5.1.1 Power-Saving in Ad Hoc Networks 54
5.1.2 Other Wireless Network Testbeds 56
5.2 Future Work 57

List of Figures

2-1 A state machine showing the possible radio states and transitions between these states. 16

2-2 Input current versus card state for typical 802.11 and Bluetooth technology implementations. The states are identical to those described above except for the Bluetooth *inquiry* state, which Bluetooth uses to form links between adjacent nodes in a Bluetooth ad hoc network. . . 18

2-3 A connected backbone does not necessarily preserve capacity. In this connected topology, black nodes are coordinators. Nodes that are within radio range of each other are connected by solid or dotted lines. Solid lines represent connections to and between coordinators. Packets between nodes 3 and 4 may contend for bandwidth with packets between nodes 1 and 2. On the other hand, if node 5 was a coordinator, no contention would occur. 19

2-4 Steady-state power consumption of the Cabletron Roamabout DS wireless network interface card, in various operating states. 21

2-5 Steady-state power consumption of the Cisco Aironet 340 series wireless network interface card, in various operating states. Note that the *off* state shown in this graph describes the card in non-powersaving mode with the MAC disabled. 22

2-6 *Left:* Instantaneous *idle*-to-*off* power consumption of the Cisco Aironet 340 series wireless network interface card. This graph shows that the idle-to-off hardware transition time is approximately 0.4 ms. *Right:* Instantaneous *off*-to-*idle* power consumption of the same card. This graph shows that the *off*-to-*idle* hardware transition time is approximately 20 ms. 23

2-7 Span is a protocol that operates under the routing layer and above the MAC and physical layers. The routing layer uses information Span provides, and Span takes advantage of any power saving features of the underlying MAC layer. 24

2-8 A scenario with 100 nodes, 19 coordinators, and a radio range of 250 meters. The nodes marked "∗" are coordinators; the nodes marked "+" are non-coordinator nodes. Solid lines connect coordinators that are within radio range of each other. 27

3-1 The overall architecture of the Span network testbed, shown in layers. The *SpanRoutingProtocol* box is an interface between Span and the underlying routing protocol and contains no functionality. 30
3-2 *Left:* the path of an incoming packet as it travels from the network interface to the *Span* layer of the networking stack. *Right:* the path of an outgoing packet as it travels from the *Span* layer of the networking stack to the network interface. Dotted lines indicate calls made in interrupt context; solid lines indicate calls made in process context. User space is shaded; kernel space is unshaded. 30
3-3 Coordinator announcement algorithm pseudo-code: a non-coordinator node periodically calls this routine to see if it should become a coordinator. 31
3-4 Coordinator announcement algorithm pseudo-code: returns the number of neighbor pairs a node can connect if it becomes a coordinator. 31
3-5 Coordinator announcement algorithm pseudo-code: returns true if and only if neighbors `a` and `b` are connected by one or two other coordinators. Note that coordinator lists are kept in the neighbor table. . . . 32
3-6 Encapsulation header for IP packets passing through the *Span* element. 32
3-7 The Span interface. The *Span* module exports this interface for the purpose of routing protocols. 33
3-8 C++ code specifying the interface for a Span-compliant routing protocol. A `NeighborEntry` contains IP and MAC addresses for a node's neighbor, and a `NeighborIterator` is an iterator that iterates over `NeighborEntry` elements. Code has been edited for clarity. 34
3-9 An example of a network where DSDV routes can fluctuate rapidly. In the figure the clouds represents sets of mobile hosts providing connectivity between edges into and out from the cloud. 36
3-10 The user-level Click configuration for our testbed experiments. Arrows in the diagram indicate packet flow between the boxes that represent click elements. The *EthRewriter*, *Span*, and *DSDV* elements are original contributions of this thesis. 38
3-11 *Left:* A node arrangement that illustrates the utility of the RTS/CTS handshake. In this figure Node 1 is sending data to Node 2. Each node's radio range is indicated by the dotted circles. *Right:* Data transmitted over the wireless medium and network allocation vector status as Node 1 sends a packet to Node 2. Time runs along the x-axis. White boxes represent data, and bold black lines represent a non-zero network allocation vector. 39
3-12 Two beacon periods' worth of traffic in the 802.11 ad hoc power-saving mode. We show traffic on the network (*top*), and (*bottom*) the on/off status of the radios at stations numbered one, two, and three. The bold areas of the timeline indicate the ATIM window. 41

3-13 Two beacon periods' worth of traffic in the 802.11 ad hoc power-saving mode, with our improvements. We show traffic on the network (*top*), and (*bottom*) the on/off status of the radios at stations numbered one, two, and three. The bold areas of the timeline indicate the ATIM window. 42

4-1 A four node test topology. In this and subsequent floorplan figures, black nodes indicate Span coordinators and white nodes indicate non-coordinators. We note that exactly the right coordinator is elected in this topology. 44

4-2 An eight node test topology. 45

4-3 An approximation to an optimal layout of coordinators in a 1000 meter × 1000 meter area. There are 14 coordinators in this layout. . . 45

4-4 Ideal and actual coordinator density as a function of node density. The ideal curve represents an approximate lower bound on the number of coordinators needed. Span elects more coordinators than the ideal case because of lower node density, coordinator rotation, and announcement collision. 46

4-5 A histogram showing the distribution of ping times over one hop between Span coordinators. The bin size is 70 ms. The mode of the histogram is the first bin, containing ping times between zero and 70 ms. 47

4-6 A histogram showing the distribution of ping times over one hop between nodes that are not Span coordinators. The bin size is 70 ms. The mode of the histogram is the 31st bin, which contains ping times between 2100 and 2170 ms. 48

4-7 A time diagram showing the latency incurred by one ping packet between a pair of Span non-coordinators in 802.11 ad hoc power-saving mode. The dashed lines indicate beacon period boundaries, while the dotted lines indicate the end of a traffic advertisement window and the beginning of a data window within one beacon period. 48

4-8 System energy remaining as a function of time, comparing Span an isolated Span coordinator versus an isolated Span non-coordinator. Each point represents a reading; readings were taken at five-minute intervals. 49

4-9 Total iPaq battery lifetime when a node is a Span coordinator or a non-coordinator. We separate the results based on whether the iPaq has its display backlight off or on. The result for a Span non-coordinator with its backlight off is calculated. 51

4-10 *Left:* a TCP trace between two Span non-coordinator nodes. *Right:* a TCP trace between two Span coordinators. In both figures the x-axis is the time that the packet was received, with $t = 0$ equal to the time the first byte was received. The y-axis is the sequence number of the last byte in the received packet. Each point on the graph represents the receipt of a 1024-byte packet. The dotted line shows the link bandwidth. 52

4-11 An example of the "two-hop matching" property. In this example, A is transmitting to B in the direction indicated by the arrow. Solid lines denote links available for use at the same time as this link while dotted lines denote links that may not be used concurrently with this link. . 53

List of Tables

2.1 Power consumption of various wireless network interface cards, as reported by Cisco corporation. Note that these results do not include idle power consumption. 22

Chapter 1

Introduction

A *wireless ad hoc network* is a system of autonomous mobile nodes that cooperatively route packets for each other. In such a network, nodes other than the packet source and destination participate in the delivery process. In this thesis, we are primarily concerned with ad hoc networks where power consumption is an issue, for instance because nodes are battery-powered. Battery technology is not improving at a rapid rate, so power consumption is likely to remain an issue in mobile wireless networks.

Since power is often an important issue in many ad hoc networks, there have been many proposals to incorporate power-awareness into the routing layer [5] or link layer [19, 25, 26, 28, 31, 34] of an ad hoc network. However, several experiments show that the relatively high idle radio power consumption in many technologies including 802.11b[1] [10, 32] plays a very significant role in the lifetime of a node in an ad hoc network. With this in mind, we designed *Span* [6], a protocol that saves energy in dense ad hoc networks by turning most nodes' radios off most of the time, while preserving capacity.

In previous work [6], we simulated the Span protocol without implementing Span in a real ad hoc network. The main contribution of this thesis is a demonstration that the Span protocol is feasible in a real implementation. In this thesis, we describe an implementation in the Linux operating system for handheld computers.

For our hardware platform, we chose the Compaq iPaq H3670 pocket PC coupled with Cisco Aironet 340 series wireless network interface cards. Our implementation has the following desirable features.

1. By construction, it is not simulated. All real-world problems such as unexpected radio propagation patterns, interference, etc., affect our testbed.

2. It scales to large numbers of real-life computers. We have developed the infrastructure needed to program an iPaq to be a Span node in a few seconds.

3. The power model is realistic, because we constructed the network with real 802.11 radio cards and Linux computers.

[1]We abbreviate the 802.11b link-layer as "802.11" throughout the rest of this thesis.

1.1 Motivating the use of Ad Hoc Networking

A second goal of this thesis is to motivate the use of ad hoc networks. By constructing a testbed of real computers that route packets along multi-hop paths, we hope to enable and encourage the development of interesting applications that take advantage of the ad hoc paradigm. For example, Buttyan and Hubaux [4] have proposed that people willing to participate in an ad hoc network forward each others' packets in exchange for a form of currency. Our implementation can serve as a platform in which such interesting ideas can be tested.

There is great interest in the mobile computing community in ad hoc networking. Researchers have proposed the following list of applications for ad hoc networks, and are also discussing the possibility of commercial applications for ad hoc networking. Recently at the 2001 ACM/IEEE Conference for Mobile Computing and Applications (Mobicom 2001), a panel of researchers discussed the question "Will there ever be a commercial market for ad hoc networks?"

1.1.1 Applications for Ad Hoc Networks

The following is a list of possible applications for ad hoc networking.

1. *Military battlefield scenarios* in which soldiers carry portable or wearable computers with radios for communication and/or planning.

2. *Post-disaster rescue efforts* in which any existing wireless infrastructure is disabled or completely destroyed. In this scenario, rescue workers would use either handheld or wearable computers to create a wireless ad hoc network, for the purpose of communication and coordination of rescue efforts.

3. *Sensor networks* in which many nodes scattered about the physical environment communicate information about the environment. Many sensor networks share the same multi-hop topology, routing, and power-saving issues as the ad hoc networks of the above two examples.

4. *Temporary collaborations among colleagues.* While infrastructure wireless technology (most notably 802.11) has become a commodity in today's academic and business campuses, it is clear that not all areas of a school or workplace can or should be covered by base stations. Long hallways, machine rooms, outdoor pathways, and sports fields all hold the lowest priority for IT managers deploying wireless base stations. Yet, people do need to work in such areas, and ad hoc networks enable connectivity between small numbers of people for short periods of time.

5. *Rooftop networks*, proposed in [30], deploy small radio units on top of homes. Homes communicate with each other to provide Internet connectivity via the ad hoc network formed by the consumer-owned radio units. Rooftop networks have been proposed as an alternative to traditional wired ISPs for residences.

6. *Vehicular networks* [20] are made of ad hoc nodes placed in automobiles. Morris et. al. suggest several interesting applications for such a network: location-directed multicast, traffic congestion monitoring, fleet tracking, over-the-horizon police radar detection, and inter-vehicle chat.

7. *Cell phone call routing* may benefit from ad hoc networking technology. Although the cellular infrastructure is well-established and covers all metropolitan areas and most suburban and well-populated rural areas, capacity is an issue during an event that brings a large crowd of people together in one geographic location. ad hoc networking techniques may provide the answer to this problem, routing calls away from busy cells, to provide connectivity for more users than would be possible without such techniques.

8. *"Beacon Networks."* Systems such as Cricket [24] utilize *beacons*, small devices capable of transmitting and receiving both RF and ultrasound signals. To ease deployment, beacons are battery-powered, necessitating some form of power conservation.

9. *Personal-area networks.* Wireless networking technology designed to facilitate communication between handheld and wearable devices exists today in the form of Bluetooth [13]. Since personal-area networks are by definition portable, any personal-area networking technology needs to conserve power.

1.2 The Case for Implementation

In this section we argue that claims made about any ad hoc networking protocol are ultimately best evaluated in a real implementation. We start with a discussion of wireless network simulators.

1.2.1 Wireless Network Simulation

To date, most ad hoc networking protocols have been designed and evaluated in simulation. The most popular simulation environment in the academic community has been the *ns-2* simulator [22], augmented with the CMU Monarch extensions for wireless radio propagation and emulation of the 802.11 protocol. Many researchers, including the author, have extended the Monarch package to include support for an energy model.

Advantages of Simulation

Simulation has many advantages. When the research community standardizes on a simulation platform, as it has done with *ns-2*, protocols can be compared with each other on fair terms. For the same reason, simulation offers great ease of coding, since researchers are already familiar with how to integrate their protocols with the *ns-2* stack. Finally, simulation yields a large fraction of reused code, since most proposals for ad hoc networking protocols do not change most of the networking stack.

Drawbacks of Simulation

However, simulation encourages researchers to make several assumptions about how wireless radios work. We first define two terms in order to discuss these assumptions.

Reception range is the maximum (approximate) distance at which a transmission may be received, given fixed transmission and receive powers and radio designs.

Interference range is the range over which a transmission interferes with another transmission. Note that the interference range will always be larger than the receive range, and that two or more interfering transmissions can add up to interfere with a third transmission.

Much work published over the last few years makes most or all of the following assumptions about the world nodes in an ad hoc network inhabit.

1. *Symmetric radio propagation patterns.* The radio propagation model in *ns-2* with Monarch extensions assumes that the reception and interference ranges of the radio is roughly constant in all directions, which leads to an idealized circular reception range and interference range.

2. *A two-dimensional world.* Nodes in *ns-2* and other simulators move in a two-dimensional world, while real ad hoc networks may occupy several floors of a building, for example.

3. *Random node movement.* Nodes in most *ns-2* mobile simulations move using a *random waypoint* model: every few seconds, a node picks a random destination and speed, and moves to the destination it chose at the speed it chose. This is obviously not a realistic mobility model. The way to generate a realistic mobility model is to either use real people (or cars, boats) in a testbed, or use a trace from a testbed, cell phone localization data, or GPS data.

4. *Energy models.* Different radios require different amounts of power to transmit, receive, and listen for packets. Although *ns-2* and other simulators use real cards' parameters, such parameters are constantly changing with each new release of a vendor's chipset. The surest way to evaluate a new protocol is to implement it on real hardware.

Consequently, we believe that claims made about ad hoc networks are ultimately best evaluated in an implementation. Due to its highly configurable nature, we believe that our testbed implementation can serve as a starting point for the easy implementation and evaluation of other researchers' ad hoc networking schemes.

1.3 Contributions

The main contribution of this thesis is a demonstration that the Span protocol is feasible in a real implementation. We contribute methodologies for organizing an ad hoc networking stack so that the underlying ad hoc routing protocol may easily integrate with Span, and a software implementation of an 802.11 ad hoc power-saving MAC layer.

Thesis Outline. Chapter 2 of this thesis describes the Span protocol in detail, formulating the theoretical background to the problem of power-saving in an ad hoc network. Chapter 3 details our implementation of Span, including our contributions to the overall ad hoc networking stack and the MAC layer. Chapter 4 presents the experimental results we obtained that evaluate our implementation. Chapter 5 concludes, presenting future directions for theoretical analysis and related work.

Chapter 2

Reducing Power Consumption in an Ad Hoc Network

This chapter discusses techniques for reducing power consumption of battery-powered, wireless devices participating in an ad hoc network. We begin by discussing why and how wireless devices consume energy in an ad hoc network.

2.1 The Problem

Minimizing energy consumption is an important challenge in mobile networking. Significant progress has been made on low-power hardware design for mobile devices, so the wireless network interface is often a device's single largest consumer of power.

The radio in a wireless device operates in one of several states: an *xmit* state where the radio is actively transmitting data over a wireless link, a *receive* state where the radio is receiving data from the wireless link and passing it up to the software layers of the mobile device, an *idle* state where the radio is listening for data but not passing any information up to the wireless device, and frequently, a *sleep* state where the radio is not fully powered-up and cannot detect any incoming data. Figure 2-1 shows the possible transitions between states; note the lack of a state transition between *sleep* and *xmit* or *receive*. This is because a sleeping radio cannot detect any incoming packets, nor can it transmit immediately—it must wake up to the *idle* mode first.

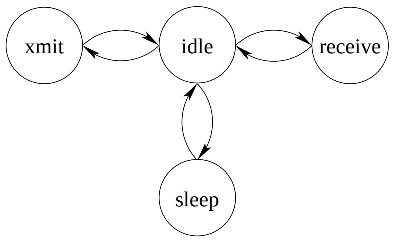


Figure 2-1: A state machine showing the possible radio states and transitions between these states.

The total energy cost of running the radio is equal to

$$\int\limits_{\tau_{xmit}(t)} P_{xmit}\,dt + \int\limits_{\tau_{recv}(t)} P_{recv}\,dt + \int\limits_{\tau_{idle}(t)} P_{idle}\,dt + \int\limits_{\tau_{sleep}(t)} P_{sleep}\,dt \tag{2.1}$$

where $\tau_x(t)$ is the time interval over which the radio is in state x and P_x is the power required for the radio to be in state x. From the Friss free-space and two-ray ground propagation models [27], it is well-known that the power required to transmit is

$$P_{xmit} = \begin{cases} \frac{(4\pi)^2}{G_t G_r \lambda^2} P_{recv-thresh} d^2 & : \quad d < d_{crossover} \\ \frac{(4\pi)^4}{G_t G_r h_t^2 h_r^2} P_{recv-thresh} d^4 & : \quad d \geq d_{crossover} \end{cases} \tag{2.2}$$

where G_t is the gain of the transmitting antenna, G_r is the gain of the receiving antenna, λ is the wavelength of the carrier frequency, $P_{recv-thresh}$ is the minimum power permissible at the receiver, h_t and h_r are the respective heights of the transmitting and receiving antennas above the ground, and $d_{crossover}$ is the distance below which the two-ray ground propagation model applies and above which the Friss free-space model applies. From Equation 2.2 above we see that transmit power is related to distance by a power law, dependent on distance.

Equation 2.2 dictates the approach we take to saving power in ad hoc networks. The basic intuition behind our approach is as follows. If the radio's transmission range is large (as in a cellular phone), then Equation 2.2 tells us that transmit power will be significantly larger than either idle or receive power (i.e., $P_{xmit} \gg P_{recv}$). Conversely, if the radio's transmission range is small (as in a Bluetooth or 802.11 radio), then transmission power will not be large compared to receive or idle power. We note here that in many useful ad hoc networks, the transmission range is relatively small ($\approx$ 100 meters).

Figure 2-2[1] shows a power study that compares the power consumption of 802.11 and Bluetooth cards. These results validate the claims made above about the relative power consumption of networking interface cards where the transmission range is relatively small. Note that in a Bluetooth ad hoc network, nodes would need to periodically enter the *inquiry* state, leading to increased idle energy consumption.

The *coordination problem* in wireless, ad hoc, shared-medium networks is to find a way for most of the radios to operate in the *sleep* state of Figure 2-1 most of the time, while the remaining nodes operate in the *idle* state and deliver packets to the sleeping nodes. Furthermore, the nodes that sleep most of the time must coordinate with the nodes in the *idle* state to arrange for packet delivery.

2.1.1 Problem Requirements

A good power-saving coordination technique for wireless ad-hoc networks ought to fulfill the following requirements. These requirements form the design goals for Span.

[1]These results are joint work with Alex Snoeren of the Networks and Mobile Systems Group at the MIT Laboratory for Computer Science.

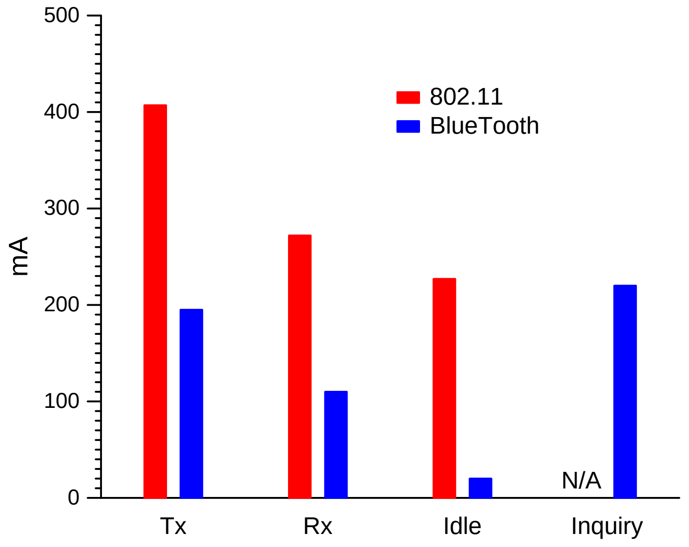


Figure 2-2: Input current versus card state for typical 802.11 and Bluetooth technology implementations. The states are identical to those described above except for the Bluetooth *inquiry* state, which Bluetooth uses to form links between adjacent nodes in a Bluetooth ad hoc network.

1. A radio must run most of its electronics to listen for packets. When a packet actually arrives for the host, the amount of energy required to pass the data up to the host is small compared to the amount of energy required to listen for packets in the first place. In other words, P_{idle} is not much smaller than P_{recv}. The consequence of this is that in a non-Span network, idle energy consumption will dominate receive energy consumption, since the radio spends much of its time in idle mode, waiting for packets to receive. Therefore, a good power-saving technique should allow as many nodes as possible to put their radio receivers into the *sleep* state most of the time.

2. On the other hand, a good coordination technique should forward packets between any source and destination with minimally-more delay than if all nodes were in the *idle* state. This implies that enough nodes must stay *idle* to form at least a connected backbone of *idle* nodes.

3. The algorithm for picking this backbone should be distributed, requiring each node to make a local decision about its radio's power state.

4. Furthermore, the backbone formed by the *idle* nodes should provide about as much total capacity as the original network, since otherwise congestion may increase. This means that paths that could operate without interference in the original network should be represented in the backbone. For example, Figure 2-3 illustrates a topology that violates this principle. In this topology, black nodes are coordinators. Nodes that are within radio range of each other are connected by solid or dotted lines. Packets between nodes 3 and 4 may contend for bandwidth with packets between nodes 1 and 2 (solid arrows). On the other

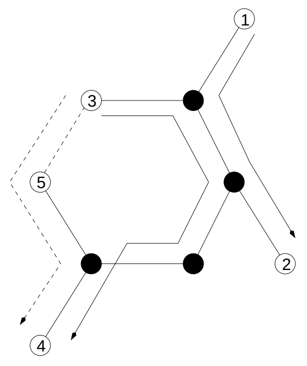


Figure 2-3: A connected backbone does not necessarily preserve capacity. In this connected topology, black nodes are coordinators. Nodes that are within radio range of each other are connected by solid or dotted lines. Solid lines represent connections to and between coordinators. Packets between nodes 3 and 4 may contend for bandwidth with packets between nodes 1 and 2. On the other hand, if node 5 was a coordinator, no contention would occur.

hand, if node 5 was a coordinator, node 3 can send packets to node 4 via the path shown by the dotted arrow, and no contention would occur.

5. A good coordination technique should not make many assumptions about the link layer's facilities for sleeping; it should work with any link-layer that provides for sleeping and periodic polling, including 802.11's ad-hoc power saving mode.

6. Finally, power saving should inter-operate correctly with whatever routing system the ad-hoc network uses.

With these requirements in mind, we now consider some proposed approaches to saving power in an ad hoc network. One might outfit nodes with power-controllable radios, and then devise a distributed algorithm to lower the transmit power at each radio so that the network is still connected, yet saves power. This is termed the "minimum-energy routing" approach [30]. This approach requires more complicated radios, capable of adjusting their transmit power. While the minimum-energy routing approach satisfies all but our first requirement for an energy-saving protocol, idling radio interfaces consume a significant amount of power. In a minimum-energy routing network, the idle energy of a radio still dominates its energy consumption.

Alternatively, one might select a few nodes to perform the bulk of the communication in the network. This is exactly what Heinzelman et. al's LEACH protocol [14] does: rotating cluster-heads send information to the base station in a wireless sensor network. The other nodes in the sensor network choose the closest cluster head and send their information to it. This approach makes sense in a sensor network, but does not immediately generalize to a multi-hop ad hoc network where traffic patters could be markedly different, nodes might be out of direct communication range, and capacity and latency requirements are of concern.

We consider a multi-hop ad hoc network with fairly primitive, fixed-transmission power radios. We also assume that the transmission range has been optimized for shorter distances, as is true in the commodity 802.11 and Bluetooth radios. We propose a distributed algorithm to switch off a subset of nodes such that the remaining nodes form a connected spanning tree over the network. This addresses our first observation by limiting most of the communication to nodes far away from each other, since if nodes are too close to each other, they will switch off their radios. Our approach addresses our second observation by limiting the amount of time a node spends in idle mode, thus reducing impact of the idle term on radio energy consumption. The challenge is thus to design a technique whereby nodes can coordinate with each other to form a backbone of awake nodes over which data can flow in the network.

2.2 Wireless Radio Power Experiments

We now present the results of an experimental verification of the claims made in the beginning of Section 2.1. Span requires that a node in an ad hoc network turn its radio off periodically to save power. To determine the feasibility of a Span implementation in a real ad hoc network, we conducted experiments where we examined both the steady-state power consumption and instantaneous transitional power consumption of various network interface cards. Our results agree with those of a similar study presented by Feeney [10].

2.2.1 Methodology

To accurately model energy consumption, we took measurements of the Cabletron Roamabout 802.11 DS High Rate network interface card (NIC). We used an IBM ThinkPad 560 running the Linux 2.2.16 kernel with the `wavelan_cs` drivers included in the RedHat 6.2 Linux distribution.

We followed the methodology outlined in [32]. To measure power consumed by the NIC, we disconnected the battery from the portable computer, and measured the voltage V_R across a 2.2 Ω resistor placed in series with the NIC on the PC Card bus. From this, we calculated the current I going into the NIC as $I = V_R/2.2$ amps. We also measured the voltage across the NIC. The steady-state voltage across the NIC was constant, even under network load, at 4.94 V. We then calculated the instantaneous power P consumed by the NIC as $P = 4.94\,\mathrm{V} \cdot I$.

For the steady-state power consumption measurements, we obtained the *receive* state measurement by putting the card into non-power saving mode, and measuring the power required to listen for a packet, decode it, and pass its contents up to the host. The *idle* state measurement was obtained in the same manner, but measuring only the power required to listen for a packet. In contrast, the *sleep* state measurement was obtained by putting the card into power saving mode, and measuring the average (lower, and near-constant) power consumption during the part of the power saving cycle where the card was not listening for packets. The key point is the experimental confirmation of the large difference between the power consumption of the *idle* and *sleeping* states.

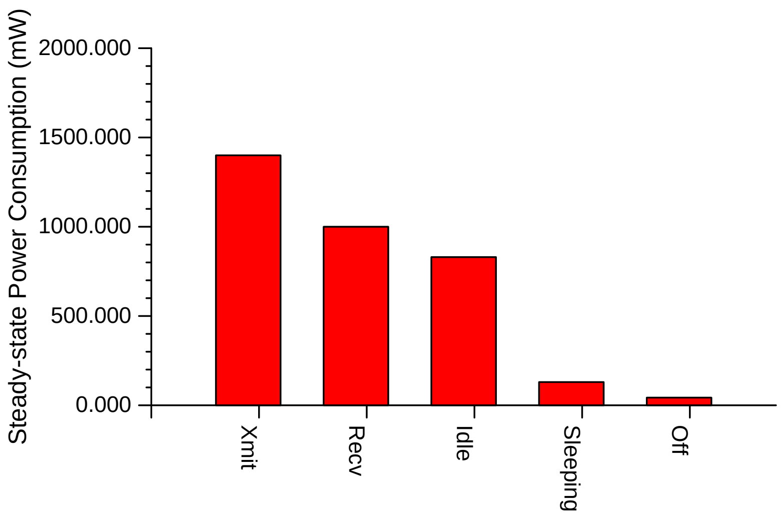


Figure 2-4: Steady-state power consumption of the Cabletron Roamabout DS wireless network interface card, in various operating states.

2.2.2 Steady-State Power Consumption Experiments

We summarize the time-averaged, steady-state results in Figure 2-4, and note that these closely match the results obtained by Feeney and Nilsson [10] for similar 802.11 network interface cards in ad hoc mode. We repeated the same experiment with the Cisco Aironet 340 series network interface card operating as a base-station (see Section 3.4 for an explanation of the different operating modes of the Cisco Aironet 340 series cards). Note that for the Cisco card, the *sleep* state is not available when the card is in base-station mode. Consequently, the state labeled *off* in the Cisco experiment is a state where the card's MAC is disabled, but the card itself is not fully powered-down nor in a true sleeping state. The results for the Cisco experiment are shown in Figure 2-5.

2.2.3 State-Transition Experiments

We now present the methodology and experimental results of our experiments to evaluate the energy and latency penalties of transitioning the card between the *sleep* and *idle* states. These experiments are important because if the energy penalty for transitioning the card between the *sleep* and *idle* states is too high, our technique will not save power. If the latency penalty of transitioning the card is too high our technique will incur high latency and capacity penalties.

Our first experiments are with the Lucent WaveLAN card. We measured this card in managed, power-saving mode (i.e., as a client of a base station). We introduce a new state, *off*. This state is functionally-equivalent to the *sleep* state, except in most hardware implementations, the card is not designed to be rapidly switched between *off* and *idle*. In this mode, the hardware transitions the card between the *sleep* and

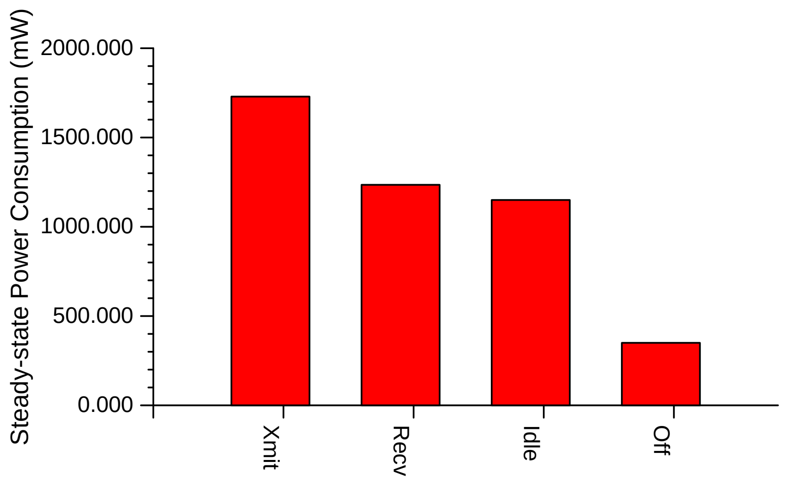


Figure 2-5: Steady-state power consumption of the Cisco Aironet 340 series wireless network interface card, in various operating states. Note that the *off* state shown in this graph describes the card in non-powersaving mode with the MAC disabled.

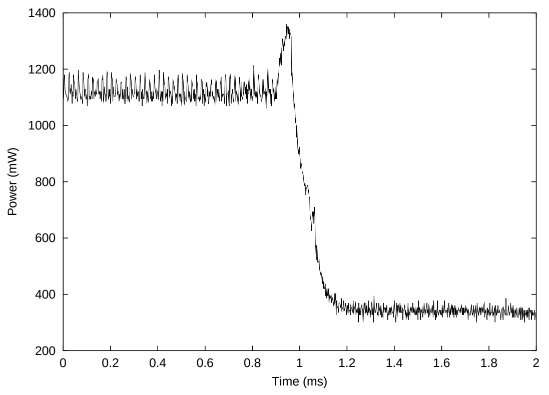
State Energy (mW)	Cisco Aironet 340 Series	Apple AirPort	Compaq WL100	Lucent Orinoco	3Com AirConnect
Transmit	1729	1729	914	1408	2420
Receive	1235	1482	914	914	1087
Sleep	49	296	N/A	44	188

Table 2.1: Power consumption of various wireless network interface cards, as reported by Cisco corporation. Note that these results do not include idle power consumption.

idle states, and consequently the card is very efficient. We measured that 0.60 mJ was consumed (beyond that required for sleeping over the same time period) when moving from the *sleep* state to the *idle* state. This transition takes 0.4 ms.

These figures are very low, and they show that when implemented in hardware, the penalty for switching card state is quite low. Compared to the following measurements for the software-controlled Cisco Aironet 340 series card, they show that the penalty for software control of the card is high both in terms of latency and energy. Figure 2-6 shows the instantaneous power consumption required to transition the Cisco card from *idle* to *off* and *off* to *idle*, respectively.

Finally, we present figures from a Cisco study [8] on the power consumption of various network interface cards. These numbers validate our previous measurements, and follow the trends noted earlier in this chapter. Note that the *sleep* state in this table is hardware-implemented, while the *off* states in Figure 2-6 are implemented in software, and so consume substantially more energy.



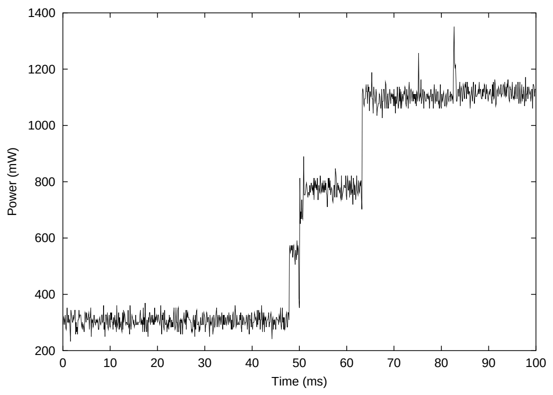


Figure 2-6: *Left:* Instantaneous *idle*-to-*off* power consumption of the Cisco Aironet 340 series wireless network interface card. This graph shows that the idle-to-off hardware transition time is approximately 0.4 ms. *Right:* Instantaneous *off*-to-*idle* power consumption of the same card. This graph shows that the *off*-to-*idle* hardware transition time is approximately 20 ms.

2.3 The Span Protocol

Span adaptively elects "coordinators" from all nodes in the network. Span coordinators stay *idle* continuously and perform multi-hop packet routing within the ad hoc network, while other nodes remain in power-saving mode and periodically check if they should become a coordinator.

In the previous section, we have shown how Span is feasible in terms of hardware implementation. Still, the Span approach is not straightforward: a node must arrange to become *idle* not just to send packets, but also to receive packets addressed to it and to participate in any higher-level routing and control protocols.

The algorithm presented in this chapter, *Span,* fulfills the above requirements. Each node in the network running Span makes periodic, local decisions on whether to transition to *sleep* or stay awake as a *coordinator* and participate in the forwarding backbone topology. To preserve capacity, a node volunteers to be a coordinator if it discovers, using information it gathered from local broadcast messages, that two of its neighbors cannot communicate with each other directly or through one or two existing coordinators. To keep the number of redundant coordinators low and rotate this role among all nodes, each node delays announcing its willingness by a random time interval that takes two factors into account: the amount of remaining battery energy, and the number of pairs of neighbors it can connect together. This combination ensures, with high probability, a capacity-preserving connected backbone of coordinators at any point in time, where nodes tend to consume energy at about the same rate. Span does all this using only local information, and consequently scales well with the number of nodes. Our simulation results, with energy parameters from measurements of 802.11 wireless interfaces, show that system lifetime with Span is more than a factor of two better than without Span, for a range of node densities, without much reduction in overall forwarding capacity.

Span achieves four goals. First, it ensures that enough coordinators are elected

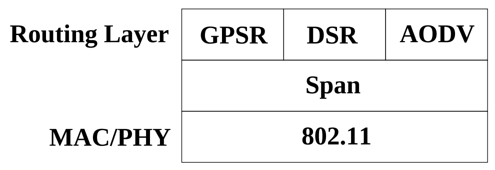


Figure 2-7: Span is a protocol that operates under the routing layer and above the MAC and physical layers. The routing layer uses information Span provides, and Span takes advantage of any power saving features of the underlying MAC layer.

so that every node is in radio range of at least one coordinator. Second, it rotates the coordinators in order to ensure that all nodes share the task of providing global connectivity roughly equally. Third, it attempts to minimize the number of nodes elected as coordinators, thereby increasing network lifetime, but without suffering a significant loss of capacity or an increase in latency. Fourth, it elects coordinators using only local information in a decentralized manner—each node only consults state stored in local routing tables during the election process.

Span is proactive: each node periodically broadcasts `HELLO` messages that contain the node's status (i.e., whether or not the node is a coordinator), its current coordinators, and its current neighbors. From these `HELLO` messages, each node constructs a list of the node's neighbors and coordinators, and for each neighbor, a list of its neighbors and coordinators.

As shown in Figure 2-7, Span runs above the link and MAC layers and interacts with the routing protocol. This structuring allows Span to take advantage of power-saving features of the link layer protocol, while still being able to affect the routing process. For example, non-coordinator nodes can periodically become *idle* and listen (as in the 802.11 power-saving mode [1]) or poll (as in LPMAC [19]) for their packets. Span leverages a feature of modern power-saving MAC layers, in which if a node has been in the *sleep* state for a while, packets destined for it are not lost but are buffered at a neighbor. When the node awakens, it can retrieve these packets from the buffering node, typically a Span coordinator. Span also requires a modification to the route lookup process at each node—at any time, only those entries in a node's routing table that correspond to currently active coordinators can be used as valid next-hops (unless the next hop is the destination itself).

A Span node switches state from time to time between being a coordinator and being a non-coordinator. A node includes its current Span state in its `HELLO` messages. The following sections describe how a node decides that it should announce that it is a coordinator, and how it decides that it should withdraw from being a coordinator.

2.3.1 Coordinator Announcement

Periodically, a non-coordinator node determines if it should become a coordinator or not. The following *coordinator eligibility rule* in Span ensures that the entire network is covered with enough coordinators:

Coordinator eligibility rule: A non-coordinator node should become a coordinator if it discovers, using only information gathered from local broadcast messages, that two of its neighbors cannot reach each other either directly or via one or two coordinators.

This election algorithm does not yield the minimum number of coordinators required to merely maintain connectedness. However, it roughly ensures that every populated radio range in the entire network contains at least one coordinator. Because packets are routed through coordinators, the resulting coordinator topology should yield good capacity.

Announcement contention occurs when multiple nodes discover the lack of a coordinator at the same time, and all decide to become a coordinator. Span resolves contention by delaying coordinator announcements with a randomized backoff delay. Each node chooses a delay value, and delays the `HELLO` message that announces the node's volunteering as a coordinator for that amount of time. At the end of the delay, the node reevaluates its eligibility based on `HELLO` messages recently received, and makes its announcement if and only if the eligibility rule still holds.

We consider a variety of factors in our derivation of the backoff delay. Consider first the case when all nodes have roughly equal energy, which implies that only topology should play a role in deciding which nodes become coordinators. Let N_i be the number of neighbors for node i and let C_i be the number of additional pairs of nodes among these neighbors that would be connected if i were to become a coordinator and forward packets. Clearly, $0 \leq C_i \leq \binom{N_i}{2}$. We call $\frac{C_i}{\binom{N_i}{2}}$ the *utility* of node i. If nodes with high C_i become coordinators, fewer coordinators in total may be needed in order to make sure every node can talk to a coordinator; thus a node with a high C_i should volunteer more quickly than one with smaller C_i.

If there are multiple nodes within radio range that all have the same utility, Span prevents too many of them becoming coordinators. This is because such coordinators would be redundant—they would not increase system capacity, but simply drain energy. If the potential coordinators make their decisions simultaneously, they may all decide to become coordinators. If, on the other hand, they decide one at a time, only the first few will become coordinators, and the rest will notice that there are already enough coordinators and go back to sleep. To handle this, we use a randomized "slotting-and-damping" method reminiscent of techniques to avoid multiple retransmissions of lost packets by multicast protocols, such as XTP [7], IGMP [11] and SRM [12]: the delay for each node is randomly chosen over an interval proportional to $N_i \times T$, where T is the round-trip delay for a small packet over the wireless link.

Thus, when all nodes have roughly equal energy, the above discussion suggests a backoff delay of the form:

$$delay = \left(\left(1 - \frac{C_i}{\binom{N_i}{2}}\right) + R\right) \times N_i \times T \tag{2.3}$$

The randomization is achieved by picking R uniformly at random from the interval $(0, 1]$.

Consider the case when nodes may have unequal energy left in their batteries. We observe that what matters in a heterogeneous network is not necessarily the absolute amount of energy available at the node, but the amount of energy scaled to the maximum amount of energy that the node can have. Let E_r denote the amount of energy (in Joules) at a node that still remains, and E_m be the maximum amount of energy available at the same node. A reasonable (but not the only) notion of fairness can be achieved by ensuring that a node with a larger value of E_r/E_m is more likely to volunteer to become a coordinator more quickly than one with a smaller ratio. Thus, we need to add a decreasing function of E_r/E_m that reflects this, to Equation 2.3. There are an infinite number of such functions, from which we choose a simple linear one: $1 - E_r/E_m$. In addition to its simplicity, this choice is attractive because it ensures that the rate with which a node reduces its propensity to advertise (as a function of the amount of energy it has left), is constant. (We experimented with a few other functions, including an exponentially decaying function of E_r/E_m and an inversely decaying function of E_r/E_m; the simple linear one worked best.)

Combining this with Equation 2.3 yields the following equation for the backoff delay in Span:

$$delay = \left(\left(1 - \frac{E_r}{E_m}\right) + \left(1 - \frac{C_i}{\binom{N_i}{2}}\right) + R \right) \times N_i \times T \tag{2.4}$$

Observe that the first term does not have a random component; thus if a node is running low on energy, its propensity to become a volunteer is guaranteed to diminish relative to other nodes in the neighborhood with similar neighbors.

In a network with uniform density and energy, our election algorithm rotates coordinators among all nodes of the network. It achieves fairness because the likelihood of becoming a coordinator falls as a coordinator uses up its battery. In practice, however, ad hoc networks are rarely uniform. Our announcement rule adapts to non-uniform topology: a node that connects network partitions together will always be elected a coordinator. This property preserves capacity over the lifetime of the network. Because of Span's emphasis on capacity-preservation to the extent possible, such critical nodes will unavoidably die before other less-critical ones. However, in a mobile Span network, a given node is rarely stuck in such a position, and this improves fairness dramatically.

2.3.2 Coordinator Withdrawal

Each coordinator periodically checks if it should withdraw as a coordinator. A node should withdraw if every pair of its neighbors can reach each other either directly or via one or two other coordinators. In order to also rotate the coordinators among all nodes fairly, after a node has been a coordinator for some period of time, it marks itself as a tentative coordinator if every pair of neighbor nodes can reach each other via one or two other neighbors, even if those neighbors are not currently coordinators. A

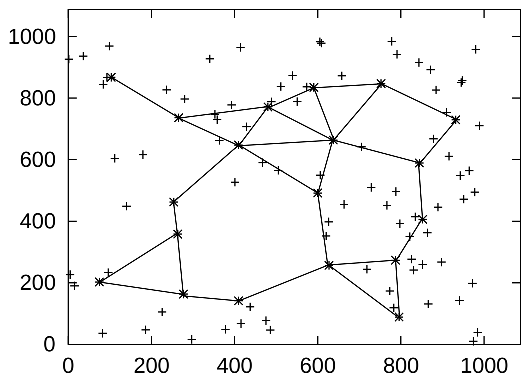


Figure 2-8: A scenario with 100 nodes, 19 coordinators, and a radio range of 250 meters. The nodes marked "$*$" are coordinators; the nodes marked "+" are non-coordinator nodes. Solid lines connect coordinators that are within radio range of each other.

tentative coordinator can still be used to forward packets. However, the coordinator announcement algorithm described above treats a tentative coordinator as a non-coordinator. Thus, by marking itself as tentative, a coordinator gives its neighbors a chance to become coordinators. A coordinator stays tentative for W_T amount of time, where W_T is the maximum value of Equation 2.4. That is,

$$W_T = 3 \times N_i \times T \tag{2.5}$$

If a coordinator has not withdrawn after W_T, it clears its tentative bit. To prevent an unlucky low energy node from draining all of its energy once it becomes a coordinator, the amount of time a node stays as a coordinator before turning on its tentative bit is proportional to the amount of energy it has (E_r/E_m).

While Span uses local `HELLO` messages to propagate topology information, it does not depend on them for correctness: when `HELLO` messages are lost, Span elects more coordinators, but does not disconnect the backbone. Figure 2-8 shows the result of our election algorithm at a random point in time on a network of 100 nodes in a 1000 meter $\times$ 1000 meter area, where each radio has an isotropic circular range with a 250 meter radius. Solid lines connect coordinators that are within radio range of each other.

2.4 Chapter Summary

This chapter presented *Span*, a distributed coordination technique for multi-hop ad hoc wireless networks that reduces energy consumption without significantly diminishing the capacity or connectivity of the network. Span adaptively elects *coordinators* from all nodes in the network, and rotates them in time. Span coordinators stay awake and perform multi-hop packet routing within the ad hoc network, while other nodes remain in power-saving mode and periodically check if they should awaken and become a coordinator.

With Span, each node uses a random backoff delay to decide whether to become a coordinator. This delay is a function of the number of other nodes in the neighborhood that can be bridged using this node, and the amount of energy it has remaining. Our results show that while Span not only preserves network connectivity, it also preserves capacity, decreases latency, and provides significant energy savings. For example, for a practical range of node densities and a practical energy model, our simulations show that the system lifetime with Span is more than a factor of two better than without Span.

Chapter 3

The Span Implementation

This chapter details our implementation of Span in an experimental network testbed of Compaq iPaqs running Linux, connected via Cisco Aironet 340 wireless network interfaces. We discuss the issues we faced implementing this network testbed, and the tools we used to leverage existing work in this area. There were two main challenges we faced while implementing Span. The first challenge was the efficient integration of Span with the underlying ad hoc routing protocol. Both Span and the underlying ad hoc routing protocol use periodic `HELLO` messages, so it is not immediately clear if one should send a `HELLO` message or wait for the other entity to send its `HELLO` message and then piggyback data on the outgoing packet. The second challenge is in the link layer. Due to lack of hardware support for 802.11 ad hoc PSM, we implemented our own PSM in software. Designing our PSM for correctness, reliability, and performance was a major effort in this thesis.

Chapter Outline. Section 3.1 of this chapter summarizes the architecture of our implementation, breaking the discussion down into details of the control and data paths. Section 3.2 discusses the details of how the Span protocol itself is implemented. Section 3.3 presents the design of the underlying routing algorithm we implemented, and discusses the design of any Span-aware routing algorithm. Section 3.4 presents the design of the software-based 802.11 ad hoc PSM MAC. We conclude in Section 3.5.

3.1 Architecture

Figure 3-1 shows the overall architecture of the software running on a Span node. In the subsequent sections of this chapter, we will discuss the design of each layer in turn.

3.1.1 Data Path

In this section, we discuss the data path of a Span node between the layers marked *802.11* and *Span* in Figure 3-1 above. Section 3.2 discusses the control path.

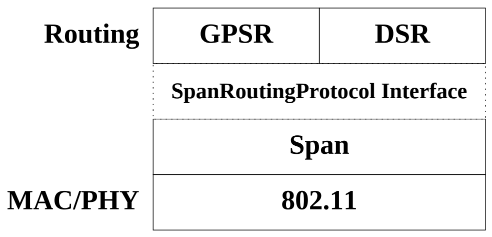


Figure 3-1: The overall architecture of the Span network testbed, shown in layers. The *SpanRoutingProtocol* box is an interface between Span and the underlying routing protocol and contains no functionality.

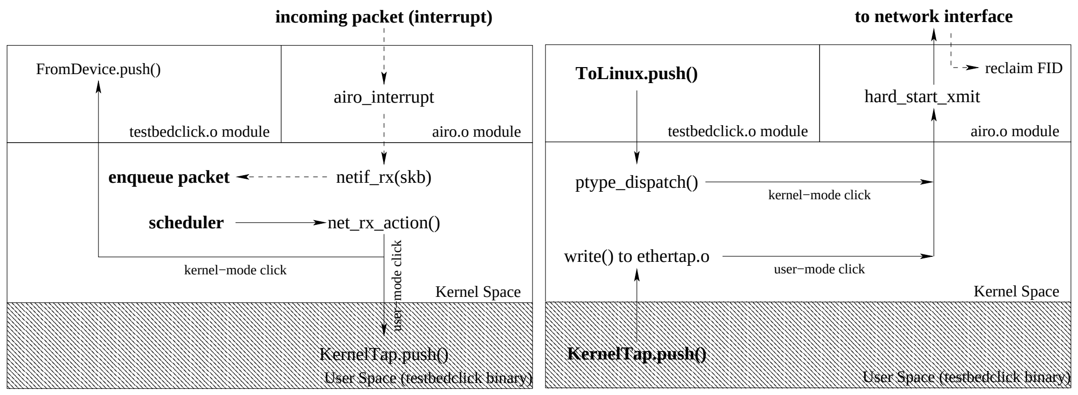


Figure 3-2: *Left:* the path of an incoming packet as it travels from the network interface to the *Span* layer of the networking stack. *Right:* the path of an outgoing packet as it travels from the *Span* layer of the networking stack to the network interface. Dotted lines indicate calls made in interrupt context; solid lines indicate calls made in process context. User space is shaded; kernel space is unshaded.

In Figure 3-2 (Left), we see the path that a packet takes upon reception through the wireless network interface. When Linux calls the interrupt handler `airo_interrupt`, `airo_interrupt` queues the packet in the Linux kernel by calling `netif_rx`. In the softirq `net_rx_action`[1] for `netif_rx`, the packet is dequeued from the kernel queue, and either pushed into the `KernelTap` element (if Span is running in user space), or into the `FromDevice` element (if Span is running in the kernel). Note that we leverage the existing incoming queue functionality in the Linux kernel.

In Figure 3-2 (Right), we see the path a packet takes upon dispatch from Span, implemented either in the kernel, or in user-space. In the former case, the *ToLinux* Click element calls a function in the Linux kernel to dispatch the outgoing packet to the correct handler internal to the Linux kernel networking stack. In the later case, the *KernelTap* Click element delivers the packet to ethertap code in the Linux kernel. In both cases, the kernel calls `hard_start_xmit` in the `airo.o` module, which communicates with Span to decide if the packet should be buffered. Based on whether

[1]A "softirq" in the Linux 2.4 kernel series is similar (but not identical) to a "bottom-half" from the 2.2 kernel series. It is called by the scheduler asynchronously from the interrupt, and is used for processing that would take too long in the interrupt handler.

```
check-announce-coordinator()
  C = connect-pairs()
  if C > 0 then
    calculate delay using C as C_i
    wait delay
    if connect-pairs() > 0 then
      announce yourself as a coordinator
```

Figure 3-3: Coordinator announcement algorithm pseudo-code: a non-coordinator node periodically calls this routine to see if it should become a coordinator.

```
int connect-pairs()
  n = 0
    for each neighbor a in neighbor table do
      for each neighbor b such that b > a, in neighbor table do
        if share-other-coordinators(a, b) == false then
          n := n + 1
  return n
```

Figure 3-4: Coordinator announcement algorithm pseudo-code: returns the number of neighbor pairs a node can connect if it becomes a coordinator.

the packet's next hop is a coordinator or not, and if the current time is within the 802.11 ad hoc power-saving mode window for advertised traffic. When necessary, the `airo.o` module buffers packets, and makes calls to the Cisco Aironet hardware to transmit packets when possible. To transmit packets, the card allocates a fixed number of frame identifiers (FIDs), which are handles to a single transmission event. After transmission is complete, the hardware generates an interrupt so that the driver can reclaim the FID it used to transmit the packet.

3.2 Span Layer

In this section, we describe our implementation of the *Span* layer shown in Figure 3-1 above and described in Chapter 2.

3.2.1 Coordinator Election

Span's election algorithm requires each node to advertise its coordinators, its neighbors, and if it is a coordinator, a tentative coordinator, or a non-coordinator. From this information, the underlying routing protocol builds a *neighbor table* that contains lists of nodes up to two hops away from it and each node's status as coordinator or non-coordinator. A node uses information from its neighbor table to determine if it should announce or withdraw itself as a coordinator. Figures 3-3, 3-4, and 3-5 show the coordinator announcement algorithm. A non-coordinator node periodically calls `check-announce-coordinator` to determine if it should become a coordinator

```
boolean share-other-coordinators(a, b)
  for each coordinator c_a in a's coordinator list do
    if c_a equals self then
      continue
    else if c_a in b's coordinator list then
      return true
    else if c_a in neighbor table then
      for each coordinator c_c_a in c_a's coordinator list do
        if c_c_a equals self then
          continue
        else if c_c_a in b's coordinator list then
          return true
  return false
```

Figure 3-5: Coordinator announcement algorithm pseudo-code: returns true if and only if neighbors `a` and `b` are connected by one or two other coordinators. Note that coordinator lists are kept in the neighbor table.

```
struct span_encap_hdr {
  bool next_hop_coord;
};
```

Figure 3-6: Encapsulation header for IP packets passing through the *Span* element.

or not. `check-announce-coordinator` first computes C, the number of additional neighbor pairs that would be connected if the node becomes a coordinator, using `connect-pair`. If $C > 0$, the node computes *delay* using Equation 2.4 and waits for *delay* seconds before recomputing C. If C continues to be greater than 0 after *delay* seconds, the node announces itself as a coordinator. `connect-pair` calculates the number of would–be connected neighbor pairs by iterating through the node's neighbors in the neighbor table. A similar routine exists for checking if every pair of neighbor nodes can reach each other via one or two other neighbors. That routine is used by the withdraw algorithm.

In addition to the coordinator election algorithm shown in Figures 3-3, 3-4, and 3-5, we implemented a special case for electing coordinators. The routing algorithm can readily detect that a coordinator has left the region through MAC layer failure feedback. However, the Span election algorithm may not react fast enough to elect new coordinators. In the worst case, nodes must wait until the old coordinator information has expired in the neighbor table before a new coordinator can be elected. Since our routing algorithm falls back to using non-coordinators to route packets if coordinators do not exist, a non-coordinator node announces itself as a coordinator if it has received a large number of packets to route in the recent past. If this coordinator turns out to be redundant, the coordinator withdraw algorithm will force the node to withdraw itself as a coordinator soon after.

```
bool coordinator() const;
bool in_grace() const;
```

Figure 3-7: The Span interface. The *Span* module exports this interface for the purpose of routing protocols.

3.2.2 Span Encapsulation Header

Span adds a header to each outgoing packet sent by the Linux networking stack. The purpose of the Span encapsulation header is to inform the link layer whether the next hop node for the packet is or is not a coordinator. Whether the next hop is a coordinator is tantamount to whether the next hop is saving power or not, which changes the link layer's actions as described below in our discussion of the link layer.

3.2.3 Span Interface

The interface that Span exports to the other Click elements is simple. As shown in Figure 3-7, it consists of two functions that query the state of Span. The function `coordinator` returns `true` if and only if Span at this node has elected itself coordinator. The function `in_grace` returns `true` if and only if Span at this node is in its grace period. The routing protocol is free to make use of this information, and a Span-aware routing protocol will make extensive use of this information, as described in the next section.

3.3 Routing Architecture

This section describes the *Routing* layer shown above in Figure 3-1. The challenge in implementing this part of Span was formulating the interface that should exist between Span and the underlying routing protocol, with the goals of efficiency (in terms of routing and Span message overhead), performance (in terms of power conservation and routing data)

3.3.1 Routing Layer Interface

By examining at the interaction between Span and several routing protocols, we have formed a "Span-compliant" routing protocol abstraction, shown in Figure 3-8. All routing protocols wishing to work with Span should implement the interface shown in this figure. This decouples the implementation of Span from the implementation of the routing protocol, facilitating code reuse.

Note that in our architecture, the routing protocol is responsible for updating two-hop neighborhood information for the node. An alternate design is to move this functionality into the Span code, separate from the routing code. We chose to implement two-hop neighborhood maintenance functionality in the routing protocol because we believe that many useful recent routing protocols (for example, the Grid location service with geographic forwarding) implement this functionality anyway.

```
class SpanRoutingProtocol {
  public:

    // returns an iterator that scans over only our
    // one-hop neighbors
    virtual NeighborIterator nbrs() = 0;

    // the number of one-hop neighbors we have
    virtual int nbr_cnt() = 0;

    virtual NeighborEntry *find_nbr(IPAddress ip) = 0;

    // hint from Span to update other nodes' routing state
    virtual void force_routing_update() = 0;

    // returns true iff the next hop for this ip packet
    // is a coordinator.  should err to give false positives
    // for safety.
    virtual bool next_hop_coord(IPAddress ip) = 0;
};
```

Figure 3-8: C++ code specifying the interface for a Span-compliant routing protocol. A `NeighborEntry` contains IP and MAC addresses for a node's neighbor, and a `NeighborIterator` is an iterator that iterates over `NeighborEntry` elements. Code has been edited for clarity.

Consequently, the routing protocol can suppress or piggyback such `HELLO` messages on its own hello messages, potentially resulting in a halving of the number of messages sent and received.

3.3.2 Dynamically-Sequenced Distance Vector (DSDV)

We implemented Dynamically-Sequenced Distance Vector (DSDV) [23] to route packets through our wireless network testbed. We implemented all features of DSDV from scratch, referring to the relevant *ns-2* simulator code when necessary.

We chose to implement DSDV because of its simplicity, and because it maintains neighbor information an infinite number of hops away. While Span only requires information about neighbors up to two hops away, it is trivial to extend this functionality to a fully-operational DSDV implementation.

The DSDV Algorithm

The idea behind DSDV is for each node to distribute distance-vector information about all destinations to each of its neighbors. More precisely, a node i maintains a routing table consisting of distances to every destination j in the network $\{d_{ij}\}$ and corresponding next hops $\{N_{ij}\}$. It periodically broadcasts this routing table to its neighbors in the network. When a node i receives node k's routing table, for each j, it compares $1 + d_{kj}$ with d_{ij} and sets $N_{ij} \leftarrow k$ if the former value is strictly less than the latter. In this way, DSDV runs a distributed Bellman-Ford shortest-paths algorithm.

DSDV adds destination-sequenced sequence numbers to the basic algorithm for the purpose of avoiding stale routes, and the consequent routing loops that can form. Each destination j keeps track of a sequence number s_j, which is incremented every time j advertises its own route (i.e., $d_{jj} = 0$) to its neighbors. A node records sequence numbers of all the routes it has seen, and rejects routes with lower sequence numbers, and always (even if the distance metric is worse) replaces routes with strictly greater sequence numbers than it has already seen. A node x thus performs the Bellman-Ford distance comparison to determine if it will accept the route if and only if the sequence number of the route is equal to the highest sequence number x has yet seen.

As described so far, DSDV suffers from a problem with fluctuating routes. To illustrate this, in Figure 3-9 we see host one advertising a route to all other hosts on the network. Suppose that the path from node one to node four through Node Collection A is one hop longer than the same path through Node Collection B. Suppose further that due to the relative phase of the periodic routing updates in A and B, that the route through A always arrives at four before the corresponding route through B. In this situation, the route to one from or through four will fluctuate at the DSDV routing update period. To prevent this, DSDV adds a *settling table* which keeps track of the time that a route takes to *settle*—stabilize on the least-metric route for a given sequence number. For each destination, the settling table keeps track of the last and average settling times. Then on receipt of a new route with a higher sequence number

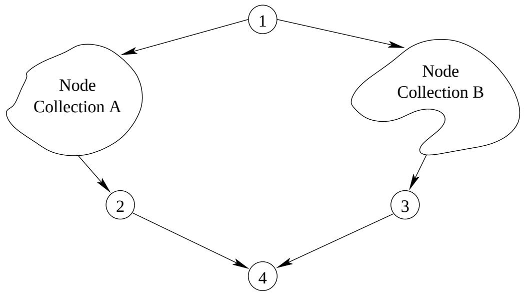


Figure 3-9: An example of a network where DSDV routes can fluctuate rapidly. In the figure the clouds represents sets of mobile hosts providing connectivity between edges into and out from the cloud.

than previously known, a node forwards on that route, but waits twice the average settling time before advertising it.

DSDV also maintains a table containing the time it last heard from each neighboring node. If this time grows past a configurable threshold, all routes through that link are deleted from the node's routing table, and route withdrawal messages are sent, containing an metric of ∞ for each route that was broken.

Scaling Properties of DSDV. In terms of storage, each host keeps an entry for each destination in the network. Thus the storage space required is on the order of the number of destinations, or nodes, in the ad hoc network. In terms of message size and number of messages, however, each host's full routing update is of size proportional to the number of destinations in the network.

Therefore, traffic in any small constant area of the network scales proportionally with the number of nodes in the network, while the total amount of traffic in the network scales with the square of network size. As a result, DSDV traffic will easily dominate the available bandwidth if the number of nodes in the network gets large enough.

Integrating DSDV with Span

In this section, we describe how we integrated DSDV routing with Span coordinator election. In order to conform to the *SpanRoutingProtocol* interface, our DSDV implementation uses the routing protocol's periodic update messages to exchange information about two-hop neighbors and their Span coordinator status. Information about two-hop neighbors, therefore, is freely propagated between non-coordinator nodes, coordinator nodes, and between the two types of nodes.

However, nodes drop route advertisements from non-coordinators to avoid routing through non-coordinators. The exception to this rule is that a non-coordinator can serve as the source or destination of a route. A problem with this scheme arises when a link breaks due to mobility or node failure. In this case, non-coordinators need to exchange routing information to discover that a route has disappeared (and potentially, a new Span coordinator needs to be elected). Consequently, nodes always listen to route withdrawal messages (route updates with an ∞ metric).

Testbed Configuration

We now present the Click configuration used in our experiments in Figure 3-10. We implemented DSDV and Span in separate Click elements; the *DSDV* and *Span* elements communicate through the `SpanRPElement` interface described above, and packets move between Click elements as shown by the arrows in the diagram. The Click elements in this diagram implement the *Span* and *Routing* layers of Figure 3-1 above. The `FromDevice`, `KernelTap`, and `ToDevice` elements interface with the Linux kernel as described above in Section 3.1.1.

The Click element *Span* performs two functions in our implementation. As shown in Figure 3-10, the Span element has four ports, two of which handle processing and generation of the Span `HELLO` packets described in the Chapter 2. The remaining two ports pass through IP packets, inserting a *span IP encapsulation header* (shown in Figure 3-6) into the packet.

3.4 Link Layer Design

This section details the design of the software-based link layer used in the testbed. In the future, we hope to design an open hardware, power-saving MAC. In fact, the main challenge in the design of the link layer is implementing a power-saving medium access control (MAC) layer.

None of the major wireless chip makers implement an ad hoc, power-saving MAC layer in hardware yet [33]. To overcome this problem, we researched two chip sets that offer a *host-based access point* operating mode. Most wireless network interface cards support two forms of access point mode. In host-based access point mode, the host performs most of the access point functionality. Typically, the card does not encapsulate 802.3 (Ethernet) frames in this mode. One significant advantage of host-based AP mode is that the driver writer is given more freedom in the 802.11 implementation. Conversely, In *firmware-based access-point mode* the access point functionality is contained in code that is downloaded to volatile card RAM (often referred to as tertiary firmware), or built in to the non-volatile chip firmware.

We found that the Cisco Aironet 340 series offers a well-documented host-based access point mode that we can use to implement the power-saving-specific parts of the MAC; we implemented the remainder of the MAC in software. The main challenge in implementing the MAC in software is to precisely control when the card sends frames. To a certain extent, this is impossible to do, and so the software has to sometimes wait for the card to finish sending or receiving a frame, introducing latency, throughput, power, and loss-rate problems into the link layer. The main objective of the design of the software MAC was to minimize these problems, as well as improve upon the 802.11 ad hoc power-saving mode MAC proposed by the IEEE.

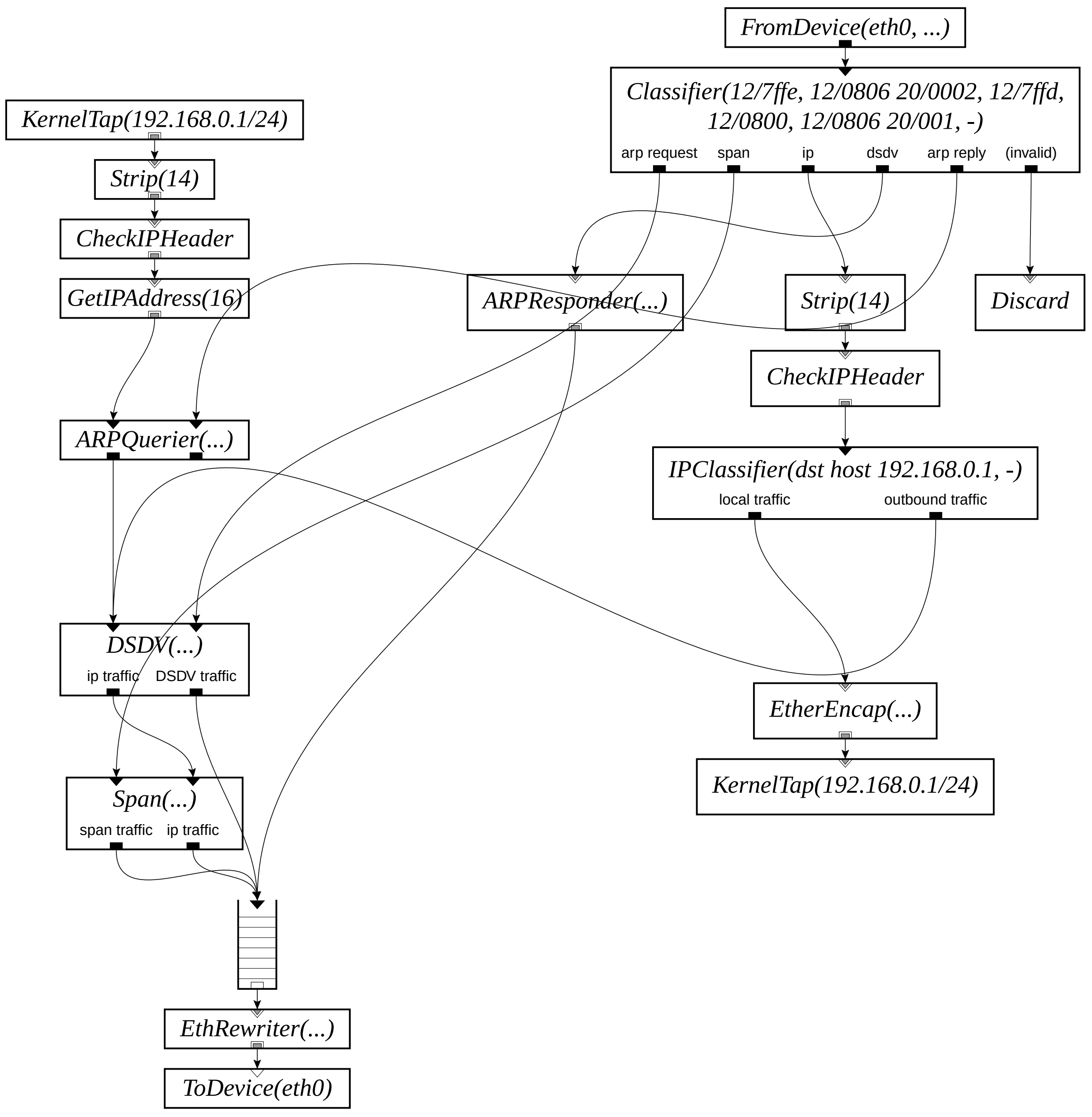
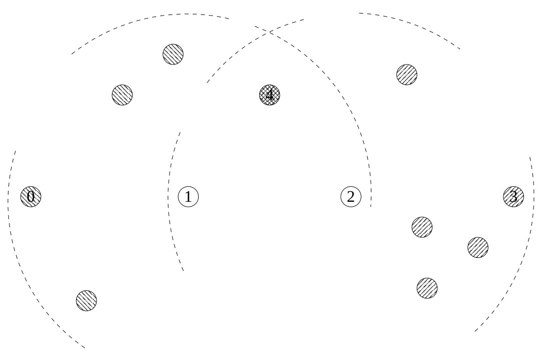


Figure 3-10: The user-level Click configuration for our testbed experiments. Arrows in the diagram indicate packet flow between the boxes that represent click elements. The *EthRewriter*, *Span*, and *DSDV* elements are original contributions of this thesis.



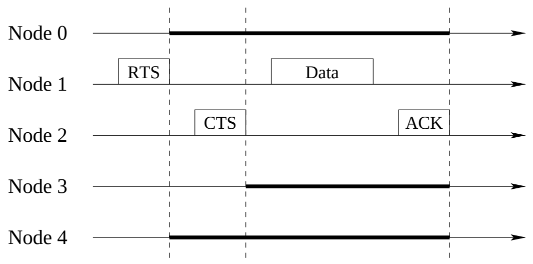


Figure 3-11: *Left:* A node arrangement that illustrates the utility of the RTS/CTS handshake. In this figure Node 1 is sending data to Node 2. Each node's radio range is indicated by the dotted circles. *Right:* Data transmitted over the wireless medium and network allocation vector status as Node 1 sends a packet to Node 2. Time runs along the x-axis. White boxes represent data, and bold black lines represent a non-zero network allocation vector.

Span determines when to turn a node's radio on or off, but depends on the low level MAC layer to support power saving functions, such as buffering packets for sleeping nodes. We have implemented Span on top of the 802.11 MAC and physical layers with ad hoc power saving support [1].

3.4.1 IEEE 802.11 Distributed Coordination Function

The IEEE refers to the basic 802.11 medium access protocol as the distributed coordination function (DCF). The DCF is important because all commodity 802.11 devices, from PCMCIA client cards to high-end access points, follow the DCF rules. The DCF is a protocol for collision avoidance (CA) in a multiple-access medium where carrier sense is available (CSMA); hence the term CSMA/CA. In our testbed we utilized the DCF implementation in the Cisco Aironet hardware.

In Figure 3-11 we show an arrangement of nodes in an ad hoc networking and the corresponding traffic over the wireless medium as Node 1 sends to Node 2. The DCF stipulates that before sending a packet, Node 1 broadcasts a *ready to send* (RTS) frame to its local neighborhood; the RTS frame contains the time duration of the transmission. All nodes keep a piece of state called the *network allocation vector* (NAV) that indicates reservation of the shared medium. Nodes that receive the RTS (Nodes 0 and 4) set their NAVs for the duration of the transmission sequence.

When Node 2 hears an RTS frame for data destined to it from Node 1, it broadcasts a *clear to send* (CTS) frame to its local neighborhood. The CTS is similar in structure to the RTS; both contain duration information. Nodes in Node 2's neighborhood (Nodes 4 and 3) set their NAVs in a similar way to those in Node 1's neighborhood do when they receive a RTS.

Next, Node 1 sends a *data* frame to Node 2, and Node 2 responds with an *acknowledgment* (ACK) frame.

Note that the only real gain in using RTS/CTS frames comes from a reduced duration of potential collisions. If stations are sending large data frames, then the penalty for collisions will be high, since entire frames are discarded at once when a collision occurs. In this case, the MAC protocol should use RTS/CTS frames. However, when stations are sending mostly small data frames (close to the size of the RTS/CTS frame), the penalty for a collision is small, and RTS/CTS frames are simply a waste of bandwidth. Hence most implementations use RTS/CTS for data frames of size larger than an adjustable threshold.

Also note that the neighborhoods of Nodes 1 and 2 may differ, and so both the RTS and the CTS are required to reserve the medium. A side-effect of this is that the medium is reserved "both ways," and links are forced to be symmetrical (i.e., if A can transmit to B, then B can transmit to A).

3.4.2 IEEE 802.11 Ad Hoc Power-Saving Mode

The 802.11 ad hoc power-saving mode (PSM) uses periodic *beacons* to synchronize nodes in the network. Beacon packets contain timestamps that synchronize nodes' clocks. A beacon period starts with an ad hoc traffic indication message window (*ATIM window*), during which all nodes are listening, and pending traffic transmissions are advertised. A node that receives and acknowledges an advertisement for unicast or broadcast traffic directed to itself must stay on for the rest of the beacon period. Otherwise, it can turn itself off at the end of the ATIM window, until the beginning of the next beacon period. After the ATIM window, advertised traffic is transmitted. Since traffic cannot be transmitted during the ATIM window, the available channel capacity is reduced. In our testbed, we implemented the 802.11 ad hoc PSM in software.

Figure 3-12 shows a typical 802.11 ad hoc PSM traffic scenario where stations 1, 2, and 3 can all hear each other. In the picture, station 1 sends an ATIM to station 2, which 2 acknowledges. Then, in the subsequent data window, the DCF RTS/CTS/DATA/ACK sequence may proceed. Note that for the first beacon period, overhearing stations (such as 3) may sleep outside of the ATIM window, but must wake up for the subsequent ATIM window. In the second ATIM window, station 1 sends a multicast ATIM in the ATIM window. Consequently it and all other stations must remain awake for the entire second beacon period. Note that multicast ATIMs are not acknowledged, and that the RTS/CTS/ACK handshaking is not used for multicast data, as stipulated by the 802.11 standard.

When the 802.11 MAC layer is asked to send a packet, it may or may not be able to send it immediately, depending on which ATIMs have been sent and acknowledged in the immediately preceding or current, ATIM window. If the packet arrives at the MAC during the ATIM window, or if the advertisement for the packet has not been acknowledged, it needs to be buffered. In our implementation, we buffer packets for two beacon periods. Packets that have not been transmitted after two beacon periods are dropped.

The beacon period and ATIM window size greatly affect routing performance [29]. While using a small ATIM window may improve energy savings, there may not be enough time for all buffered packets to be advertised. Using an ATIM window that

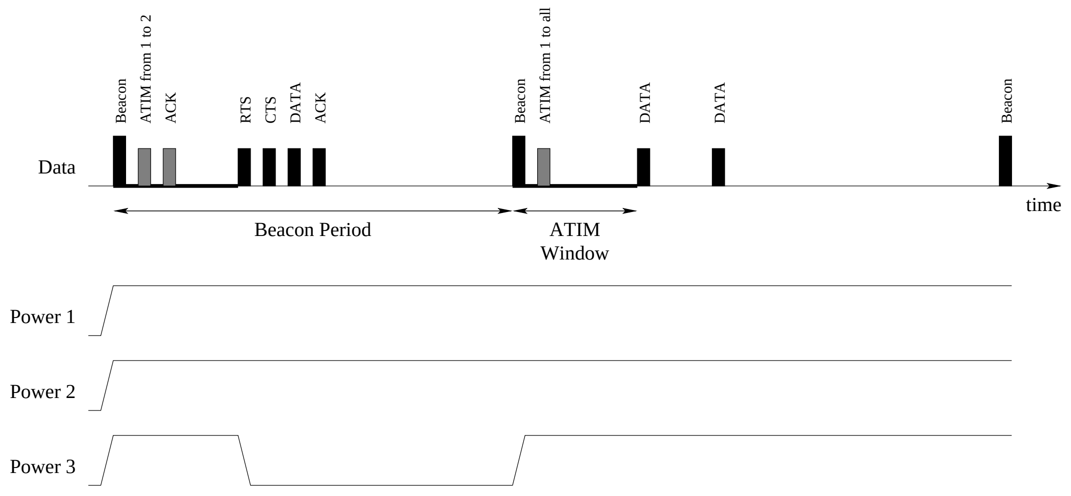


Figure 3-12: Two beacon periods' worth of traffic in the 802.11 ad hoc power-saving mode. We show traffic on the network (*top*), and (*bottom*) the on/off status of the radios at stations numbered one, two, and three. The bold areas of the timeline indicate the ATIM window.

is too large not only decreases available channel utilization, it may also not leave enough room between the end of the ATIM window and the beginning of the next beacon period to transmit all advertised traffic.

Aside from decreased channel capacity, 802.11 PSM (without Span) also suffers from a long packet delivery latency: for each hop that a packet traverses, the expected packet delay is half a beacon period.

3.4.3 Improving the 802.11 Ad Hoc Power-Saving MAC

Span can improve routing throughput and packet delivery latency of the 802.11 ad hoc PSM. Since coordinators do not operate in power saving mode, packets routed between coordinators do not need to be advertised or delayed. To further take advantage of the synergy between Span and 802.11 power saving mode, we have made the following modifications to our simulation of 802.11 power saving mode.

1. *No advertisements for packets between coordinators.* Packets routed between coordinators are marked by Span. While the MAC layer still needs to buffer these packets if they arrive during the ATIM window, it does not send traffic advertisements for them. To ensure that Span does not provide incorrect information due to topology changes, the MAC maintains a separate neighbor table. The MAC layer uses a bit in the MAC header of each packet it sends to notify neighbors of its power saving status. Since the MAC layer can sniff the header of every packet, including RTS packets, this neighbor table is likely to be correct. When a node withdraws as a coordinator, advertisements for traffic

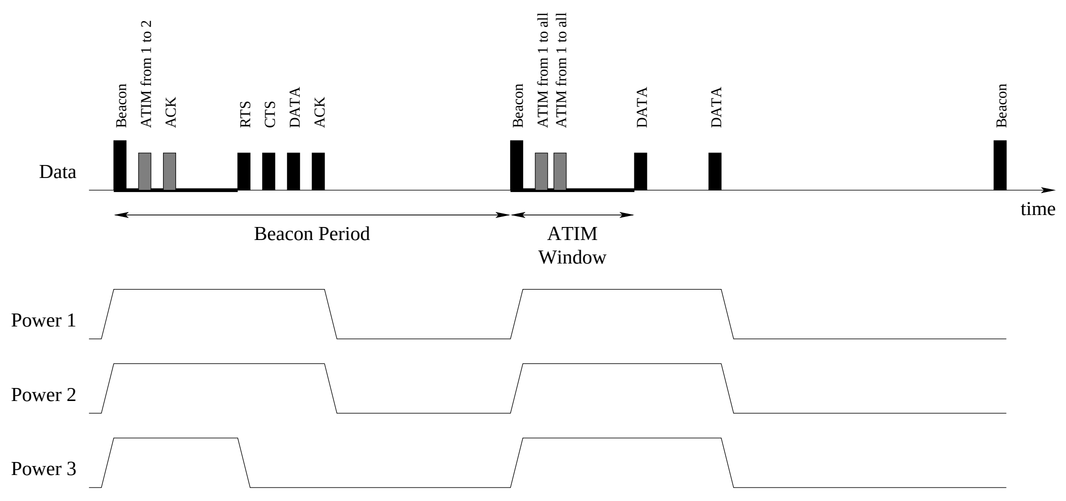


Figure 3-13: Two beacon periods' worth of traffic in the 802.11 ad hoc power-saving mode, with our improvements. We show traffic on the network (*top*), and (*bottom*) the on/off status of the radios at stations numbered one, two, and three. The bold areas of the timeline indicate the ATIM window.

to that node will be sent during the next ATIM window. This optimization allows the ATIM window to be reduced without hurting throughput.

2. *Individually advertise each broadcast message.* With unmodified 802.11 power saving mode, a node only needs to send one broadcast advertisement even if it has more than one broadcast message to send. This is because once a node hears an advertisement for a broadcast message, it stays up for the entire duration of the beacon period. Since most traffic to non-coordinator nodes in our network would be broadcast messages sent by Span and the routing protocol, we modified the MAC so each broadcast message must be explicitly advertised. For example, if a node receives 5 broadcast advertisements, no unicast advertisements, and then 5 broadcast messages after the ATIM window, it can safely turn itself off.

3. *New advertised traffic window.* In the unmodified 802.11 power-saving mode, if a node receives a unicast advertisement, it must remain on for the rest of the beacon period. In a Span network, packets routed via non-coordinator nodes are rare. To take advantage of this, we introduced a new *advertised traffic window* in the MAC. The advertised traffic window is smaller than the beacon period. It starts at the beginning of the beacon period, and extends beyond the end of the ATIM window. Outside the ATIM window but inside the advertised traffic window, advertised packets and packets to coordinators can be transmitted. Outside the advertised traffic window, however, only packets between coordinators can be transmitted. This allows a node in power saving mode to turn itself off at the end of the advertised traffic window until the next beacon period.

These three modifications allow each node to use a long beacon period and a short ATIM window. The short ATIM window improves channel utilization, while the long beacon period increases the fraction of time a non-coordinator node can remain asleep. Span does not require these modifications, but does better when they are implemented.

3.5 Chapter Summary

In this chapter we have described the design of each layer of the Span implementation. We have also described the interfaces between the different layers of the system, and pointed out where others can add functionality to the system through these interfaces. Finally, we have discussed the tools we used to implement each of the layers, and our reasons for choosing the tools we chose.

Chapter 4

Experimental Evaluation

In this chapter we present the results of experiments that measure how well Span performs in a real ad hoc network of handheld computers.

4.1 Coordinator Election Experiments

This section shows that our Span implementation makes good choices about which nodes to elect as coordinator.

4.1.1 Test Topologies

We now present some test topologies to verify the correctness of the entire Span implementation, including the 802.11 ad hoc power-saving software MAC, routing module, Span module, and their interactions with the Linux kernel and networking stack.

We began with the four node network topology shown in Figure 4-1, to initially test our implementation. Not long after, we added four more nodes to form the eight node topology shown in Figure 4-2. In both topologies, note that our implementation was

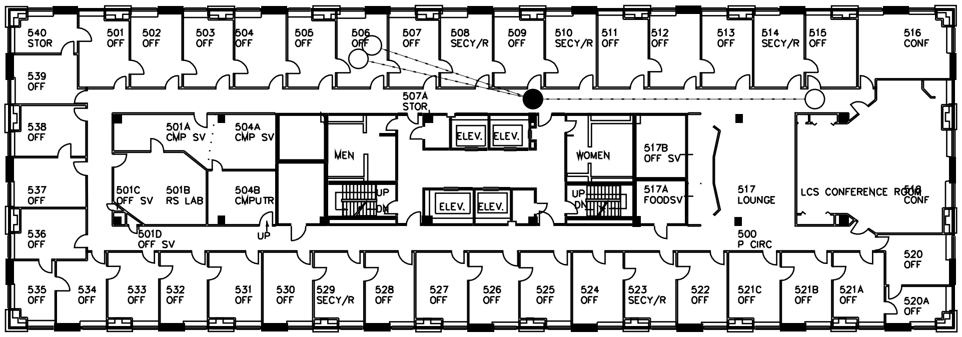


Figure 4-1: A four node test topology. In this and subsequent floorplan figures, black nodes indicate Span coordinators and white nodes indicate non-coordinators. We note that exactly the right coordinator is elected in this topology.

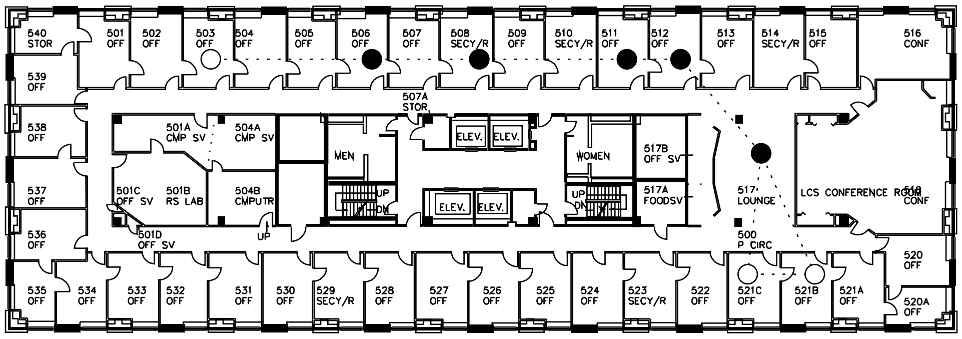


Figure 4-2: An eight node test topology.

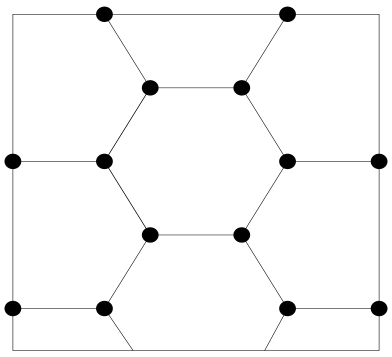


Figure 4-3: An approximation to an optimal layout of coordinators in a 1000 meter $\times$ 1000 meter area. There are 14 coordinators in this layout.

successful in electing the right Span coordinators to provide a connected forwarding backbone.

4.1.2 Simulation Results

Ideally, Span would choose just enough coordinators to preserve connectivity and capacity, but no more. Any number above this minimum would waste power in the network. This section compares the number of coordinators Span chooses with the number that would be required to form a hexagonal grid layout, shown in Figure 4-3; the hexagonal grid layout of nodes, while perhaps not optimal, produces a connected backbone in every direction with very few coordinators.

The hexagonal grid layout of coordinators place a coordinator at each vertex of a hexagon. Every coordinator can communicate with the three coordinators that it is connected to through an edge of a hexagon, which is 250 meters long (the radio range). Each hexagon has six coordinators, but each coordinator is shared by three hexagons. Therefore each hexagon is only responsible for two coordinators. Each hexagon has an area of 162,380 m^2. Thus, given a simulation area of d^2 meters, the number of

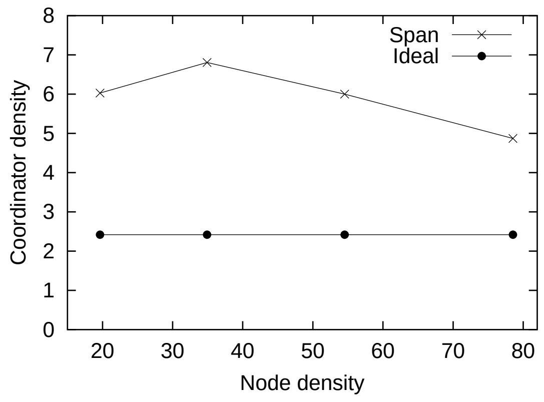


Figure 4-4: Ideal and actual coordinator density as a function of node density. The ideal curve represents an approximate lower bound on the number of coordinators needed. Span elects more coordinators than the ideal case because of lower node density, coordinator rotation, and announcement collision.

coordinators expected in this area, C_{ideal} is

$$C_{ideal} = 2 \cdot \frac{d^2}{162380} \tag{4.1}$$

Figure 4-4 shows coordinator density as a function of node density. For each node density, coordinator density is computed from the average number of coordinators elected by Span over 500 seconds of five mobile simulations. Points on the "Ideal" curve in Figure 4-4 are computed using the ideal number of coordinators predicted by Equation 4.1.

Span elects more coordinators than Equation 4.1 suggests. There are two reasons for this. First, Equation 4.1 describes a layout in a network that is dense enough such that there is a node at every corner of every hexagon. When the node density is moderate, on the other hand, more nodes are needed to provide connectivity between the hexagons. Second, to rotate coordinators among all nodes, the optimal set of coordinators may not always be selected.

4.2 Latency Experiments

This section describes the experiments we performed to measure how Span affects latency in our testbed.

4.2.1 One-hop Latency

To test the latency of our 802.11 power-saving mode MAC, we used `ping` to measure the one-hop latencies between a pair of Span coordinators and a pair of non-

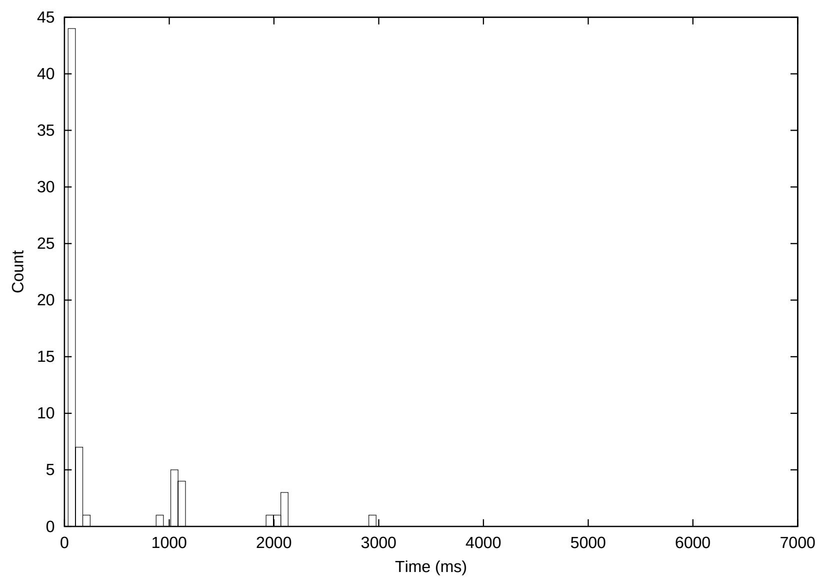


Figure 4-5: A histogram showing the distribution of ping times over one hop between Span coordinators. The bin size is 70 ms. The mode of the histogram is the first bin, containing ping times between zero and 70 ms.

coordinators. Figures 4-5 and 4-6 show the resulting distribution of ping times. In Figure 4-5 we see that most packets take about $L = 35$ ms to propagate between Span coordinators. This time is mostly due to processing in the MAC layer.

In Figure 4-6 we see that most packets take either 2170 ms to propagate between non-coordinators. Note that our 802.11 ad hoc power-saving mode beacon period is one second for these experiments. From Figure 4-7, we see that the latency between non-coordinators is composed of the following parts:

1. Wait for the time of the traffic advertisement window (at the beginning of each 1000 ms beacon period) [0-1000 ms, denoted (a) in Figure 4-7],

2. send the data packet advertisement [approximately 10 ms],

3. wait for an acknowledgment to the advertisement (this acknowledgment should come immediately) [approximately 10 ms],

4. wait for the data window (250 ms from the beginning of the beacon period, to the beginning of the next beacon period) [approximately 190 ms; this and the previous two steps are denoted (b) in Figure 4-7],

5. send the data [approximately 10 ms].

As shown in Figure 4-7, this sequence is repeated for the other side to respond to the ping, with the exception that the other side always waits for approximately 790 ms (denoted (c) in Figure 4-7). Summing the steps in this sequence gives us a latency L of 1210–2210 ms, depending on when the packet is sent. This calculation is consistent with the ping times in Figure 4-6.

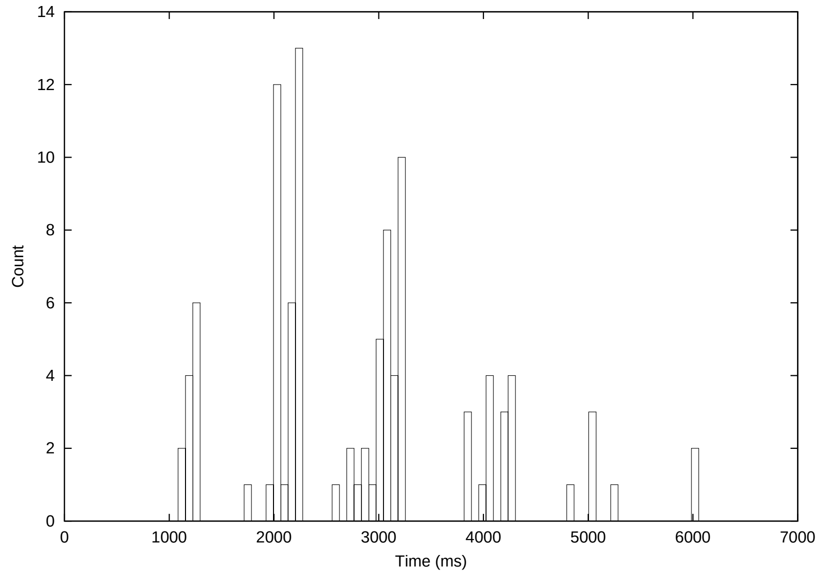


Figure 4-6: A histogram showing the distribution of ping times over one hop between nodes that are not Span coordinators. The bin size is 70 ms. The mode of the histogram is the 31st bin, which contains ping times between 2100 and 2170 ms.

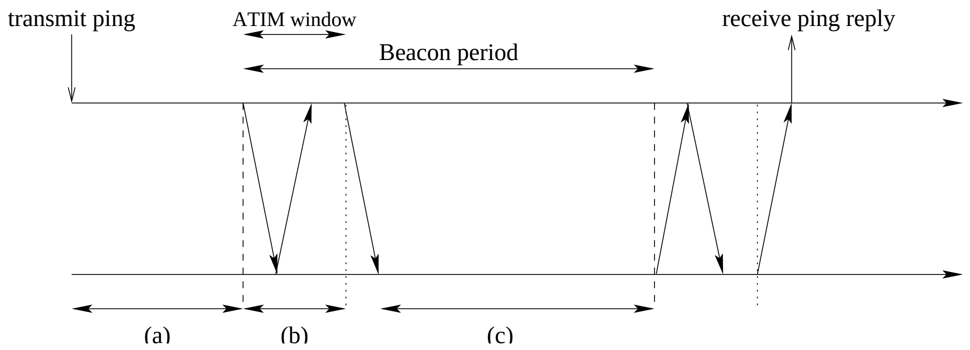


Figure 4-7: A time diagram showing the latency incurred by one ping packet between a pair of Span non-coordinators in 802.11 ad hoc power-saving mode. The dashed lines indicate beacon period boundaries, while the dotted lines indicate the end of a traffic advertisement window and the beginning of a data window within one beacon period.

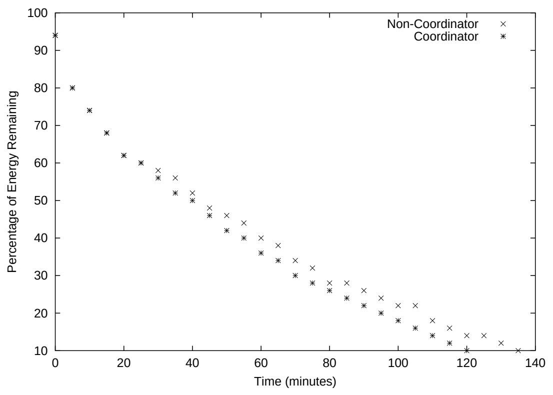


Figure 4-8: System energy remaining as a function of time, comparing Span an isolated Span coordinator versus an isolated Span non-coordinator. Each point represents a reading; readings were taken at five-minute intervals.

In both Figure 4-5 and Figure 4-6 the ping times cluster around $L + 1000k$, $k = 1, 2, 3, ...$ because sometimes the MAC cannot send and receive packets fast enough to meet the traffic advertisement window and beacon period deadlines. When this happens, the MAC buffers packets and tries at the next beacon period, resulting in a delay of one beacon period or 1000 ms. We believe that a hardware implementation of the MAC would restore the ping time between non-coordinators to L ms.

4.3 Energy Experiments

Using the automatic power management daemon (`apm`) available with our Linux distribution on the iPaq, we measured an iPaq's remaining battery energy at five-minute intervals, starting from a fully-charged battery, and terminating when battery charge reached a critical low threshold, chosen to be 10%. Both the iPaq and its PC-Card sleeve were disconnected from the line power throughout the experiment, and the screen was turned on and remained on throughout the duration of the experiment, to simulate a human using the iPaq. The readings were taken by a Perl script that queried the `/proc` interface to `apm` daemon.

Figure 4-8 shows the results of this experiment for a Span coordinator and a non-coordinator. In both cases the nodes were isolated from any other nodes. We note that the non-coordinator saves 15 minutes' worth of energy by turning off its radio. The battery lifetime of an iPaq running its radio at full power is 120 minutes. While this is a modest power savings, it is noticeable in a real system comprised of many parts that are major power consumers, such as the display, CPU, and memory.

In a separate experiment, we measured the lifetime of a Span coordinator with the display off. We now calculate the lifetime of a non-coordinator with the display off. Define T_{nc}, T_c as the lifetime of a Span (non-) coordinator with the display on, T'_{nc}, T'_c as the lifetime of a Span (non-) coordinator with the display off, and P_{nc}, P_c, P'_{nc}, and P'_c as the power consumed by the analogous configurations. Furthermore, define P_{on}, P_{off} as the power consumed by the radio of a Span coordinator or non-coordinator, respectively, P_{screen} as the power consumed by the iPaq screen and backlight, and P_{μ} as the power consumed by the rest of the iPaq, including the microprocessor. Then, by definition and from experimental results:

$$\begin{aligned} T_c &= \tfrac{E}{P_c} &= 120 \text{ minutes} \\ T_{nc} &= \tfrac{E}{P_{nc}} &= 135 \text{ minutes} \\ T'_c &= \tfrac{E}{P'_c} &= 240 \text{ minutes;} \end{aligned}$$

and also by definition, we have that

$$\begin{aligned} P_c &= P_{screen} + & P_{on} + P_{\mu} \\ P'_c &= & P_{on} + P_{\mu} \\ P_{nc} &= P_{screen} + & P_{off} + P_{\mu} \\ P'_{nc} &= & P_{off} + P_{\mu}. \end{aligned}$$

We can then compute the lifetime of an iPaq non-coordinator with its display backlight off as

$$\begin{aligned} T'_{nc} &= \frac{E}{P'_{nc}} \\ &= \frac{E}{P_{off} + P_{\mu}} \\ &= \frac{E}{P_{nc} - P_c + P'_c} \\ &= \frac{1}{\frac{1}{135} - \frac{1}{120} + \frac{1}{240}} \\ &= 309 \text{ minutes.} \end{aligned}$$

In Figure 4-9 we present the results of the above analysis. We note that while the difference between a Span coordinator and a non-coordinator with the backlight on is modest (only a 12% increase in node lifetime), our implementation of a Span non-coordinator increases node lifetime by 29% when the iPaq backlight is turned off.

4.4 Capacity Experiments

We used the `ttcp` tool [21] to measure the bandwidth available between a pair of Span coordinators, and a pair of non-coordinators. Figure 4-10 presents the results. In this experiment, the packet size was 1024 bytes and the number of packets was 500, for a total of approximately 51KB to transfer across the wireless link. The 802.11 ad hoc power-saving mode beacon period was one second. In Figure 4-10 (Left), we see the TCP algorithm adjusting to the slow round-trip time of the link between non-coordinators. Consequently, the TCP transmission timer expires before the sender receives an acknowledgment for the packet ending with byte 10,000 (for example),

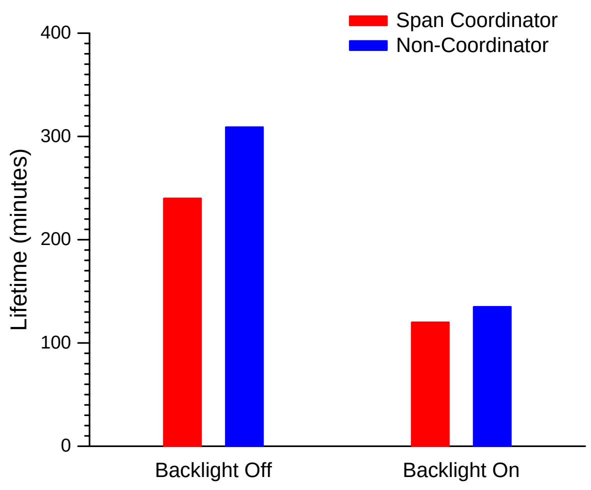


Figure 4-9: Total iPaq battery lifetime when a node is a Span coordinator or a non-coordinator. We separate the results based on whether the iPaq has its display backlight off or on. The result for a Span non-coordinator with its backlight off is calculated.

and the sender retransmits several packets in the first 15 seconds. Later in the trace, we see the sender has adjusted to the slow round trip time. The receiver receives packets at one-second intervals; sometimes missing receipt of packets in a one-second period due to lack of CPU time to send and receive a packet advertisement for that beacon period.

In Figure 4-10 (Right), we see note the subtle break in throughput at every beacon period boundary. This is due to the 802.11 ad hoc power-saving mode MAC disallowing data transmission during the advertisement window.

Between non-coordinators, this experiment achieved a throughput of 10.8 Kbits/s; between coordinators, this experiment achieved a throughput of 445.3 Kbits/s. The reason for the difference in throughput is not that the flow between non-coordinators did not have time to increase its congestion window adequately—both flows finished with a congestion window of 5440 bytes. There are several reasons for the difference in throughput. First, as we showed in Chapter 2, it takes some time to enable and disable the MAC. The second reason there is less capacity pertains directly to the 802.11 card hardware. During the `ttcp` run between non-coordinators, the link-layer driver software has less FIDs (see Section 3.1.1) available for use because each ATIM packet takes a FID. This requires us to periodically stop the Linux transmission queue (also see Section 3.1.1) more than in the coordinator-coordinator communication case, resulting in a lower throughput. Finally, because of the closed licensing of the 802.11 hardware we are using, we do not know the effect of repeatedly disabling and enabling the MAC in software as described in Section 3.4; quite possibly the MAC needs some time to reinitialize its state after being disabled; this would delay processing of transmit commands and decrease throughput.

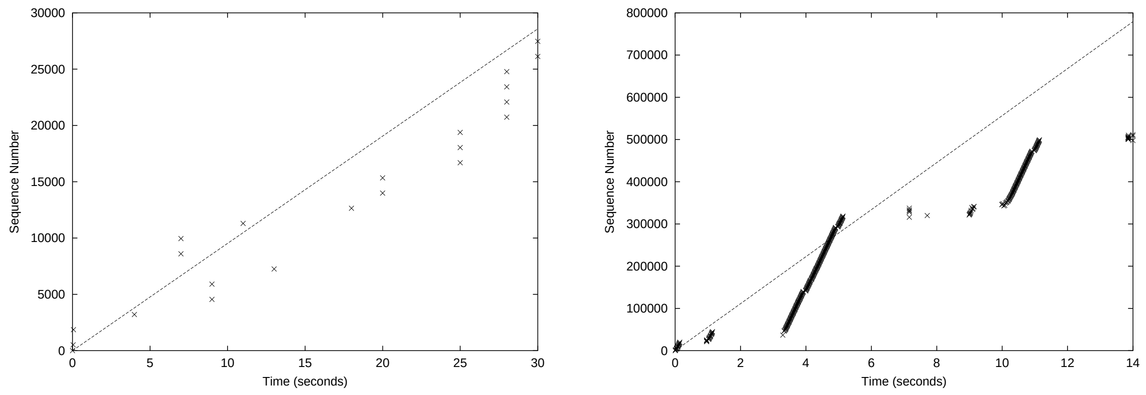


Figure 4-10: *Left:* a TCP trace between two Span non-coordinator nodes. *Right:* a TCP trace between two Span coordinators. In both figures the x-axis is the time that the packet was received, with $t = 0$ equal to the time the first byte was received. The y-axis is the sequence number of the last byte in the received packet. Each point on the graph represents the receipt of a 1024-byte packet. The dotted line shows the link bandwidth.

4.5 Capacity of 802.11 Networks

We now describe a graph-theoretic approach to the analysis of the capacity of an 802.11 wireless network, or any MACAW-type [2] network. First, we define *total one-hop capacity* C_1 ($\text{bit} \cdot \text{sec}^{-1}$) as the total rate of data that all nodes can send over one hop at once.

In Figure 4-11 we see node A transmitting to node B. When node A sends its RTS to B, all nodes in A's radio range will not initiate any transmissions for the duration of A's transmission. This prevents transmissions *from* nodes in A's radio range to any other node. Furthermore, nodes in A's radio range will not respond to RTS packets with a CTS for the duration of A's transmission[1]. This prevents transmissions *to* any nodes in A's radio range. The net result of the RTS packet is that any links terminating within A's radio range may not be used.

When B receives A's RTS packet, it responds with a CTS packet that has the same effect on nodes in its neighborhood. The net result is that any link terminating in either A or B's neighborhood may not be used.

The *maximum graph matching problem* is a related graph-theoretic problem that has been closely studied. The maximum graph matching problem is as follows. Given an undirected graph $G = (V, E)$, select a maximum subset of the edges $M \subseteq E$ such that no two elements of M are adjacent.

We define the *maximum two-hop graph matching problem* as follows. Given an undirected graph $G = (V, E)$, select a maximum subset of the edges $M \subseteq E$ such that no element e_1 of M is adjacent to the neighborhood of all $e_2 \neq e_1 \in M$, where the neighborhood of $e = (v_1, v_2) \in E$ is $\{v \in V : (v_1, v) \in E\} \cap \{v \in V : (v_2, v) \in E\}$.

[1]Note that this is not true of the MACA [15] protocol, but is true for MACAW and its derivatives, such as 802.11.

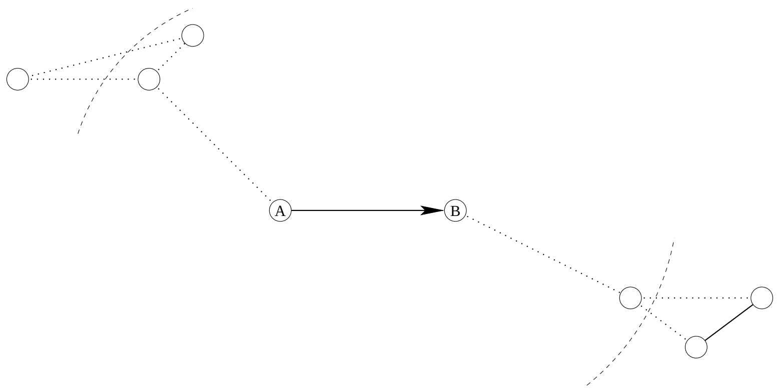


Figure 4-11: An example of the "two-hop matching" property. In this example, A is transmitting to B in the direction indicated by the arrow. Solid lines denote links available for use at the same time as this link while dotted lines denote links that may not be used concurrently with this link.

In the context of an 802.11 network, we observe that the cardinality $|M|$ of the maximum two-hop graph matching is equal to the maximum number of nodes in the network that can transmit at the same time. Thus we can bound the one-hop capacity as follows: $C_1 \leq |M| \cdot L$ where L is the link bandwidth ($\text{bit} \cdot \text{sec}^{-1}$). We plan to pursue this line of analysis in future work.

4.6 Chapter Summary

In this chapter we have presented experiments that validate the functionality and performance of our Span implementation. We have presented experiments characterizing our implementation's coordinator election, latency, node lifetime, and capacity. We have also presented four and eight-node wireless testbeds constructed at the MIT Laboratory for Computer Science on the fifth floor of the building.

Chapter 5

Conclusions

We have presented the design of Span, a power-saving protocol for ad hoc networks. We have shown some initial results to show that Span preserves the capacity of the ad hoc network as a whole. We have also described the implementation of an ad hoc networking testbed that implements Span. Our testbed is highly-scalable: a new node can be added to the testbed in a total of 4 minutes. Our testbed is also highly-extensible: our use of the Click modular router software enables us to change properties of the link layer or routing layer easily and without interference to other layers of the system. Finally, we have reported the results of experiments to show that Span does in fact perform as simulated in the testbed.

Key Conclusions. Our key conclusion is that Span is feasible to implement on a real platform such as Linux. Chapter 2 showed that a hardware implementation of a link-layer PSM can save large amounts of energy, while Chapter 4 showed that our implementation of Span can function in a real ad hoc network. Finally, Chapter 3 showed that our implementation can be easily extended. From these observations, we conclude that both Span and improvements to Span can be implemented on an iPaq in Linux.

5.1 Related Work

Much work on power-saving is being done in simulation; here the flexibility of a simulator can enable the analysis of more sophisticated algorithms. Another body of work comes from other efforts to develop ad hoc networking testbeds.

5.1.1 Power-Saving in Ad Hoc Networks

The set of coordinators elected by Span at any time is a connected dominating set of the graph formed by the nodes of the ad hoc network. A connected dominating set S of a graph G is a connected subgraph of G such that every vertex u in G is either in S or adjacent to some v in S. Routing using connected dominating sets of a graph can reduce the search space for routes [9, 36].

Das and Bharghavan [9] approximate the minimum connected dominating set of an ad hoc network, and route packets using nodes from that set. The set of coordinators elected by Span, however, has the additional property of being capacity preserving. Consequently, the connected dominating set elected by Span is likely to be larger than a minimal connected dominating set. For example, the black nodes in Figure 2-3 form a minimal connected dominating set. However, Span's election algorithm would additionally elect node 5 to be a coordinator to preserve capacity.

Wu and Li [36] propose a distributed algorithm for approximating connected dominating sets in an ad hoc network that also appears to preserve capacity. In a later paper, Wu and Gao [35] discuss power aware routing using the connected dominating sets. Their algorithm is similar to Span's coordinator election algorithm. Span, however, elects fewer coordinators because it actively prevents redundant coordinators by using randomized slotting and damping.

The recent GAF [38] scheme of Xu *et al.* has similar goals to Span. In GAF, nodes use geographic location information to divide the world into fixed square grids. The size of each grid stays constant, regardless of node density. Nodes within a grid switch between sleeping and listening, with the guarantee that one node in each grid stays up to route packets. Span differs from GAF in two important ways. First, unlike GAF, Span does not require that nodes know their geographic positions. Instead, Span uses local broadcast messages to discover and react to changes in the network topology. Second, Span integrates with 802.11 power saving mode nicely: non-coordinator nodes can still receive packets when operating in power saving mode.

In AFECA [37], each node maintains a count of the number of nodes within radio range, obtained by listening to transmissions on the channel. A node switches between sleeping and listening, with randomized sleep times proportional to the number of nearby nodes. The net effect is that the number of listening nodes is roughly constant, regardless of node density; as the density increases, more energy can be saved. AFECA's constants are chosen so that there is a high probability that the listening nodes form a connected graph, so that ad hoc forwarding works. An AFECA node does not know whether it is required to listen in order to maintain connectivity, so to be conservative AFECA tends to make nodes listen even when they could be asleep. Span differs from AFECA in that, with high likelihood, Span never keeps a node awake unless it is absolutely essential for connecting two of its neighbors. Furthermore, Span explicitly attempts to preserve the same overall system capacity as the underlying network where all nodes are awake, which ensures that no increase in congestion occurs.

The PAMAS power-saving medium access protocol [25, 31] turns off a node's radio when it is overhearing a packet not addressed to it. This approach is suitable for radios in which processing a received packet is expensive compared to listening to an idle radio channel. Kravets and Krishnan [16] present a system in which mobile units wake up periodically and poll a base station for newly arrived packets. Like Stemm and Katz [32], they show that setting the on/off periods based on application hints reduces both power and delay. Span assumes the presence of an ad hoc polling mechanism such as that provided by 802.11, and could potentially work in concert with application hints; such hints would apply only to sleeping nodes, not coordinators. Smith *et al.*

[19] propose an ad hoc network that elects a virtual base station to buffer packets for local nodes. They do not, however, attempt to make sure that enough of these base stations are present to preserve connectivity in a multi-hop ad hoc network.

As described in Chapter 2, minimum-energy routing saves power by choosing paths through a multi-hop ad hoc network that minimize the total transmit energy. Minimum-energy routing has been extended by Chang and Tassiulas [5] to maximize overall network lifetime by distributing energy consumption fairly. In this protocol, nodes adjust their transmission power levels and select routes to optimize performance. Ramanathan and Rosales-Hain describe distributed algorithms that vary transmission power and attempt to maintain connectedness [26]. Rodoplu and Meng give a distributed algorithm to produce minimum-power routes by varying node transmission power [28]. Wattenhofer *et al.* [34] describe a topology maintenance algorithm using similar underlying radio support, but their algorithm guarantees global connectedness using directional information. Span controls whether or not the receiver is powered on, rather than controlling the transmit power level. It also pays close attention to overall system capacity, in addition to maintaining connectivity.

In general, the basic idea that a path with many short hops is sometimes more energy-efficient than one with a few long hops could be applied to any ad hoc network with variable-power radios and knowledge of positions. This technique and Span's are orthogonal, so their benefits could potentially be combined.

5.1.2 Other Wireless Network Testbeds

The Grid [17] ad hoc networking testbed is comprised of approximately 20 desktop computers placed on the floor of an office building. Like our testbed, each node runs Click modular router software to perform routing tasks.

Unlike our testbed, the Grid testbed uses a standard 802.11 ad hoc link layer. The Grid testbed does not directly address power-conservation issues; each Grid node is a line-powered computer.

The Monarch Project

Maltz et. al. [18] constructed a five-node wireless network testbed. Each of their nodes was a car equipped with a laptop computer, a 900 MHz WaveLan I wireless network interface card, and a GPS receiver. Their routing protocol was DSR [3], which they implemented in the BSD networking stack.

As in our work, Maltz et. al. used simulation techniques in order to design and debug their upper-level routing software. To debug their implementation, they assigned virtual locations to nodes, and wrote a tool called `macfilter` that drops packets from machines not "within range" (according to the virtual locations) of the destination machine.

5.2 Future Work

The link-layer of our networking testbed is suboptimal; with a link-layer fully implemented in hardware, the following benefits can be realized.

1. Substantially more energy savings, since a hardware implementation of the MAC would free host CPU cycles and perform MAC tasks more efficiently.

2. Lower latency and more bandwidth, since much host intervention is required to send packets. Bandwidth is currently limited because host CPU processing time is the limiting factor of the link layer.

3. Better host CPU performance, allowing us to run a GUI on the host computer.

We plan to build an open hardware implementation of the 802.11 ad hoc power-saving mode, or an improved power-saving mode for ad hoc networks. We plan to make our implementation open because we believe that it will provide an excellent path for other researchers to implement their own MACAW-like power-saving MACs. Many power-saving MACs currently exist, yet are not implemented because of the difficulty involved in doing so.

Bibliography

[1] Wireless LAN Medium Access Control and Physical Layer Specifications, Aug. 1999. IEEE 802.11 Standard (IEEE Computer Society LAN MAN Standards Committee).

[2] Bharghavan, V., Demers, A. J., Shenker, S., and Zhang, L. MACAW: A Media Access Protocol for Wireless LANs. In *SIGCOMM* (1994), pp. 212–225.

[3] Broch, J., Johnson, D., and Maltz, D. The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks. In *Internet draft, IETF Mobile Ad Hoc Networking Working Group* (Dec. 1998).

[4] Buttyan, L., and Hubaux, J.-P. Stimulating Cooperation in Self-Organizing Mobile Ad Hoc Networks. Tech. Rep. DSC/2001/046, Institute for Computer Communications and Applications, Swiss Federal Institute of Technology, July 2001.

[5] Chang, J., and Tassiulas, L. Energy Conserving Routing in Wireless Ad Hoc Networks. In *Proceedings of IEEE INFOCOM* (Tel Aviv, Israel, 2000), pp. 22–31.

[6] Chen, B., Jamieson, K., Balakrishnan, H., and Morris, R. Span: An Energy-Efficient Coordination Algorithm for Topology Maintenance in Ad Hoc Wireless Networks. In *ACM/IEEE Conference on Mobile Computing and Networking (MobiCom)* (Rome, Italy, 2001), pp. 85–96.

[7] Chesson, G. XTP/protocol engine design. In *Proceedings of the IFIP WG6.1/6.4 Workshop* (Rüschlikon, May 1989).

[8] Corporation, C. Web site. `http://www.cisco.com`.

[9] Das, B., and Bharghavan, V. Routing in Ad-Hoc Networks Using Minimum Connected Dominating Sets. In *Proceedings of the IEEE International Conference on Communications (ICC)* (June 1997).

[10] Feeney, L., and Nilsson, M. Investigating the Energy Consumption of a Wireless Network Interface in an Ad Hoc Networking Environment. In *Proceedings of IEEE INFOCOM* (Anchorage, AK, 2001).

[11] Fenner, W. *Internet Group Management Protocol, Version 2*, Nov 1997. RFC-2236.

[12] Floyd, S., Jacobson, V., McCanne, S., Liu, C. G., and Zhang, L. A Reliable Multicast Framework for Lightweight Sessions and Application Level Framing. In *Proceedings of the ACM SIGCOMM* (Boston, MA, Sept. 1995), pp. 342–356.

[13] Haartsen, J., Naghshineh, M., Inouye, J., Joeressen, O., and Allen, W. Bluetooth: Vision, Goals, and Architecture. *Mobile Computing and Communications Review 2*, 4 (Oct 1998), 38–45.

[14] Heinzelman, W. R., Chandrakasan, A., and Balakrishnan, H. Energy-efficient Communication Protocols for Wireless Microsensor Networks. In *Proceedings of the Hawaaian International Conference on Systems Science* (Jan. 2000).

[15] Karn, P. MACA—A New Channel Access Method for Packet Radio. In *ARRL/CRRL Amateur Radio 9th Computer Networking Conference* (1990), pp. 134–140.

[16] Kravets, R., and Krishnan, P. Application-driven Power Management for Mobile Communication. In *Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)* (Dallas, TX, Oct. 1998), pp. 263–277.

[17] Li, J., Jannotti, J., Couto, D. D., Karger, D., and Morris, R. A Scalable Location Service for Geographic Ad-Hoc Routing. In *Proceedings of the Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)* (Aug. 2000), pp. 120–130.

[18] Maltz, D., Broch, J., and Johnson, D. Experiences designing and building a multi-hop wireless ad hoc network testbed. Tech. Rep. CMU-CS-99-116, Carnegie Mellon University, Pittsburgh, Pennsylvania, March 1999.

[19] Mangione-Smith, W., Ghang, P. S., Nazareth, S., Lettieri, P., Boring, W., and Jain, R. A Low Power Architecture for Wireless Multimedia Systems: Lessons Learned From Building a Power Hog. In *1996 International Symposium on Low Power Electronics and Design Digest of Technical Papers* (Monterey, CA, Aug. 1996), pp. 23–28.

[20] Morris, R., Jannotti, J., Kaashoek, F., Li, J., and De Couto, D. CarNet: A scalable ad hoc wireless network system. In *Proceedings of the 9th ACM SIGOPS European workshop: Beyond the PC: New Challenges for the Operating System* (Kolding, Denmark, Sep 2000).

[21] Muuss, M. The Story of the TTCP Program. `http://ftp.arl.mil/~mike/ttcp.html`.

[22] ns Notes and Documentation. http://www.isi.edu/vint/nsnam/, 2000.

[23] Perkins, C. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. In *Proceedings of the ACM SIGCOMM* (London, U.K., 1994), pp. 234–244.

[24] Priyantha, N., Chakraborty, A., and Balakrishnan, H. The Cricket Location-Support system. In *Proceedings of the 6th Annual ACM International Conference on Mobile Computing and Networking (MobiCom '00)* (August 2000), pp. 1–13.

[25] Raghavendra, C., and Singh, S. PAMAS: Power Aware Multi-Access Protocol with Signaling for Ad Hoc Networks. *ACM Computer Communication Review* (July 1998), 5–26.

[26] Ramanathan, R., and Rosales-Hain, R. Topology Control of Multi-hop Wireless Networks Using Transmit Power Adjustment. In *Proceedings of IEEE INFOCOM* (Tel Aviv, Israel, March 2000), pp. 404–413.

[27] Rappaport, T. S. *Wireless Communications: Principles and Practice*. Prentice Hall, New Jersey, 1996.

[28] Rodoplu, V., and Meng, T. H. Minimum Energy Mobile Wireless Networks. In *Proceedings of the IEEE International Conference on Communications (ICC)* (Atlanta, GA, June 1998), vol. 3, pp. 1633–1639.

[29] Rohl, C., Woesner, H., and Wolisz, A. A Short Look on Power Saving Mechanisms in the Wireless LAN Standard Draft IEEE 802.11. In *Proceedings of the the 6th WINLAB Workshop on Third Generation Wireless Systems* (New Brunswick, NJ, Mar. 1997).

[30] Shepard, T. A Channel Access Scheme for Large Dense Packet Radio Networks. In *Proceedings of the ACM SIGCOMM* (Stanford University, CA, Aug. 1996), pp. 219–230.

[31] Singh, S., Woo, M., and Raghavendra, C. S. Power-Aware Routing in Mobile Ad Hoc Networks. In *Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)* (Dallas, TX, 1998), pp. 181–190.

[32] Stemm, M., and Katz, R. Reducing Power Consumption of Network Interfaces in Handheld Devices. In *Proceedings of the Third Workshop on Mobile Multimedia Communications (MoMuC-3)* (Princeton, NJ, 1996).

[33] Tourrilhes, J. Wireless LAN Resources for Linux. http://www.hpl.hp.com/-personal/Jean_Tourrilhes/Linux/.

[34] Wattenhofer, R., Li, L., Bahl, P., and Wang, Y.-M. Distributed Topology Control for Power Efficient Operation in Multihop Wireless Ad Hoc Networks. In *Proceedings of IEEE INFOCOM* (Anchorage, AK, 2001).

[35] Wu, J., and Gao, M. On Calculating Power-Aware Connected Dominating Sets for Efficient Routing in Ad Hoc Wireless Networks. In *Proceedings of the 30th Annual International Conference On Parallel Processing* (Valencia, Spain, Sept. 2001).

[36] Wu, J., and Li, H. On Calculating Connected Dominating Set for Efficient Routing in Ad Hoc Wireless Networks. In *Proceedings of the Third International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications* (Seattle, WA, Aug. 1999).

[37] Xu, Y., Heidemann, J., and Estrin, D. Adaptive Energy-Conserving Routing for Multihop Ad Hoc Networks. Tech. Rep. 527, USC/ISI, Oct. 2000.

[38] Xu, Y., Heidemann, J., and Estrin, D. Geography-Informed Energy Conservation for Ad Hoc Routing. In *Proceedings of the Seventh Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)* (Rome, Italy, July 2001), pp. 70–83.