ENERGY EFFICIENCY AND SECURITY FOR MULTIHOP WIRELESS NETWORKS

BY

MATTHEW JEFFERSON MILLER

B.S., Clemson University, 2001
M.S., University of Illinois at Urbana-Champaign, 2003

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2006

Urbana, Illinois

ENERGY EFFICIENCY AND SECURITY FOR MULTIHOP WIRELESS NETWORKS

Matthew Jefferson Miller, Ph.D.
Department of Computer Science
University of Illinois at Urbana-Champaign, 2006
Nitin H. Vaidya, Adviser

As wireless devices are more widely used, it is clear that security and energy consumption are major concerns. From a energy perspective, it is increasingly evident that marginal gains in battery energy density necessitate energy efficient protocols. In the security realm, growth in the value and amount of information being transmitted over wireless channels demands confidentiality and integrity.

In the energy efficiency domain, this dissertation focuses on the wireless interface since this has been identified as a major source of energy consumption on devices such as sensors. Within this domain, many previous approaches propose using fixed listening and sleeping intervals regardless of the network conditions. We propose adaptive listening and sleeping techniques where these intervals are adjusted based on observations of traffic patterns and channel state. Another shortcoming of many power save protocols is that they wastefully listen for entire packets as a wake-up signal. In this dissertation, we propose carrier sensing techniques that reduce the cost of checking for such signals.

In the security domain, this dissertation looks at key distribution in wireless sensor networks. Because such devices may face severe resource constraints, symmetric keys are used since public-key cryptography may be infeasible. Previous approaches to this problem include key predistribution, and broadcasting plaintext keys, under the assumption that few eavesdroppers are present during key discovery. However, drawbacks to these approaches include poor secure connectivity or degraded security when several eavesdroppers are in the network. Our work exploits the underlying wireless channel diversity to address the problem. In doing so, our key distribution protocol effectively addresses the drawbacks of previous techniques.

The major contributions of this dissertation are: (1) leveraging multiple channels to improve the connectivity and security of key distribution, (2) proposing adaptive power save mechanisms to reduce energy consumption, and (3) improving power save protocols by using carrier sensing to enhance their energy efficiency.

Abstract

As wireless devices are more widely used, it is clear that security and energy consumption are major concerns. From a energy perspective, it is increasingly evident that marginal gains in battery energy density necessitate energy efficient protocols. In the security realm, growth in the value and amount of information being transmitted over wireless channels demands confidentiality and integrity.

In the energy efficiency domain, this dissertation focuses on the wireless interface since this has been identified as a major source of energy consumption on devices such as sensors. Within this domain, many previous approaches propose using fixed listening and sleeping intervals regardless of the network conditions. We propose adaptive listening and sleeping techniques where these intervals are adjusted based on observations of traffic patterns and channel state. Another shortcoming of many power save protocols is that they wastefully listen for entire packets as a wake-up signal. In this dissertation, we propose carrier sensing techniques that reduce the cost of checking for such signals.

In the security domain, this dissertation looks at key distribution in wireless sensor networks. Because such devices may face severe resource constraints, symmetric keys are used since public-key cryptography may be infeasible. Previous approaches to this problem include key predistribution, and broadcasting plaintext keys, under the assumption that few eavesdroppers are present during key discovery. However, drawbacks to these approaches include poor secure connectivity or degraded security when several eavesdroppers are in the network. Our work exploits the underlying wireless channel diversity to address the problem. In doing so, our key distribution protocol effectively addresses the drawbacks of previous techniques.

The major contributions of this dissertation are: (1) leveraging multiple channels to improve the connectivity and security of key distribution, (2) proposing adaptive power save mechanisms to reduce energy consumption, and (3) improving power save protocols by using carrier sensing to enhance their energy efficiency.

To Leigh Ann, Dad, and Mom

Acknowledgments

There are three people without whom I am certain I never would have reached this achievement. They are my wife, Leigh Ann, my father, David, and my mother, Charlene. Getting a Ph.D. is quite a roller coaster ride and I cannot imagine a better partner to have along for the journey than Leigh Ann. Having her to celebrate in my accomplishments, console me in my disappointments, and always have complete confidence in me is one of God's greatest blessings in my life. Ever since I was young, my dad has taken a sincere interest in everything I do and always encouraged me to be curious. My mom has kept me in her daily prayers since before I was born and always been there to comfort me.

In addition, many others have indirectly helped me make it to this point. For brevity, I will mention only a few by name. My brother, Eric, and I have always enjoyed a special bond and many good times. My grandparents, Charles and Ruth Hughes, have served as wonderful role models to me. I am forever in debt to Bill and Carol Menees for their hospitality to allow me to finish high school in South Carolina. Finally, I thank my in-laws, John, Judy, and Jared Buchanan, for letting me take their little girl all the way to Illinois in the pursuit of our dreams.

I thank all of my school teachers for giving me a solid educational foundation. I hope that an accomplishment such as this by a former student makes your efforts seem worthwhile. I would also like to acknowledge Dan Stanzione, Roy Pargas, and Walter Ligon for giving me an opportunity to participate in research as an undergraduate at Clemson University.

I have benefited greatly from working with a wonderful research group and would like to thank my adviser, Nitin Vaidya. Aside from mentoring me on my research, I appreciate his calm demeanor and ability to quickly assess research problems and solutions. I also thank my dissertation committee for their helpful discussions and Indranil Gupta, in particular, for his effort to help convert our class project into a conference publication. I would also like to acknowledge Cigdem Sengul who co-authored the work in Chapter 5 [1].

Finally, some financial support for this research was provided by National Defense Science and Engineering Graduate (NDSEG) and National Science Foundation (NSF) Fellowships.

Table of Contents

Chapter 1

Introduction

The use of wireless networks in our society is increasing at a rapid pace. The number of 802.11 [7] hotspots is expected to have a 31.5% compound annual growth rate during a four year period [8]. RFID production is projected to increase by a factor of 25 in the next four years [9]. During a five year period, the number of Bluetooth devices expects a 60% compound annual growth rate [10]. TinyOS [11], a popular open-source sensor operating system, typically has between 50 and 200 downloads per day [12].

Most existing wireless networks are *single hop* which means that devices communicating wirelessly are within range of each other. By contrast, *multihop* wireless networks may transmit packets over multiple wireless hops before reaching their final destination. While single-hop wireless networks are widespread, multihop networking offers several additional advantages:

- *Cost of Deployment:* Installing network wiring can be expensive. Their is a labor cost that often includes burying or running wires through walls, floors, or the ground. In addition, there may be costs associated with considerations such as ownership of the property across which the wires need laid, preservation of historical structures, and conservation of environmentally protected areas. By contrast, wireless infrastructure can be cheaper and less invasive to deploy. Off-the-shelf wireless routers can be used to extend the infrastructure hundreds of meters. Mesh networking [13] is an area of research that aims to do just that.

- *Rapid Deployment:* When the wire infrastructure is destroyed or unavailable, wireless networks can be used to quickly extend the reach of the wired infrastructure to areas of need. An example is military applications where a temporary network must be set up quickly. Another illustration is improving network coverage in a disaster area. This was attempted by some wireless enthusiasts in the wake of hurricane Katrina [14]. Vehicular [15,16] and underwater networks [17] are two more applications where wired communication may be impossible and, hence, are conducive to multihop wireless networking.

Sensors [18] are another example that may benefit from quickly deployed networks that can monitor the environment and/or detect events.

- *Capacity Improvement:* In a single-hop network, the capacity available to each node will decrease linearly with the size of the network. That is, given a channel bitrate of W and N nodes in the network, the maximum available capacity per node will be $\frac{W}{N}$. In a seminal work on multihop wireless networks, Gupta and Kumar [19] showed that the per node capacity in a multihop environment decreases with the square root of the number of nodes (i.e., $O\left(\frac{W}{\sqrt{N}}\right)$). Thus, we can observe a $O(\sqrt{N})$ improvement in the available capacity per node by using multihop wireless networks (as compared with single hop networks). Intuitively, this occurs because multiple communications can occur simultaneously among non-interfering pairs of nodes.

However, despite their promise, there remain research challenges associated with multihop wireless networks. Some issues include improving reliability, increasing throughput, and providing incentives to efficiently share resources. In this dissertation we focus on two areas of importance: energy efficiency and security.

As wireless devices are more widely used, clearly security and energy consumption are major concerns. From an energy perspective, it is increasingly evident that marginal gains in battery energy density necessitate energy efficient protocols. In the security realm, growth in the value and amount of information being transmitted over wireless channels demands confidentiality and integrity. Both are problems that need to be addressed if ubiquitous wireless networks are to become a reality.

Energy Efficiency: The necessity of energy efficient protocols for wireless devices is motivated by the fact that battery capacity has improved at a much smaller rate than that of other wireless device components. This trend is quantified in Table 1.1, that shows the relative improvement over a decade of various laptop components. While all of the other major components of the laptop showed one to three orders of magnitude improvement, the battery energy density increased by only a disappointing factor of less than three. The problem of available energy is further exacerbated by trends toward smaller devices (e.g., cell phones, sensors, lightweight laptops) which will have less available physical space for batteries. Thus, it is safe to assume that energy-constrained devices are a reality for the foreseeable future and that wireless protocol designers must cope with this rather than hoping for Moore's Law improvements in available energy.

In this work, we aim to reduce the energy consumption of the wireless networking interface by modifying network protocols. Table 1.2 shows an experimental energy breakdown, by component, for data traffic on a laptop and voice traffic on a cell phone. From this, we see that reducing the wireless interface energy consumption is only one aspect of a comprehensive solution towards energy efficient wireless devices that

Table 1.1: Improvement of various laptop components between 1990 and 2001 [20–22].

Laptop Component	**Relative Improvement from 1990 to 2001**
Disk Capacity	1200 ×
CPU Speed	393 ×
Available RAM	128 ×
Wireless Transfer Speed	18 ×
Battery Energy Density (J/kg)	2.7 ×

also requires research in the areas of architecture, operating systems, and application design [23]. Though our design techniques are applicable to multihop wireless networks in general, we note that our work is particularly beneficial for devices with no display (e.g., sensors) or small displays (e.g., cell phones, iPods, PDAs). Table 1.2 shows that the wireless interface energy consumption of such devices can account for over 60% of the device's overall energy usage.

Table 1.2: Fraction of energy used by device components for different hardware and traffic [24].

	Data Traffic on a Laptop	**Voice Traffic on a Cell Phone**
Display	45%	2%
Transmit	5%	24%
Receive/Idle	10%	37%
CPU	40%	37%

Wireless interfaces often have four power levels corresponding to the following states: transmitting, receiving, listening, and sleeping. Typically, the power required to listen is about the same as the power to receive. The power to transmit is generally slightly higher than the receive/listen power. However, the sleep power is usually one to four orders of magnitude less than the receive/listen power. For Mica2 Mote sensors [25], these power levels are shown in Table 1.3. Thus, to save energy, the interface should sleep as much as possible when it is not engaged in communication.

Table 1.3: Characteristics of a Mica2 Mote radio [25].

Radio State	**Power Consumption (mW)**
Transmit	81
Receive/Idle	30
Sleep	0.003

Motivated by the large reduction in energy consumption that is possible from entering the sleep state, we focus on *power save protocols*. Our work looks at three techniques to improve energy efficiency in power save protocols:

- *Carrier Sensing for Energy-Efficient Signaling:* Many power save protocols wastefully check for wake-up signals by listening to the channel on the order of the time it takes to receive a packet. In Chapter 3, we explore the use of carrier sensing to reduce the energy consumption of such protocols.

- *Adaptive Energy-Saving Protocols:* A common design used in both 802.11 [7] and sensor protocols [3,26] is to use fixed listening and sleeping intervals regardless of the network environment. Building on our previous work [27–29], in Chapter 4 we propose methods to dynamically adjust these intervals in response to indicators such as the sending rate and desired latency.

- *Energy-Efficient Broadcast Dissemination:* Prior to our work, doing a broadcast flood in a power save network gave a designer only two choices: a low-latency, high-energy flood or a high-latency, low-energy flood. We provide a framework to allow more fine-grained control where the broadcast latency can be lowered to a desirable level without immediately resorting to the highest energy state. Thus, devices can save more energy while still providing an acceptable latency for a broadcast dissemination.

Security: Multihop wireless networks give rise to a new set of security and privacy issues. The most obvious difference is the ease with which the channel can be eavesdropped. With wired networks, it takes significantly more skill to find and access a network cable, splice it, and interpret its signals. With wireless networks, even so-called “script kiddies” with little hacking expertise can download programs to tap the wireless channel and view packets [30, 31].

This real-world implications of this problem were greatly exacerbated when a fundamental flaw was discovered in 802.11’s security protocol, WEP, by Fluhrer, Mantin, and Shamir [32] that allows the network key to be discovered by sniffing encrypted packets. A popular implementation of this attack, AirSnort [33], requires about 5-10 million packets to be overheard to crack the key, but other tools, such as aircrack [34] use statistical methods to significantly reduce this number to the order of hundreds of thousands of packets on average. Additionally, these tools can use packet injection techniques to actively force a vulnerable network to generate more packets for collection. Even with the WPA protocol that replaces WEP, many users remain vulnerable to standard dictionary-based password attacks that circumvent encryption.

A second issue with multihop wireless networks is that devices may be pushed farther away from a trusted infrastructure. Thus, protocols must allow nodes to establish security among neighbors and multihop endpoints without having a direct connection to a trusted entity.

Finally, the resource constraints of many devices may require new techniques to provide security and privacy. On wired networks, public-key cryptography has been extremely effective in creating a secure system of communication. However, the hardware used for sensors may have the resources to do only symmetric key operations which are orders of magnitude more efficient.

Our work looks at the problem of key distribution in the context of wireless sensor networks where devices are resource-constrained, can do only symmetric key cryptography, and cannot communicate with a trusted source after deployment. Shared keys are fundamental in providing confidentiality and integrity of data packets in such systems. In Chapter 6, we propose *leveraging the channel diversity available in wireless networks for key distribution.* Our work exploits the underlying wireless channel diversity to address the problem. In doing so, our key distribution protocol effectively addresses drawbacks in connectivity and attacker resilience of previous techniques. Additionally, we look at using path diversity to further improve security.

1.1 Main Contributions

The main contributions of our work are as follows:

- We propose carrier sensing techniques to improve the energy efficiency of power save protocols and demonstrate its use with both in-band and out-of-band protocols (these terms will be defined in Chapter 2).
- We propose adaptive sleeping and listening for in-band protocols to reduce energy consumption. This compliments our earlier work of adaptive sleeping for out-of-band protocols [27–29].
- We propose a probabilistic approach for broadcast dissemination that allows a tradeoff in energy, latency, and reliability. This gives users fine-grained control of these metrics to reduce energy consumption while maintaining a desired latency and reliability. We implemented our protocol in TinyOS [11] to demonstrate its effectiveness on sensor hardware.
- We propose using the underlying channel diversity to improve security in resource-constrained networks. We design a protocol for symmetric key distribution that improves connectivity and resilience to adversaries when compared with previous work.

1.2 Dissertation Outline

In Chapter 2, we review past work related to our dissertation. In Section 2.1, we discuss power save protocols and describe *in-band*, and *out-of-band* protocols. In Section 2.2, we give an overview of protocols for the efficient propagation of broadcasts. Section 2.3 reviews work in symmetric key distribution for sensor networks.

In Chapter 3, we propose carrier sensing techniques to improve energy efficiency. In particular, we observe that the energy nodes spend listening to detect a signal to wake up can be significantly reduced by using the carrier sensing capabilities. We demonstrate how this technique can be used to augment both synchronous and out-of-band protocols.

Chapter 4 explores methods of adaptive energy saving. By dynamically adjusting listening and sleeping intervals, we reduce the energy consumption of in-band protocols (our previous work addressed adaptive techniques for out-of-band protocols [27–29]). We look at both link layer and network layer protocols.

In Chapter 5, we quantify the effects on energy-saving on the latency and reliability of applications which propagate information via multihop broadcast. We develop a simple, lightweight protocol that can augment existing power save protocols to achieve a desired tradeoff among energy, latency, and reliability. This allows broadcast propagation to be energy efficient while still achieving a desired latency and reliability. We also describe our implementation of the protocol in TinyOS [11].

In Chapter 6, we develop a protocol for symmetric key distribution to address security in multihop wireless networks. Our approach leverages the underlying channel diversity to create pairwise symmetric keys that, with high probability, are known to only the two communicating nodes. Our results demonstrate that the protocol performs well in connectivity and resilience to adversary devices.

Chapter 7 concludes the dissertation and offers some directions for future work.

Chapter 2

Related Work

In this chapter, we discuss related work for power save and key distribution in multihop wireless networks. In Section 2.1, we give an overview of the power save problem and define in-band and out-of-band protocols in Section 2.1.1. In Section 2.1.2 and Section 2.1.3, we focus on related work for our carrier sensing and adaptive energy saving techniques, respectively. Section 2.2 reviews previous work in efficient broadcast propagation. We discuss key distribution in sensor networks in Section 2.3.

2.1 Power Save Protocols

The fundamental question power save protocols seek to answer is: *When should a radio switch to sleep mode and for how long?* In Section 2.1.1, we broadly categorize protocols as either *in-band* or *out-of-band*. In-band protocols do all wake-up signaling on the data channel. By contrast, out-of-band protocols use a separate, orthogonal channel to do the wake-up signaling.

We note that the focus of this dissertation is on power save protocols to reduce idle listening energy. A vast area of research exists in energy efficient wireless transmission (e.g., power control, physical layer encoding) that is independent from our work. We do not discuss these techniques in this work, but refer interested readers to [35–38] and references therein for discussion of these techniques.

2.1.1 Taxonomy

In-Band Protocols: These protocols use one channel for both wake-up signaling and data communications. The most obvious advantage to this approach is that devices only need one half-duplex channel, which is available on any wireless device. A disadvantage is that the signaling overhead may now interfere with data communication. Protocols in this class can be sub-categorized as *synchronous* or *asynchronous* as described below.

- *Synchronous Protocols:* Nodes schedule a time in the future to wake up. The scheduled time can be absolute (e.g., using synchronized clocks to wake up at certain epochs) or relative to some event (e.g., a node wakes up T seconds after the last packet reception). One example [7] is IEEE 802.11's Power Save Mode (PSM) where all nodes wake up and remain on for a fixed time at the start of each beacon interval. Another example [27–29] is two communicating nodes that wake up T seconds after the last packet reception and T is adjusted dynamically based on traffic patterns.

  Protocols that require global synchronization need some external synchronization mechanism. If available and operating in the proper environment (e.g., outdoors), GPS could be used for this purpose. For a survey of other synchronization protocols, see [39]. Recent synchronization protocols for sensors [40] demonstrate precision on the order of *a microsecond*. For the purposes of our work, we assume that some such external mechanism is available.

- *Asynchronous Protocols:* Nodes wake up independently according to their own schedule and try to discover other nodes that are also awake. When the wake-ups of two nodes overlap, they can communicate. For example [41, 42], nodes may choose deterministic schedules to guarantee overlap within a bounded latency. Another example [43, 44] is nodes that wake up non-deterministically such that overlap is within a bounded time with high probability.

Generally, these techniques are orthogonal. For example, a node could use an asynchronous protocol to discover neighbors and, after discovery, use a synchronous protocol to schedule subsequent wake-ups. Similarly, an out-of-band protocol (described later in this section) could be used to wake up a neighbor to send the first data packet and synchronous wake-ups could be scheduled for later packets (see [27–29] for an example of this combination of techniques).

We begin by describing 802.11 PSM [7]. Most of our work on in-band protocols focuses on improving the 802.11 PSM design. The reasons for this are two-fold. First, it has the most complete specification of any open standard power save protocol. Second, almost any protocol that schedules a wake-up time when a node and all of its neighbors will be awake bears a strong resemblance to 802.11 PSM's design. As an example, S-MAC [45], a synchronous wake-up protocol for sensors, basically uses the same design as 802.11 PSM with minor differences. So, the 802.11 PSM design is versatile and is the basis for many synchronous protocols.

In 802.11 PSM [7], nodes are assumed to be synchronized and awake at the beginning of each *beacon interval*. After waking up, each node stays on for a period of time called the *Ad hoc Traffic Indication Message (ATIM) window*. During the ATIM window, since all nodes are guaranteed to be listening, packets that have been queued since the previous beacon interval are advertised. These advertisements take the form of ATIM packets. More formally, when a node has a packet to advertise, it sends an ATIM packet to the intended receiver during the ATIM window (following IEEE 802.11's CSMA/CA rules). In response to receiving an ATIM packet, the destination will respond with an ATIM-ACK packet (unless the ATIM specified a broadcast or multicast destination address). When this ATIM handshake has occurred, both nodes will remain on after the ATIM window and try to send their advertised data packets before the next beacon interval (subject to CSMA/CA rules). If a node remains on after the ATIM window, it must keep its radio on until the next beacon interval [7]. If a node does not send or receive an ATIM, it will enter sleep mode at the end of the ATIM window until the next beacon interval. This process is illustrated in Figure 2.1. The dotted arrows indicate events that cause other events to occur. Node **A** sends a data packet to **B**, while **C**, not receiving any ATIM packets, returns to sleep for the rest of the beacon interval.

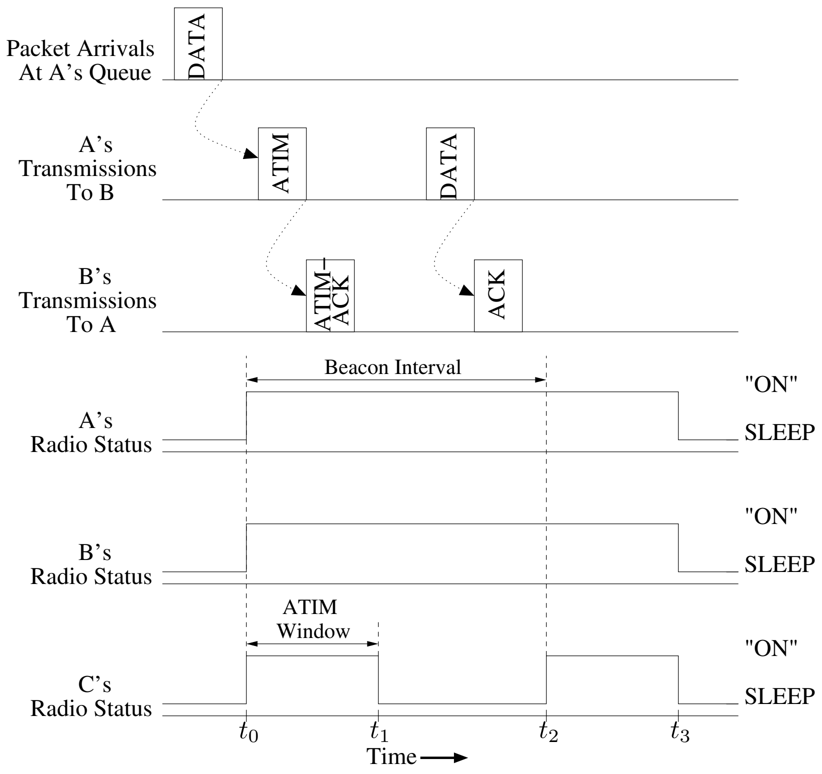

Figure 2.1: IEEE 802.11 IBSS power save mode [7].

S-MAC [45] is similar to 802.11 PSM, but with some modifications specifically for sensor networks. It reduces energy consumption at the expense of fairness and latency. S-MAC uses a simple scheduling scheme to allow neighbors to sleep for long periods and synchronize wake-ups. A group of nodes synchronize by one node broadcasting a duration of time it will be awake. After this period, the node will sleep for the same amount of time. Each node will follow this sleep/awake schedule also and broadcast it to their neighbors. If a node receives two different schedules, it will remain awake according to both schedules. In S-MAC, nodes enter sleep mode when a neighbor is transmitting and fragment long packets to avoid costly retransmissions. After each fragment, an ACK is sent by the receiver so that nodes waking up in its vicinity will sense the transmission.

TRAMA [46] uses TDMA to schedule queued packets. The TDMA scheduling is done based on an election algorithm within a node's two-hop neighborhood to ensure that every node has a slot to transmit data to its receiver while avoiding collisions. Also, when a node does not have anything to send in its assigned slot, other transmitters may use the slot.

Other protocols use TDMA to schedule "flows" of data packets [47, 48] where periodic flows try to find slots to transmit data at regular intervals without interfering with existing flows. Thus, in these protocols, the wake-up procedure requires the sender/receiver pair to wake up during a slot when they will have exclusive access to the medium. In both [47] and [48], non-interfering slots are discovered listening to the beginning of a slot for transmissions. If no transmission is detected within a specified time, a node can claim the slot for its flow. Then, in subsequent cycles, the node can always transmit a packet in that slot without other nodes interfering.

In the asynchronous protocol presented in [42], nodes choose their awake times such that they are guaranteed to overlap with each neighbor's awake time within a bounded time period. In [42], three protocols are proposed that allow neighbors to advertise to each other by guaranteeing some overlap in their awake windows. The first protocol calls for nodes to be awake for at least half of each beacon interval and alternate their advertisement windows at the beginning and end of intervals. This guarantees overlap but requires significant energy consumption. The second approach requires the nodes to stay awake for a long active interval only once every T beacon intervals. During the other $T - 1$ intervals, the node will wake up for only the duration of an advertisement window. The final approach is quorum-based. In this approach, each node picks $2n - 1$ out of n^2 intervals (where n is a specified value) in such a way that at least two chosen intervals are guaranteed to overlap with a neighbor's. Each chosen interval, the node will stay awake for the entire interval. During the other intervals, the node will stay awake for only an advertisement window. The authors note that broadcast is still difficult in such a scheme. Also, these methods require all nodes to have the same advertisement window and beacon interval lengths.

In [41], a deterministic protocol for neighbor discovery is presented. Sleep schedules are chosen such that every pair of neighbors is guaranteed to overlap for at least one slot. Thus, if a node is awake X out of Y slots, energy consumption is reduced and all neighbors can contact each other within Y slots to synchronize their communication.

In [44], a non-deterministic approach is used for neighbor discovery. Nodes wake up probabilistically in each slot and can communicate only with other nodes that are also on in that slot. Thus, this is a nondeterministic protocol where a node is awake for randomly chosen X time slots out of Y (where $X \ll Y$). Each node enters a listen or transmit mode with a specified probability such that, with high probability, neighbors will discover each other over some time interval.

The protocol in [43] is based on continuum percolation theory. Packets are broadcast throughout a network of nodes following independent sleep schedules. A packet sender broadcasts a packet until most of its neighbors are likely to have received the packet with high probability.

Out-of-Band Protocols: In this domain, a node's data radio sleeps until an out-of-band channel alerts it to wake up. An example [49,50] is a low-power radio idly listening on a separate, wake-up channel. Another example [2,3] is a wake-up radio that periodically idly listens to the channel. In both examples, when a wake-up signal is detected, the data radio turns on. The out-of-band channel is non-interfering with respect to the data channel. The disadvantage of this approach is the hardware complexity and potentially increased bandwidth usage. However, the advantage is that no coordination is required to avoid interfering with data packets and that the wake-up radio may be designed to use less power than the data radio.

Examples of out-of-band protocols include PicoRadio [49,51–53] which uses a low-power hardware device to serve as a wake-up channel with a low idle listening cost. A MAC protocol has been designed that allows nodes to wake up a neighbor when data needs to be sent. When a node wishes to send data, it encodes the neighbor's receiving channel in a beacon on the wake-up channel. The nodes then communicate over the high powered data channel of the receiver. This design uses a CDMA scheme that requires each neighbor within a 2-hop range to be assigned a unique channel and discover and maintain the channel IDs for each 1-hop neighbor, which is difficult in a distributed setting. Also, the channel ID is encoded in the wake-up signal, which increases the hardware complexity. Table 2.1 shows the target specifications for the PicoRadio hardware.[1]

Similar to PicoRadio, in [54] as well, a hardware design for a wake-up radio is presented. A wake-up channel is also used in [50]. Here, a low-power radio is integrated with a PDA. The protocol is implemented from off-the-shelf hardware. The devices register their presence with a server via a proxy. When another node wishes to communicate, the proxy will send a short wake-up packet over the low power, low bit rate

[1]These values were obtained in an email correspondence with Brian Otis, while he was at the University of California–Berkeley.

Table 2.1: Target specifications for PicoRadio hardware.

	Wakeup Radio	**Data Radio**
Transmit Power (μW)	1000	1000
Receive/Idle (μW)	50	1000
Sleep (μW)	—	0
Bitrate	$\sim$ 100 bps	50 kbps
Range (m)	10	10
Transition Energy, sleep→idle	—	$1\,\mu\text{s} \times 1\,\text{mW}$
Transition Energy, idle→sleep	—	$1\,\mu\text{s} \times 0\,\text{mW}$

channel. This will cause the high powered radio to turn on so that data communication can begin. However, this protocol is designed for systems with centralized access points or proxies.

In [55], paging interfaces are used so a base station can wake up certain nodes when it has data to send. Here a base station uses RFID tags to wake up devices that could be in any one of L sleep states. Each sleep state uses less power in steady state, but requires more delay and power when transitioning to the fully awake state. A device will remain in a power save state at least long enough to get a positive energy gain before transitioning to the next lower power state. The base station tracks this cycle for each device and when it has data to send, it waits as long as possible before waking the device and transmitting subject to QoS requirements. When the base station wishes to wake a device up, it pages all devices in that current sleep state. The non-target devices in the paged sleep state will then start the sleep cycle again once they determine that the data is not for them. This allows the size of the paging message to be on the order of the number of sleep states instead of the number of nodes.

Another work [56] uses out-of-band channels to pipeline wake-ups. This allows a node receiving a data packet to start waking up the next node on the path using the out-of-band channel. Thus, the data reception and wake-up process occur in parallel.

STEM and STEM-BT [2, 3] are also out-of-band wake-up protocols. In Chapter 3, we propose carrier sense techniques can be applied to these protocol. Thus, we defer a detailed description of these protocols to Section 3.2.1.

The PAMAS protocol [57] adapts basic mechanisms of IEEE 802.11 [7] to a two-radio architecture. PAMAS allows a node to sleep to avoid overhearing a packet intended for a different destination or to avoid interfering with another node's reception by transmitting. The control channel is used to exchange RTS/CTS packets, emit busy tones to eliminate interference, and probe ongoing communications for their duration. Whenever a node awakes and detects another transmission, it can probe the control channel to determine how much longer this transmission will continue. Unlike our work, it ignores the idle listening problem.

2.1.2 Carrier Sense Techniques

The idea of preamble sampling has been used with B-MAC [26]. The basic idea of preamble sampling is that the packet preamble is long enough to be detected by all nodes that are periodically sampling the channel in between sleep periods (i.e., the preamble must be slightly longer than the sleep time between sampling periods). When sleeping nodes sample the channel and detect the preamble, they remain on to receive the entire packet.

WiseMAC [58] improves on B-MAC by having nodes store the next sampling time of a node with which it is sending packets. Thus, after accounting for the maximum clock drift since the last packet was sent, a node can usually transmit a much shorter preamble than is required by B-MAC and, therefore, greatly improves energy consumption. While preamble sampling is similar to one of our proposed carrier sensing techniques in Section 3.1, some key differences are discussed in Section 3.1.2.

2.1.3 Adaptive Energy-Saving Techniques

Most adaptive protocols try to adjust sleeping and/or listening intervals based on traffic in the network. Another class of adaptive power save protocols adjust in response to the topology. We primarily focus on the traffic-based approaches because they relate closely our work. However, at the end of this subsection, we mention the topology-based research.

In Table 2.2, we give classify our previous work [27–29] in relation to our work in this dissertation. In [27–29], we used synchronous wake-ups to adaptively sleep in an out-of-band protocol. Nodes engaged in communication schedule times in the future to wake up based on past traffic patterns. In our protocols [27–29], nodes dynamically adapt to changing traffic rates try to minimize energy consumption for their communication. The adaptive listening techniques from Section 4.1 could be applied to our previous work as well.

Table 2.2: Classification of our work.

	Adaptive Listening	**Adaptive Sleeping**
In-Band	Section 4.1	Section 4.2
Out-of-Band	Techniques from Section 4.1 applicable	Our previous work [27–29]

In [59], it is shown that the static ATIM window of 802.11 PSM does not work well for all traffic loads. Intuitively, higher traffic loads need larger ATIM windows. This observation motivates our adaptive design in Section 4.1 that dynamically adjusts the ATIM window.

Other works have also proposed dynamic ATIM window adjustment. DPSM [60] is designed for single-hop networks (i.e., WLANs) and uses indications such as the listening time at the end of the ATIM, the number of packets pending for a node, and the number of packets that could not be advertised in the previous beacon interval. Unlike our work, this protocol adjusts the current ATIM window based on traffic in past beacon intervals. By contrast, our protocol adjusts the current ATIM window based on the traffic in the current beacon interval. IPSM [61] is similar to our work in that the ATIM window ends when the channel is idle for a specified amount of time. However, IPSM works in only single-hop networks since it relies on a node and all its neighbors having a consistent view of channel activity. Unlike DPSM and IPSM, all of our protocols are designed for multihop networks.

In TIPS [62], the ATIM window is divided into two slots. If a beacon packet is received during the first slot, it indicates that nodes should stay on to receive ATIMs later in the ATIM window. If the first beacon packet is not received until the second slot, then the node can return to sleep since no more advertisements will follow. In our work, carrier sensing is used as an indication that nodes should remain on longer. The time it takes to carrier sense is usually much shorter than the time it takes to access the channel and send an entire packet. Additionally, TIPS uses only static ATIM window sizes whereas our techniques allows dynamic adjustment of the window.

T-MAC [63] extends S-MAC by adjusting the length of time sensors are awake between sleep intervals based on communication of nearby neighbors. Thus, less energy is wasted due to idle listening when traffic is light. In the T-MAC work, the authors refer to the *early sleeping* problem that occurs when a node returns to sleep when one of its neighbors has data to send to it but is deferring to another sender. Essentially, this early sleeping problem is the main problem that we address in Section 4.1. The two techniques that T-MAC proposes to address the problem are not applicable to advertisement windows. One reason is that T-MAC is designed for relatively large data packets and relies on short RTS and CTS control packets to address the early sleeping problem. By contrast, since the ATIM and ATIM-ACK packets exchanged in the advertisement window are about the same size as RTS and CTS packets, it would be a significant increase in overhead to precede the ATIM/ATIM-ACK handshake with RTS and CTS packets. Also, T-MAC results in an increase in energy consumption to improve throughput. In our work, we prefer reducing energy consumption over increasing throughput. Our work addresses the early sleeping problem *without* inducing any extra control overhead (since the ATIM/ATIM-ACK packets are already small) and is designed to reduce energy consumption, not improve throughput.

In [64, 65], modifications are made to S-MAC to reduce the multihop delay of packet forwarding. Also in [64], a global scheduling algorithm is developed for S-MAC to converge to one sleep schedule in the network. The carrier sensing techniques in Section 3.1 could be used to complement the S-MAC protocols.

In [66], a protocol is proposed that works with on-demand routing and uses 802.11's PSM when a node is not engaged in sending, receiving, or forwarding data. When a node is communicating, soft-timers are used to transition the node to an idle listening mode that reduces latency and preserves throughput better than using only 802.11's power save. However, the timers do not adjust to the traffic rate, so if traffic is not frequent enough to refresh the timers, the benefits of the protocol are lost. Nodes must promiscuously listen to the packets of neighbors to determine if they are disconnected or in power save mode. In this sense, the protocol does a coarse-grained form of adaptive sleeping based on whether a node is forwarding traffic. Our approach in Section 4.2 takes a much more fine-grained adaptive sleeping approach based on the desired latency of an application. TITAN [67] extends the work from [66]. In TITAN, route requests are delayed by sleeping nodes to allow the route discovery procedure to favor nodes that are already in the idle listening state. This helps reduce the overall energy consumption in the network.

LISP [68] is an extension to 802.11 PSM where nodes try to predictively remain on after the ATIM window to forward multihop traffic at a lower latency. When a node is scheduled to sleep at the end of an ATIM window, it may remain on based on correlations between overheard ATIM-ACKs and previous ATIM/ATIM-ACK handshakes. Our adaptive sleeping technique, on the other hand, attempts to achieve a *latency bound* while still conserving as much energy as possible.

In [69], ESSAT is designed to handle Constant BitRate (CBR) traffic in sensor networks. In particular, ESSAT predictively wakes up downstream neighbors based on past reception times for CBR flows. The wake up times are adjusted when phase shifts occur in the flow due to packet loss and contention. Our protocols are designed for traffic that is not necessarily CBR.

Topology Adaptive Protocols: Another common strategy is for nodes to remain awake based on their local topology and/or traffic [70, 71]. Work in this area investigates how a subset of the nodes in a system can enter a low power state without significantly degrading the performance achievable if all nodes were to remain in high power mode.

The AFECA algorithm [72] allows nodes to sleep based on the size of their neighborhood. If node density is large, then more nodes can sleep without greatly increasing the latency of data flows. GAF [71] assumes the nodes have some location information and form virtual grids. The size of the grids is chosen such that the nodes in two adjacent grids are equivalent with respect to forwarding packets. Then, within each grid, a discovery protocol tries to ensure that most of the time one node remains active while the rest enter a low-power state. As mobility increases, the discovery process should be more frequent.

The goal of SPAN [70] is to save energy while not degrading the latency and throughput achievable in 802.11 without power save mode. This protocol operates between the MAC and routing layers. The system allows all nodes to enter power save mode except for elected *coordinators*. At the MAC layer, nodes periodically exchange *hello* messages that contain its set of neighbors, coordinators, and whether it is a coordinator. Nodes will then elect themselves coordinators if their neighbors would get better connectivity by it doing so. A random delay is introduced before nodes declare themselves coordinators. This delay varies inversely with the amount of connectivity that would be achieved and inversely with the amount of energy remaining at the node. For fairness, the coordinators will periodically withdraw.

2.2 Efficient Broadcast Propagation

Broadcast is prevalent in wireless networks as a means to propagate information. The application on which we focus in testing our protocol in Chapter 5 is code distribution, whereby a source periodically sends out patches for sensors to apply to their software. In [73], the authors demonstrate a software architecture to allow the application of such updates. In other works [74–76], the focus is on reducing the flooding overhead for disseminating code updates. In our work, we look at the effects on energy-saving on the reception rate of code updates. Other applications of multihop broadcasts include route discovery in ad hoc routing protocols [77, 78] and querying for sensor data [79].

One popular method for reducing the overhead of broadcast is to form a backbone in the network where only certain nodes forward data [80–82], which can reduce overhead. Another method, which is most similar to our work in Chapter 5, is probabilistic broadcast [83–85], where nodes only forward a broadcast with some probability, p. By doing this, the broadcast is capable of reaching most of the nodes in the network while reducing the overhead. This is based on the observation that a broadcast flood typically has a high level of redundancy [86]. In our protocol, we try to use this redundancy to reduce the *energy* consumed by the broadcast.

2.3 Symmetric Key Distribution

In Chapter 6, we propose a novel method of symmetric key establishment for a sensor network that uses channel diversity, as well as spatial diversity, to create link keys for one-hop neighbors. Given this protocol, we characterize the tradeoffs that arise in energy and security. Establishing such keys is important because public keys are too computationally intensive for many sensors. Sharing a symmetric key with neighbors allows for secure aggregation as well as transmitting data used to authenticate hash chains, for example. In this section, we give an overview of five approaches to the problem.

Trusted Intermediary: This approach is similar to that of Kerberos [87], which is used widely on wired networks. Every node shares a secret symmetric key with a trusted intermediary that is loaded before deployment. The key establishment protocol requires two sensors that wish to establish a pairwise key to communicate with the trusted intermediary to create the key. SPINS is an example of this approach [88] A disadvantage of this approach is that the server may become a bottleneck in large networks and that the trusted intermediary must be online whenever key establishment is desired.

Key Predistribution: Eschenauer and Gligor [89] were among the first to consider key predistribution for sensor networks. In their work, referred to here as the basic scheme, sensors are loaded with randomly chosen keys out of a master key pool before deployment. After deployment, a sensor can securely communicate with its neighbors if it shares at least one key in common with the neighbor. Chan et al. [90] extend the basic scheme to require neighbors to share q keys in common before a link is possible. This improves security at the cost of decreased connectivity. Their work also proposes the idea of using multiple node disjoint paths to strengthen security. This is a different form of diversity than what we propose, but demonstrates how the concept can improve security in diverse path selection. Other schemes propose that keys be distributed deterministically based on a sensor's ID [91, 92].

Du et al. [93] adapt a key predistribution scheme originally proposed by Blom [94] for sensor networks by using finite fields to generate multiple key spaces that can be randomly deployed to sensors. Liu et al. [95] extend a key distribution method proposed by Blundo et al. [96] that uses polynomial based distribution methods.

In Section 6.5.1, we discuss the advantages and disadvantages of our approach compared with key predistribution.

Transitory Keys: The LEAP architecture [97] provides a method of establishing pairwise keys provided sensors cannot be compromised during a short initialization phase after deployment and the sensor hardware that can be trusted to completely erase a master key after initialization. Thus, the master key is transitory at each device and only available during the initialization. Some disadvantages of this approach are that (1) if a node is compromised during the initialization period, the entire network may be compromised and (2) the low-end sensor hardware must ensure that the master key is erased from memory such that recovery is not possible.

Public Key Exchange: Public key exchanges to generate the symmetric keys for bulk encryption are used in many Internet protocols (e.g., IPsec). The Diffie-Hellman key exchange [98] and Elliptic Curve Diffie-Hellman (ECDH) [99] are two widely used protocols designed for this purpose. The primary reason for not using such protocols in sensor networks is the large computational overhead incurred by asymmetric key cryptography when compared with symmetric key protocols. However, elliptic curve cryptography has been implemented in TinyOS [100, 101]. In Table 2.3, we give the performance metrics for the two Mica2 Mote public key implementations of which we are aware, EccM 2.0 [100] and Sizzle [101].

Table 2.3: Performance of public key exchange implementations on Mica2 Motes.

	EccM 2.0 [100]	**Sizzle [101]**
Key Size (bits)	163	160
Bits of Security[a]	80	80
RAM Usage (KB)	1.03	3.08
ROM Usage (KB)	33.5	60
ECDH Time (s)	34.173	3.8

[a] "An algorithm that has a 'Y' bit key, but whose strength is comparable to an 'X' bit key of such a symmetric algorithm is said have a 'security strength of X bits' or to provide 'X bits of security'. Given a few plaintext blocks and corresponding cipher, an algorithm that provides X bits of security would, on average, take $2^{X-1}T$ of time to attack, where T is the amount of time that is required to perform one encryption of a plaintext value and comparison of the result against the corresponding ciphertext value." [102]

While computationally expensive, this approach may be acceptable in long-lived sensor networks where the cost of the key exchange is amortized over the lifetime of the sensors. We are unaware of any rigorous quantitative analysis comparing the public key exchange implementations with pure symmetric key approaches in security, performance, and memory usage. We believe that the research community could greatly benefit from such a detailed comparison. In the absence of such results, our work explores a pure symmetric key exchange approach.

Broadcasting Plaintext Keys: The work that is most similar to ours is that of Anderson et al. [6]. The protocol is based on the assumption that the number of adversary devices in the network at the time of key establishment is small (in their results, less than 3% of the nodes are adversaries). Thus, during the initialization phase, a sensor, u, will broadcast a randomly generated plaintext key, k_u, that is overheard by all its one-hop neighbors (including adversaries). Each one of u's neighbors replies with the message

$\{v, k_{uv}\}_{k_u}$, where v is the ID of the neighbor and k_{uv} is a pairwise key randomly generated by v.[2] After this exchange, u and v use key k_{uv} for communication. Power control is used to reduce the number of devices that overhear the key exchange.

In Section 6.5.2, we discuss in detail the differences between our work and Anderson's. We briefly mention these differences here. First, our protocol is much more resilient to eavesdropping by attacking devices since we leverage channel diversity and use location diversity more.[3] Second, a link cannot be authenticated in Anderson's scheme since u or v has no way to verify the sender of the messages. In contrast, we provide mechanisms that allow a trusted source to authenticate sensor IDs and broadcasted keys. Refer to Section 6.5.2 for more details about these differences.

[2] We use the notation $\{M\}_k$ to indicate a message, M, encrypted using key k.

[3] We note that the goal in [6], unlike our work, is not to make it difficult for a nearby attacker to compromise a link or to operate in hostile environments where there may be many adversaries.

Chapter 3

Carrier Sensing for Energy-Efficient Signaling

Many power save protocols described in Section 2.1 follow a common design where potential receivers periodically awake to listen for some type of wake-up signal in between long periods of sleep. However, many such protocols are inefficient from an energy perspective in that this listening period is on the order of a packet transmission time. The increase in energy consumption is particularly significant in networks with light traffic, as might be expected in many sensor applications.[1]

Based on this observation, we propose using the carrier sensing capabilities that are available at the physical layer to reduce the listening period for wake-up signals to be on the order of the time it takes to detect the channel busy. This detection time is typically much smaller than a packet transmission time. In this chapter, we demonstrate how this technique can be applied to both a in-band protocol (Section 3.1.1) as well as out-of-band protocols (Section 3.2.5 and Section 3.2.6).

3.1 Carrier Sensing for In-Band Protocols

In this section, we further discuss our proposed techniques to leverage carrier sensing for energy efficiency and demonstrate their application to an in-band power save protocol. Specifically, we look at techniques to improve the IBSS *Power Save Mode* (PSM) in IEEE 802.11 [7]. IBSS (Independent Basic Service Set) is the protocol set for ad hoc networks. While the techniques we propose are tested with 802.11 PSM, in Section 3.1.1 we discuss how they can augment other power save protocols. Our results show that the proposed improvements to 802.11 PSM can greatly reduce energy consumption with little increase in the average packet latency. Our carrier sensing protocol will be combined with an adaptive listening and adaptive

[1]We note that if traffic is heavy in a network, then using any type of power save is generally not useful.

sleeping scheme in Section 4.1 and Section 4.2, respectively. We defer the presentation of simulation results to these sections.

3.1.1 Protocol Description

We use a short carrier sensing period preceding the ATIM window where nodes can indicate whether they intend to advertise any data. Thus, when none of a node's neighbors are going to advertise any data, the node can return to sleep without remaining on for the ATIM window. In Section 4.1.1, we further improve the energy consumption of the protocol by allowing nodes that participate in the ATIM window to dynamically adjust the size of their ATIM window. By using this technique, nodes that do not receive any ATIMs can usually return to sleep sooner than if a static ATIM window size is used.

We make the assumption that the nodes in the network are time synchronized by some out-of-band means. For example, the nodes may be GPS-equipped. Later, we discuss modifications to the protocols to handle some synchronization errors. Thus, the timing synchronization function (TSF) of 802.11 is disabled and beacons are never sent. For consistency with the terminology in related work, we will still refer to the time between ATIM windows as a "beacon interval" even though no beacons are sent.

From the description of 802.11 PSM in Section 2, we observe that it is possible that most beacon intervals have no packets to be advertised. In this case, the ATIM window needlessly wastes energy. However, when traffic is queued at the beginning of a beacon interval, nodes need a mechanism to advertise their packets. Thus, the ATIM window concept cannot be completely removed. What is needed is a energy-efficient binary signal so that a node can let neighbors know when it has traffic to advertise and, hence, an ATIM window is needed for that beacon interval.

For this purpose, we propose *Carrier Sense ATIM* (CS-ATIM) that adds a short carrier sensing period at the beginning of each beacon interval as shown in Figure 3.1. The basic idea is that the time it takes to carrier sense the channel busy or idle, T_{cs}, is significantly smaller than the ATIM window, T_{aw}. Rather than every node waking up for T_{aw} at the beginning of every beacon interval, the nodes will wake up for only T_{cs} at the beginning of every interval when no packets are to be advertised in their neighborhood. When packets are to be advertised, the nodes will wake up for an entire ATIM window after the carrier sensing period.

Using Figure 3.1, we will explain how CS-ATIM works. The shaded regions in Figure 3.1 indicate that a node is transmitting a packet. At time t_0, no packets are to be advertised so all nodes wake up for T_{cs} time and return to sleep when the channel is detected idle. At time t_1, the nodes wake up for the start of the next beacon interval. This time, node **A** has a packet to advertise, so it transmits a "dummy" packet to make the channel busy. When nodes **B** and **C** finish carrier sensing the channel at time $t_1 + T_{cs}$, the

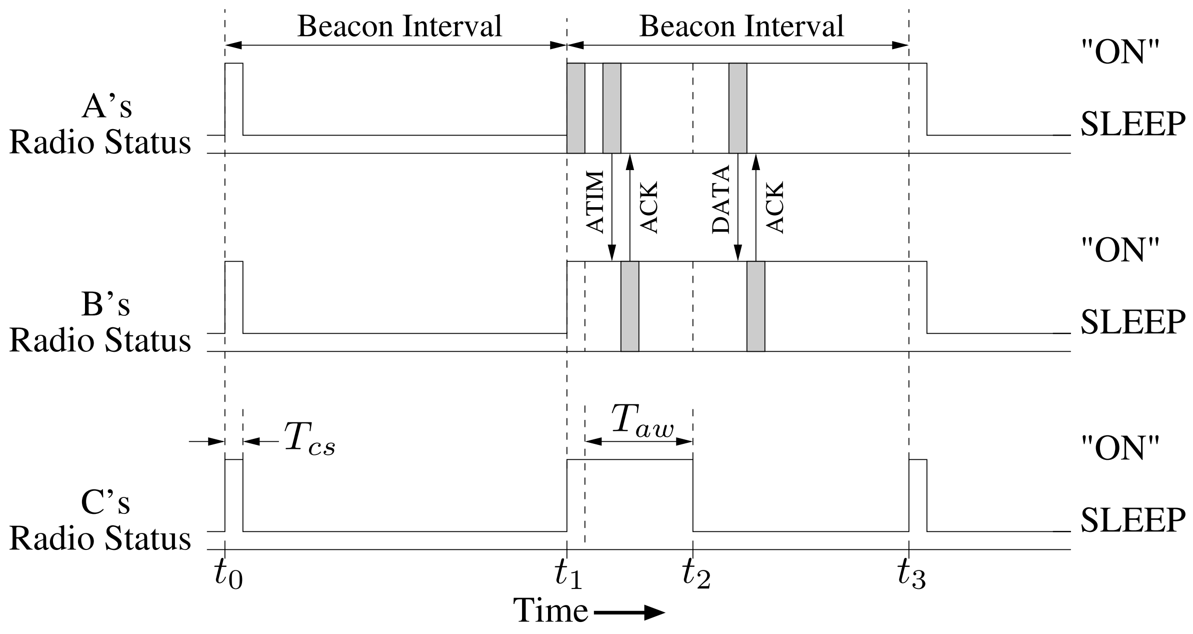

Figure 3.1: CS-ATIM protocol (T_{cs} and T_{aw} are not drawn to scale).

channel is detected busy because of **A**'s packet transmission. Thus, all nodes who carrier sensed the channel busy or transmitted a "dummy" packet will remain on for an ATIM window of length T_{aw} after the carrier sensing period. During the ATIM window, **A** sends an ATIM to **B** and **B** replies to **A** with an ATIM-ACK. Because of this exchange, **A** and **B** will remain on for the rest of the beacon interval. Because **C** did not send or receive an ATIM during the ATIM window, it returns to sleep at the end of the ATIM window at time t_2. After the ATIM window, **A** and **B** exchange the data packet and corresponding ACK. At time t_3, a new beacon interval begins and all of the nodes return to sleep after carrier sensing the channel as idle.

The value of T_{cs} is chosen to be long enough to carrier sense the channel as idle or busy with a desired level of reliability. According to the 802.11 specification [7], the clear channel assessment (CCA) for compliant hardware must be less than 15 μs. In our experiments, we use a much larger value for T_{cs} to mitigate the effects of short-term fading. The dummy packet transmitted by a node with packets to advertise does not contain any information that needs to be decoded; its only purpose is to cause other nodes to detect the channel as busy. The advantage of not having information in the dummy packet is that multiple nodes can transmit simultaneously, causing collisions at the receivers, without hindering the protocol. If a collision occurs at the receiver, it can still detect the channel as busy and remain on for the ATIM window. In the ATIM window, nodes use the standard 802.11 CSMA/CA protocol to send their ATIMs and ATIM-ACKs while avoiding collisions. A node that transmits a dummy packet cannot carrier sense dummy packets being sent by other nodes at the beginning of the beacon interval. However, this does not affect the protocol since a node stays on for the ATIM interval whenever it transmits a dummy packet *or* carrier senses the channel busy.

From this description of CS-ATIM, clearly nodes can use significantly less energy than 802.11 PSM listening at the beginning of each beacon interval when no packets are to be advertised. When packets are to be advertised, CS-ATIM uses only slightly more energy than 802.11 PSM because of the short carrier sensing period. For packet latency, 802.11 PSM does slightly better than CS-ATIM. One reason is that data packets that arrive after the carrier sensing period but before the end of the ATIM window may be sent in the current beacon interval in 802.11 PSM. In CS-ATIM, such packets may have to wait until the next beacon interval. Also, CS-ATIM has a slightly larger delay since the ATIM window does not end until $T_{cs} + T_{aw}$, whereas the 802.11 PSM ATIM window ends T_{aw} after the beginning of the beacon interval.

With CS-ATIM, we note that carrier sensing for energy on the channel, as opposed to actually decoding a packet, runs the risk that nodes may erroneously carrier sense energy that is due to interference in the frequency band rather than the dummy packet transmission. In this case, a node remains on for the ATIM window even though none of its neighbors sent a dummy packet. We refer to this as a *false positive*. In Section 4.1.3, the effects of false positives on CS-ATIM are tested.

As mentioned, CS-ATIM can be adapted to operate in networks without perfect synchronization. We assume that the node's clocks are always within Δ seconds of each other. Thus, Δ represents the maximum error between the clocks of *any* two nodes in the network. To handle synchronization errors, the following changes are made to CS-ATIM:

- At the beginning of a beacon interval, a dummy packet is transmitted for $2\Delta + T_{cs}$ time instead of T_{cs} time.

- Nodes that do not have packets to advertise begin their carrier sensing period Δ time after the beginning of the beacon interval (according to their local clock). Originally, these nodes would begin carrier sensing immediately at the beginning of a beacon interval.

- For dummy packet transmitters, ATIM windows last for $2\Delta + T_{aw}$ time instead of T_{aw} time. A node is not allowed to transmit any ATIMs for the first Δ time of the ATIM window and the last Δ time of the ATIM window. However, a node *may* send ACKs, ATIM-ACKs, and receive packets during the entire $2\Delta + T_{aw}$ duration of the ATIM window.

- For nodes that sense a dummy packet, ATIM windows last for $3\Delta + T_{aw}$ time instead of T_{aw} time. Such a node may reply to any ATIMs that they receive during this $3\Delta + T_{aw}$ period with ATIM-ACKs.

To preserve the flow of the dissertation, we move the discussion of the correctness of these modifications to Appendix A.

The basic idea from CS-ATIM can be adapted to other power save protocols besides 802.11 PSM. Whenever a node is scheduled to listen in a power save protocol, it can do carrier sensing at the start of its scheduled wake-up time to determine if it can return to sleep because no nodes have data to send. For example, in a TDMA protocol, nodes can carrier sense at the beginning of their scheduled slot and return to sleep if no data needs to be sent.

3.1.2 Comparison with Preamble Sampling

We note that the technique described in Section 3.1.1 is similar to the preamble sampling used in protocols such as B-MAC [26] and WiseMAC [58]. However, our proposed carrier sensing technique has some key differences when compared with these preamble sampling protocols. The advantage of preamble sampling is that it works in completely unsynchronized environments.[2] However, because the nodes may transmit their preamble at any time (which serves as the wake-up signal), the probability increases that the preamble will collide with an ongoing data transmission (e.g., due to hidden terminals). By contrast, our protocol restricts wake-up signals to a specific time when data packets are not being transmitted. Also, because the wake-up signal in our protocol serves as only a binary indication of whether or not the channel is busy, interference among wake-up signals is tolerable, as discussed in Section 3.1.1. With preamble sampling, interference may affect packet reception since the preamble may synchronize the bits of the incoming data packet.

Another disadvantage of preamble sampling is that broadcast, which is commonly used in wireless communication, requires a large preamble transmission. In particular, the preamble must be slightly longer than a beacon interval (which is *at least* on the order of tens of milliseconds and longer if less energy consumption is desired). This is the only way to ensure that all of a node's neighbors are able to detect the preamble. By contrast, in our protocol, the overhead for a wake-up signal is slightly longer than the time to reliably carrier sense the channel (e.g., typically on the order of tens of microseconds) for both unicast and broadcast packets.[3] Thus, the well-known overhead problem associated with broadcast storms [86] is exacerbated by the preamble sampling protocols whereas our protocol does not add any extra overhead to broadcast packets when compared with unicast packets.

[2]WiseMAC [58] does require nodes to keep track of their last communication time with each neighbor as well as the maximum possible clock drift of the hardware.

[3]To do broadcast in our protocol, a dummy packet is transmitted causing all of a node's neighbors to remain on for an ATIM window. At this point, the transmitter can simply transmit a broadcast ATIM packet as discussed in Section 2.1.1 and follow the standard 802.11 PSM protocol [7].

3.1.3 Simulation Results

See Section 4.1.3 for CS-ATIM simulation results.

3.2 Carrier Sensing for Out-of-Band protocols

In this section, we demonstrate how carrier sensing can be applied to an out-of-band power save protocol. An advantage of this technique is that, unlike synchronous protocols, no clock synchronization is needed. Unlike asynchronous protocols, nodes do not have to probe the channel whenever they wake up (i.e., less channel contention and control overhead). Also, out-of-band protocols have a deterministic bound on wake-up latency, which is not true of asynchronous protocols with non-deterministic schedules. However, tradeoffs exist when using out-of-band protocols. One disadvantage is the increased hardware complexity and cost to provide an extra wake-up channel. Also, the wake-up channel requires extra bandwidth to avoid interference with the data channel. Finally, the wake-up channel must be designed such that its monitoring does not consume much energy. Obviously, the wake-up channel is of little use, from an energy perspective, if it consumes a large amount of energy idly listening to the channel while the data radio is saving energy by sleeping.

STEM [2, 3] and STEM-BT [3] (STEM Busy Tone) are out-of-band protocols that use periodic idle listening on the wake-up channel. In this section, using carrier sensing, we identify ways to make each of these protocols more efficient.

3.2.1 Protocol Descriptions

In STEM [2, 3], a two-radio architecture achieves energy savings by letting the data radio sleep until communication is necessary while the wake-up radio periodically listens according to a duty cycle. When a node has data to send, it begins transmitting continuously on the wake-up channel long enough to guarantee that all neighbors will receive the wake-up signal. STEM-BT [3] is a variant of STEM where the wake-up radio uses a busy tone, instead of encoded data, for the wake-up signal. Both protocols are orthogonal to the data radio MAC layer transmission scheduling scheme.

In this section, we describe the operation of STEM and STEM-BT. Based on this discussion, we make some observations about how the protocols could achieve better energy efficiency using carrier sensing. Based on this, we present two new protocols, STEM-H and STEM-BT2, which reduce the energy consumption for STEM and STEM-BT, respectively. For each of the protocols, we have two sub-protocols. One is the *transmitting* sub-protocol, which is performed when a node has data to send and tries to wake up the

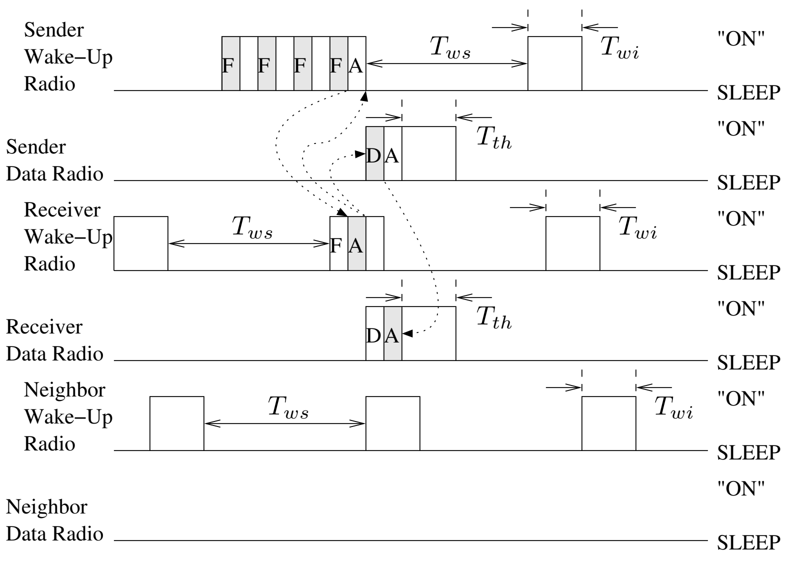

Figure 3.2: STEM protocol [2, 3].

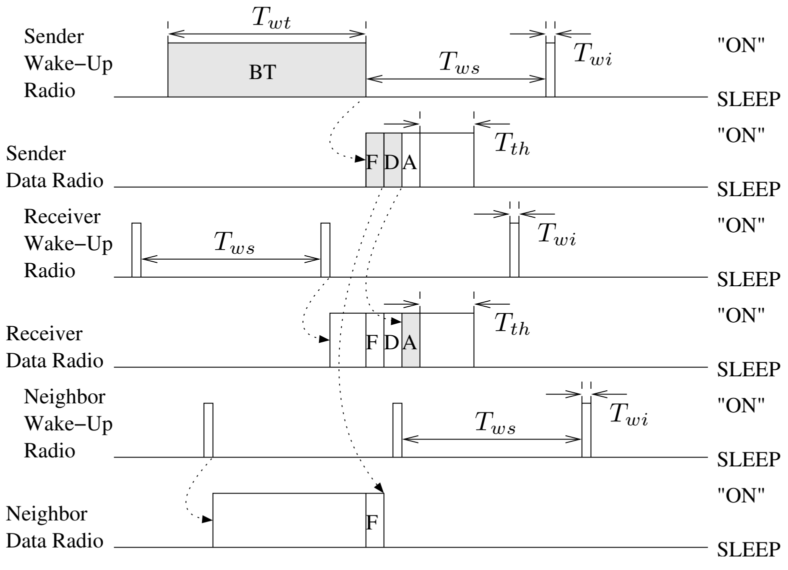

Figure 3.3: STEM-BT protocol [3].

intended receiver. The other sub-protocol is for *monitoring*; typically, the nodes spend most of their time in the monitoring state where they periodically listen to the wake-up channel to determine if a signal is being sent and they need to wake up their data radio. In STEM and STEM-H, the wake-up radio must be able to send and receive data packets. By contrast, in STEM-BT and STEM-BT2, the wake-up radio needs to be able to only send and detect a busy tone (i.e., making a binary decision whether the channel is busy or not). Figures 3.2, 3.3, 3.4, and 3.5 give a pictorial example of the protocols described below. In each of these figures, the arrows show a "causes" relationship between events. The key for the figures is: **F** is a filter packet, **D** is a data packet, **A** is an ACK packet, and **BT** is a busy tone. When **F**, **D**, **A**, or **BT** is in a

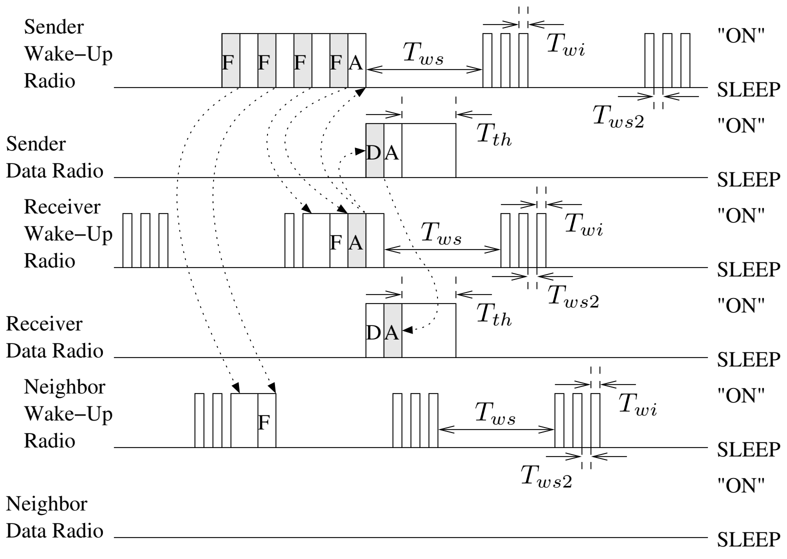

Figure 3.4: Proposed STEM-H protocol.

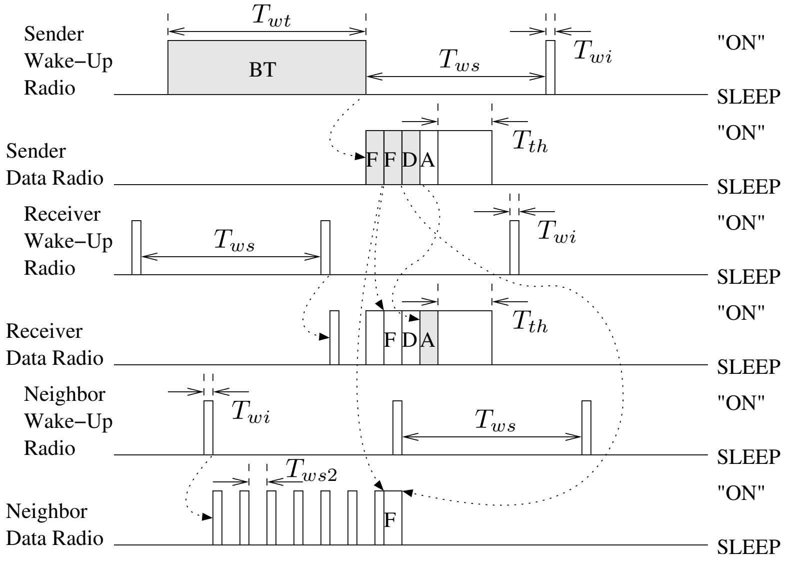

Figure 3.5: Proposed STEM-BT2 protocol.

shaded area, a node is sending; otherwise, a node is receiving.

3.2.2 STEM Description [2, 3] (Figure 3.2)

Sending Protocol: When a node has data to send, it begins a continuous cycle of transmitting a *FILTER* packet on the wake-up channel followed by a idle listening period for the corresponding *FILTER-ACK* packet. The idle listening time in between FILTERs, denoted as T_A, has to be long enough to receive a FILTER-ACK. Thus, T_A is the time to transmit a FILTER-ACK plus extra time due to factors such as propagation delay and hardware switching time.

When a sender gets the corresponding FILTER-ACK, it turns on its data radio and begins sending data packets according to the data radio MAC protocol. At this point, the sender stops sending FILTERs and the wake-up radio enters the monitoring state. For a discussion of how wake-up channel collisions are handled, see [2, 3]. When a node with its data radio on does not send or receive packets for an idle threshold time, T_{th}, it returns the data radio to sleep.

Monitoring Protocol: Nodes periodically wake up long enough to receive a FILTER and respond with a FILTER-ACK if they are the intended receiver. After idly listening for some time, T_{wi}, the node's wake-up radio returns to sleep for a period of time, T_{ws}. The T_{ws} value is chosen by the user. A long T_{ws} saves more energy in the monitoring state, but increases the wake-up process latency. T_{wi} is a function of T_F and T_A [2].

When a node receives a FILTER and it is the intended receiver, it sends a FILTER-ACK and turns its data radio on to idly listen for packets on the data channel. On the wake-up channel, the node continues in the monitoring state. When a node with its data radio on does not send or receive packets for an idle threshold time, T_{th}, it returns the data radio to sleep.

3.2.3 STEM-BT Description [3] (Figure 3.3)

Sending Protocol: When a node has data to send in STEM-BT, it starts transmitting a busy tone on the wake-up channel. The busy tone is sent for T_{wt} time, long enough to guarantee overlap with every neighbor's wake-up channel carrier sensing period.

After the sender has transmitted a busy tone for T_{wt} time, it turns on its data radio. Once the data radio is on, a FILTER packet is sent on the data channel indicating which receiver will receive more data. The sender then begins transmitting the data to the receiver on the data channel. As in STEM, when a node with its data radio on does not send or receive packets for T_{th} time, it returns its data radio to sleep.

Monitoring Protocol: For monitoring nodes, the protocol is similar to STEM's. A difference between the monitoring protocol of STEM and STEM-BT is the length of T_{wi}, the carrier sensing time. In STEM-BT, T_{wi} is shorter because a monitoring node has to only detect a busy tone. On the other hand, in STEM, the monitoring node has to decode a packet and send a FILTER-ACK if it is the intended receiver.

When a node detects a busy tone, it turns on its data radio and idly listens for a FILTER packet on the data channel. When the FILTER packet is received, the node remains on if it is the intended receiver. Otherwise, its data radio returns to sleep. If a node keeps its data radio on to receive data packets, it returns the data radio to sleep when no packet has been sent or received for T_{th} time. One key point about

STEM-BT's monitoring protocol is that *all* one-hop neighbors of the sender must turn their data radio on and idly listen until the FILTER packet is received.

3.2.4 Discussion of STEM and STEM-BT

Based on these protocol descriptions, we make a couple of observations. First, the wake-up process of STEM is relatively inexpensive (in energy consumption) for all nodes other than the sender when compared with the steady-state monitoring process. In particular, neighbor nodes use almost the same amount of energy whether they are monitoring the channel or receiving the FILTER packet.[4] The receiving node uses slightly more energy because it responds with a FILTER-ACK packet. While the sender uses more energy due to its FILTER transmissions, it transmits FILTERs for only $T_{wt}/2$ time on average. By contrast, the wake-up procedure for STEM-BT is relatively expensive compared with STEM. In STEM-BT, every neighbor node that detects the busy tone turns its data radio on to listen for the FILTER packet on the data channel. Thus, on average, each neighbor node idly listens to the data channel for half of the time that the busy tone is emitted. Based on this, we can conclude that STEM-BT's performance degrades when (1) a large number of neighbor nodes are in the vicinity of the sender[5] or (2) wake-ups become more frequent (e.g., due to a higher traffic load). STEM's wake-up procedure, however, is relatively inexpensive and does not greatly increase energy consumption as the size of the sender's neighborhood increases.

The second observation is that STEM's steady-state monitoring process is relatively expensive when compared with STEM-BT. Since T_{wi} is proportional to T_F and T_A, it can be large for sensor networks which tend to have a relatively low bitrate (e.g., 19.2 kbps for Mica2 Motes [25]). All nodes in the network must spend $\frac{T_{wi}}{T_{ws}+T_{wi}}$ fraction of the time idly listening even if no network traffic is present. In STEM-BT, T_{wi} is significantly smaller since it is only long enough to detect if a busy tone is being emitted. In STEM-BT, T_{wi} is independent of the size of FILTER and FILTER-ACK packets. Thus, the monitoring process in STEM-BT uses much less energy than STEM. For example, as we will see in Section 3.2.8, T_{wi} is about 80 times larger for STEM than for STEM-BT for our simulation parameters. Thus, when the traffic load is low in a network, STEM-BT is more energy efficient than STEM because of its low monitoring costs.

3.2.5 Proposed Protocol Description: STEM-H (Figure 3.4)

Based on the discussion in Section 3.2.4, we see that STEM's energy consumption can be improved by reducing the monitoring costs while retaining its relatively low wake-up cost. Thus, we propose STEM-H

[4]We assume the idle listening power and receiving power are about the same.

[5]Many applications assume that sensor networks can be rather dense for reasons such as increased reliability, connectivity, and adequate sensing coverage.

(STEM Hybrid) which combines aspects of STEM and STEM-BT to create a protocol more energy efficient than STEM. The basic idea of STEM-H is to idly listen only long enough to carrier sense whether the wake-up channel is busy during the monitoring phase. This detection time is relatively small (e.g., similar to STEM-BT). When the wake-up channel is carrier sensed busy, then the monitoring node leaves its wake-up radio on to receive and decode a FILTER packet.

Sending Protocol: The sending protocol is identical to STEM's, described in Section 3.2.2. The only difference is in the length of T_{wt}, which guarantees sufficient overlap with STEM-H's monitoring protocol.

Monitoring Protocol: For STEM-H's monitoring state, nodes wake up only long enough to carrier sense whether the wake-up channel is busy or idle. This differs from STEM's monitoring protocol, where nodes wake up long enough to receive FILTER packets and send FILTER-ACKs. STEM-H's monitoring protocol has two phases. In the first phase, nodes sleep for T_{ws} time. In the second phase, nodes wake up and periodically carrier sense the wake-up channel, then return to the first phase. This is shown in Figure 3.4, where in between phase one periods of length T_{ws}, nodes probe the wake-up channel multiple times. During the second phase, if the wake-up channel is detected as busy, a node stays on to receive the next FILTER and, if necessary, send a FILTER-ACK. Like STEM, nodes reply with a FILTER-ACK if the FILTER is for them. Otherwise, the node returns to its regular monitoring state. Once the FILTER/FILTER-ACK handshake occurs, nodes follow the same procedure for turning on their data radios as described in Section 3.2.2. They also follow the same protocol for returning their data radios to sleep. It is important to note that in STEM-H, T_{wi}, the carrier sensing time when probing the wake-up channel, is comparable to that of STEM-BT since only a binary decision on the channel status is necessary. STEM, on the other hand, requires a much longer T_{wi} because it must completely decode packets during its idle listening period on the wake-up channel.

Discussion: STEM-H improves the energy consumption of STEM by reducing the monitoring process cost while keeping the benefits from the relatively inexpensive wake-up process discussed in Section 3.2.4. As we see in Section 3.2.8, STEM-H does no worse than STEM, in energy consumption, and in most environments does significantly better. This is true even with substantial degradation from false positive detection on the wake-up channel. Intuitively, even if every carrier sensing period results in a false positive for STEM-H, monitoring nodes will stay on, in this worst case, about as long as STEM's idle listening period.

3.2.6 Proposed Protocol Description: STEM-BT2 (Figure 3.5)

From Section 3.2.4, we see that STEM-BT's energy consumption can be improved by reducing the wake-up cost while retaining its relatively low monitoring cost. Thus, we propose STEM-BT2 to augment STEM-BT's wake-up protocol with data channel probing for improved energy efficiency. The basic idea of STEM-BT2 is to perform the same wake-up protocol as STEM-BT while avoiding excessive idle listening on the data channel while waiting for the FILTER packet to be sent. Rather than turning on the data radio and doing continuous idle listening like STEM-BT, STEM-BT2 will periodically carrier sense the data channel to detect whether it is busy or not. When the data channel is detected busy, then STEM-BT2 remains on to receive the FILTER packet like STEM-BT.

Sending Protocol: The sending protocol is nearly identical to that of STEM-BT described in Section 3.2.3. The only difference is that two FILTER packets are sent on the data channel rather than one. The first FILTER packet is a "dummy" packet that allows probing nodes to detect the channel as busy and the second packet is the one that actually gets decoded.

Monitoring Protocol: The monitoring cycle is the same as STEM-BT. The difference in STEM-BT and STEM-BT2 is the reaction after a monitoring node detects a wake-up channel busy tone. STEM-BT's protocol is described in Section 3.2.3. STEM-BT2 reacts by turning on its data radio and carrier sensing for T_{wi} time.[6] If the data channel is detected as busy, the data radio remains on in anticipation of receiving a FILTER packet. If the data channel is detected as idle, the data radio returns to sleep for T_{ws2} time before attempting to sense again. This cycle continues until either the channel is detected busy or a timeout occurs.

Once the FILTER packet has been received, nodes behave the same as in STEM-BT. The intended receiver, as specified by the FILTER packet, remains on while all other nodes return their data radios to sleep. The sender and receiver return their data radios to sleep when the data channel has been idle for T_{th} time.

Discussion: STEM-BT2 improves the energy consumption of STEM-BT by reducing the cost of the expensive wake-up process while maintaining the benefits from the relatively inexpensive steady-state monitoring process discussed in Section 3.2.4. As we see in Section 3.2.8, STEM-BT2 rarely does worse than STEM-BT, in terms of energy consumption, and in most environments does significantly better. This is true even with significant degradation due to false positives (i.e., carrier sensing the channel busy when it is idle) being

[6]More generally, the carrier sensing time for the data radio and wake-up radio can be T_{wi2} and T_{wi}, respectively, where $T_{wi2} \neq T_{wi}$.

detected on the data channel. Intuitively, in the worst case, where every carrier sensing period on the data radio detects the channel as busy, STEM-BT2 will behave identical to STEM-BT except that it sends two FILTER packets instead of one. Thus, in this case, STEM-BT2 uses extra energy to transmit the extra FILTER packet, but otherwise behaves the same as STEM-BT.

3.2.7 Parameter Values

In this section, we discuss the values needed for the T_{wi}, T_{wt}, T_{ws2} parameters described in Section 3.2.1. The values for STEM and STEM-BT were discussed previously in [2, 3], but we mention them here for completeness.

STEM: The idle listening period, T_{wi}, must be long enough that a node will successfully receive a FILTER packet even if the node wakes up in the middle of a FILTER transmission (and hence cannot correctly decode the first FILTER packet). In the worst case, the node wakes up just after a FILTER transmission has begun. Thus, the node has to wait for this first, undecodable FILTER packet to finish, which takes about T_F time. It then must wait for the sender to idly listen for a FILTER-ACK, which takes αT_A time, where α accounts for propagation delays and hardware switching time. Finally, it must stay on long enough to receive the *next* FILTER packet, which takes T_F time. Thus, $T_{wi} = T_F + \alpha T_A + T_F = 2T_F + \alpha T_A$.

Now, we discuss T_{wt}, the duration for which the sending protocol must be performed to ensure enough overlap that every neighbor doing the monitoring protocol will receive a FILTER and have time to respond with the FILTER-ACK, if necessary. In the worst case, a monitoring node's idle listening period ends just before the sender's first FILTER transmission ends, which takes T_F time. The sender must continue the process for T_{ws} time since the monitoring node will be asleep for that duration. When the monitoring node begins idly listening again, it may wake up just after a FILTER transmission began (T_F time). After the sending node idly listens for αT_A time, it sends another FILTER packet (T_F time) which will successfully be decoded by the receiver. Finally, the sending node must wait long enough to receive the corresponding FILTER-ACK, if necessary, which takes αT_A time. Thus, summing up the terms mentioned in this paragraph, we get: $T_{wt} = T_F + T_{ws} + T_F + \alpha T_A + T_F + \alpha T_A = 3T_F + T_{ws} + 2\alpha T_A$.

STEM-BT: In STEM-BT, T_{wi} is a fixed value based on how long the radio must listen to detect a busy tone with a specified level of confidence (see [56] for a discussion on this). The busy tone transmission time, T_{wt}, must be sufficiently long to ensure enough overlap such that every neighbor doing the monitoring protocol receives the busy tone. In the worst case, a monitoring node's idle listening period begins just before the sender starts transmitting the busy tone. In this situation, the busy tone is not detected at the specified level of confidence. Thus, the sending node must transmit the busy tone long enough that the monitoring node's *next* idle listening period will completely overlap with the busy tone. Thus, $T_{wt} = T_{wi} + T_{ws} + T_{wi} = 2T_{wi} + T_{ws}$.

STEM-H: We begin by determining T_{ws2} and w_i, the sleeping time between idle listening periods and the number of idle listening period required to guarantee a FILTER is detected, respectively. To determine the frequency and number of times a nodes must wake-up during the monitoring periods, we observe the following constraint. If a node begins its idle listening period after the sender has started its FILTER packet transmissions, T_{ws2} and w_i must be chosen to guarantee at least one of the T_{wi} duration wake-ups will completely overlap with one of the sender's FILTER transmissions. Similar to STEM-BT, we assume that T_{wi} is the minimum amount of time required to classify the wake-up channel as busy (with sufficiently low error probability). Thus, if a FILTER packet only partially overlaps with a T_{wi} period, the channel may not be detected as busy.

In the worst case, a wake-up idle listening period begins just before a FILTER transmission begins. For example, the listening period begins at time t_0 and the FILTER transmission begins at $t_1 = t_0 + \epsilon$, where ϵ is a small positive number close to zero. In this case, $t_0 + T_{wi} < t_1 + T_{wi}$, which means that the FILTER transmission will not be detected for T_{wi}, the minimum required time for correct detection. Thus, T_{ws2} needs to be chosen such that the next idle listening period will begin and end before the current FILTER transmission ends. The FILTER transmission will end at $t_1 + T_F$. Thus, the next idle listening period needs to begin by $t_1 + T_F - T_{wi} = t_0 + \epsilon + T_F - T_{wi}$ to allow the minimum detection time. The first idle listening period ended at $t_0 + T_{wi}$. Thus, subtracting the first idle listening period's end time from the second idle listening period's start time, we get: $(t_0 + \epsilon + T_F - T_{wi}) - (t_0 + T_{wi}) = T_F - 2T_{wi} + \epsilon$. Thus, we need: $T_{ws2} \leq T_F - 2T_{wi} + \epsilon$. Because $\epsilon \rightarrow 0$ and the T_{ws2} inequality must be valid for the smallest ϵ possible, we get:[7] $T_{ws2} \leq T_F - 2T_{wi}$ to ensure that the second idle listening period completely overlaps with part of the FILTER packet transmission. To avoid unnecessary wake-ups, we set:[8]

$$T_{ws2} = T_F - 2T_{wi} \tag{3.1}$$

Next, we consider w_i, the number of times these idle listening periods must occur on the wake-up radio to ensure that one overlaps with an adequate part of a FILTER packet transmission. This is necessary since the

[7]It is assumed that $T_F > 2T_{wi}$. If this is not true, then it is impossible to guarantee sufficient overlap between the FILTER and carrier sensing periods without synchronizing the boundaries of T_F and T_{wi}.

[8]If false negatives are a problem with detecting the wake-up channel busy, T_{ws2} and w_i could be adjusted to provide redundancy in the amount of times idle listening periods are guaranteed to overlap with FILTER packet transmission. The obvious tradeoff is that more energy is consumed during the monitoring phase as T_{ws2} becomes smaller and w_i becomes larger.

idle listening periods may occur during the αT_A time that the sender is idly listening for a FILTER-ACK. We assume that T_{ws2} is set according to Equation 3.1.

In the worst case, the first wake-up idle listening period ends just after a FILTER transmission ends. For example, the FILTER packet transmission ends at t_1 and the first idle listening period ends at $t_2 = t_1 + \epsilon$. Thus, that idle listening period began at $t_0 = t_2 - T_{wi}$. In this case, $t_0 + T_{wi} > t_1$, which means that the FILTER transmission will not be detected for T_{wi}, the minimum time required for correct detection. After this most recent FILTER transmission, the sender will wait for αT_A time before beginning the next FILTER transmission (i.e., it begins at $t_1 + \alpha T_A$). We need to guarantee that enough idle listening periods with the T_{ws2} spacing will occur such that the last one begins at or after $t_1 + \alpha T_A$ (and hence is detects the next FILTER transmission). The next (second) idle listening period begins at $t_2 + T_{ws2}$. If another one occurs (the third), it will begin at $t_2 + T_{ws2} + T_{wi} + T_{ws2}$. If we have w_i such idle listening periods, the last one will begin at $t_2 + (w_i - 1)T_{ws2} + (w_i - 2)T_{wi}$ (trivially, $w_i \geq 2$). Using Equation 3.1, the last idle listening period begins at: $t_2 + (w_i - 1)(T_F - 2T_{wi}) + (w_i - 2)T_{wi} = t_2 + (w_i - 1)T_F - w_i T_{wi}$. Thus, we need:

$$\begin{aligned} t_1 + \alpha T_A &\leq t_2 + (w_i - 1)T_F - w_i T_{wi} \\ t_1 + \alpha T_A &\leq t_1 + \epsilon + (w_i - 1)T_F - w_i T_{wi} \\ \alpha T_A &\leq \epsilon + (w_i - 1)T_F - w_i T_{wi} \end{aligned} \tag{3.2}$$

Because $\epsilon \rightarrow 0$ and the inequality in Equation 3.2 must be valid for the smallest ϵ possible, we get:

$$\alpha T_A \leq (w_i - 1)T_F - w_i T_{wi} \tag{3.3}$$

Therefore, we need w_i to be the smallest integer which satisfies the inequality in Equation 3.3. This gives us:

$$w_i = \left\lceil \frac{\alpha T_A + T_F}{T_F - T_{wi}} \right\rceil \tag{3.4}$$

To determine T_{wt}, we consider the worst case where the first FILTER transmission starts just after the last idle listening period begins for a monitoring node (i.e., just before the last idle period of length T_{wi} returns to sleep for T_{ws} time). In this case, the sender has to do the wake-up procedure for the length of that idle listening period (i.e., T_{wi}) plus the subsequent sleeping time (i.e., T_{ws}). Additionally, it must transmit for the time it takes the monitoring node to do w_i idle listening periods (i.e., $w_i T_{wi} + (w_i - 1)T_{ws2}$). Recall that T_{ws2} and w_i were set in Equation 3.1 and Equation 3.4, respectively, to ensure that the channel is carrier sensed busy during one of these w_i idle listening periods. Thus, in the worst case, the beginning of the FILTER transmission is detected during the monitoring node's last idle listening period. Thus, the monitoring node will keep its wake-up radio on to receive the *next* FILTER. The next FILTER transmission occurs *after* the sender idly listens for a FILTER-ACK. In this case, the time needed is enough to detect (but not correctly receive) the first FILTER packet, then wait for the transmitter to listen for the FILTER-ACK, and then for the monitoring node to *receive* the next FILTER packet send by the transmitter (i.e., $2T_F + \alpha T_A$). Finally, the sender must continue the wake-up process long enough to receive the FILTER-ACK which follows the last FILTER transmission (i.e., αT_A). Combining all this time, we get:

$$\begin{aligned} T_{wt} &= T_{wi} + T_{ws} + w_i T_{wi} + (w_i - 1)T_{ws2} + 2\alpha T_A + 2T_F \\ &= (w_i + 1)T_{wi} + T_{ws} + (w_i - 1)T_{ws2} + 2\alpha T_A + 2T_F \end{aligned} \tag{3.5}$$

STEM-BT2: The value of T_{wt} in STEM-BT2 is computed exactly the same as for STEM-BT, as described previously. We set $T_{ws2} = T_F - 2T_{wi}$ for the same reasons discussed for STEM-H.

3.2.8 Simulation Results

To test the protocols from Section 3.2.1, we implemented them by modifying the 802.11 MAC and physical layer code in *ns-2* [103]. We use the values from Table 3.3. These values are based on Mica2 Motes [104] and TinyOS [11]. For STEM-BT, STEM-H, and STEM-BT2, we set $T_{wi} = 1\,\text{ms}$ [56]. This is the time it takes to reliably detect that the wake-up and data channel are busy when a busy tone or packet is being transmitted.[9] Each data point is averaged over 20 runs. The standard deviation for the figures is given in Table 3.1 and Table 3.2.

Table 3.1: Standard deviation as percentage of mean for Section 3.2.8 figures (Average | Maximum).

	Fig. 3.6		**Fig. 3.7**		**Fig. 3.9**		**Fig. 3.12**		**Fig. 3.13**		**Fig. 3.14**[a]	
STEM	3.54	3.99	2.18	2.90	2.096	2.628	1.00	1.00	1.60	2.58	1.93	3.18
STEM-BT	1.90	2.61	1.08	1.50	1.96	2.76	0.44	1.43	20.10	24.57	0.53	0.80
STEM-H	2.27	2.86	1.61	2.57	1.84	2.45	1.59	15.87	18.63	22.04	1.92	3.13
STEM-BT2	1.84	2.41	1.13	1.52	1.64	2.28	0.37	0.84	21.34	24.14	0.83	1.36

[a] In Figure 3.14, STEM-BT, STEM-H, and STEM-BT2 all have 50% false positive probabilities.

Initially, we look at a single-hop scenario with 10 sensors in range of each other. A random sender and receiver are chosen to communicate with Poisson traffic at a fixed rate of one packet per second (unless otherwise noted). Each simulation runs for 500 seconds. Because each data packet has a corresponding ACK

[9]While we do not explicitly consider switching energy and delay, T_{wi} can be adjusted to this cost into account. We note that radios on earlier versions of Mica Motes had transition times of less than $10\,\mu$s [105].

Table 3.2: Standard deviation as percentage of mean for Section 3.2.8 figures with false positive curves (Average | Maximum).

	Figure 3.10		Figure 3.11	
STEM	3.66	4.04	—	—
STEM-H, 0%	2.17	2.95	—	—
STEM-H, 1%	2.31	3.17	—	—
STEM-H, 5%	2.77	3.71	—	—
STEM-H, 50%	3.87	4.98	—	—
STEM-H, 100%	3.71	4.72	—	—
STEM-BT, 0%	—	—	0.31	1.10
STEM-BT2, 0%	—	—	0.09	0.16
STEM-BT, 5%	—	—	0.79	1.09
STEM-BT2, 5%	—	—	0.91	1.29
STEM-BT, 50%	—	—	0.40	0.69
STEM-BT2, 50%	—	—	0.42	0.58
STEM-BT, 95%	—	—	0.44	0.87
STEM-BT2, 95%	—	—	0.47	1.12

Table 3.3: Protocol parameter values.

Parameter	Value
Physical Layer Header	28 bytes
MAC Layer Header	6 bytes
Payload per Packet	30 bytes
Total Packet Size[a]	64 bytes
Bitrate	19.2 kbps
P_{TX}	81 mW
P_I	30 mW
P_S	3 μW
T_{th}	30 ms

[a] We assume that FILTER, FILTER-ACK, data, and ACK packets are the same size.

packet, 128 bytes (see Table 3.3) are transferred per data packet. This translates to a 5.33% utilization of the channel bitrate. The sleep interval, T_{ws}, for the protocols is varied to determine its effects on energy and latency. The goal of these experiments is to investigate properties of the protocols in a simple environment.

We also evaluate performance in a more realistic, multi-hop scenario. We define the node density of a network, Δ, as $\Delta = \frac{N \cdot \pi r^2}{A}$, where N is the number of nodes in the network, r is the range of a node's radios, and A is the geographic area of the network. Given this definition, we place 50 sensor nodes uniformly at random in a square region such that $\Delta \approx 9.8$. For each topology tested, a path exists between every node in the network. We vary the number of connections per scenario while keeping T_{ws} and the Poisson traffic

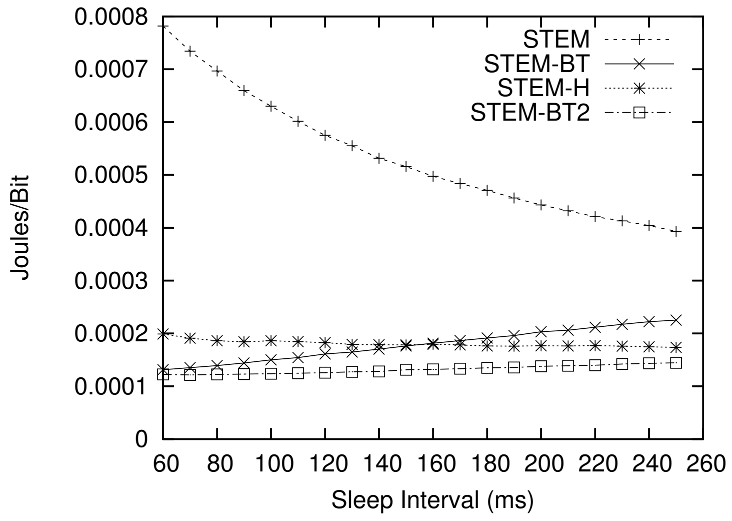

Figure 3.6: Energy consumption of the protocols.

rate fixed. For each connection the sender and receiver of the flow is chosen uniformly at random. For the sensor network application at Great Duck Island [106], the traffic rate is one data packet every 70 seconds. To keep our simulation execution times reasonable, the traffic rate per flow is set to be one data packet every 20 seconds and each simulation runs for 1000 seconds.

Energy and Latency Comparison: We adjust T_{ws} from 60 ms to 250 ms to see the performance of the protocols, shown in Figure 3.6. The energy metric we use is Joules per bit, which is the aggregate energy used by all nodes during the simulation divided by the total number of data bits received by the destination(s).

The simulation results are presented in Figure 3.6. We see that STEM uses the most energy of the protocols, but shows a large improvement as T_{ws} increases. Recall that STEM's major weakness is its large energy cost to monitor the wake-up channel in steady-state. As T_{ws} increases, the relative amount of sleep time for the monitoring process increases. Thus, the monitoring process uses less energy while the wake-up process uses only slightly more energy (due to the increase in T_{wt} for the sender). STEM-H consistently does much better than STEM. STEM-H's energy consumption also decreases as T_{ws} increases, though it is much less dramatic than STEM's decrease.

Figure 3.6 also shows that STEM-BT2 consistently outperforms STEM-BT. Both STEM-BT and STEM-BT2 show the same trend, a linear increase in energy consumption as T_{ws} increases. This is because the wake-up cost for these protocols increases while the monitoring cost decreases only slightly. In particular, the busy tone is transmitted for a longer time and, hence, neighbors have to keep their data radios idly listening (or probing) for a longer period of time on average.

In Figure 3.7, we see a linear increase in latency for all protocols as T_{ws} increases. This is because the wake-up process takes longer as T_{ws} grows. STEM and STEM-H show a more gradual increase in latency because the time of their wake-up processes are proportional to roughly $\frac{1}{2}T_{ws}$, whereas STEM-BT and STEM-BT2's wake-up processes are proportional to T_{ws}. We also note that STEM-H and STEM-BT2 have a latency that is larger than that of STEM and STEM-BT, respectively, by a constant amount. This constant amount is approximately equal to T_F since STEM-H has to wait for an extra FILTER to be sent on the wake-up channel (when compared with STEM) and STEM-BT2 has to wait for an extra FILTER to be sent on the data channel (when compared with STEM-BT).

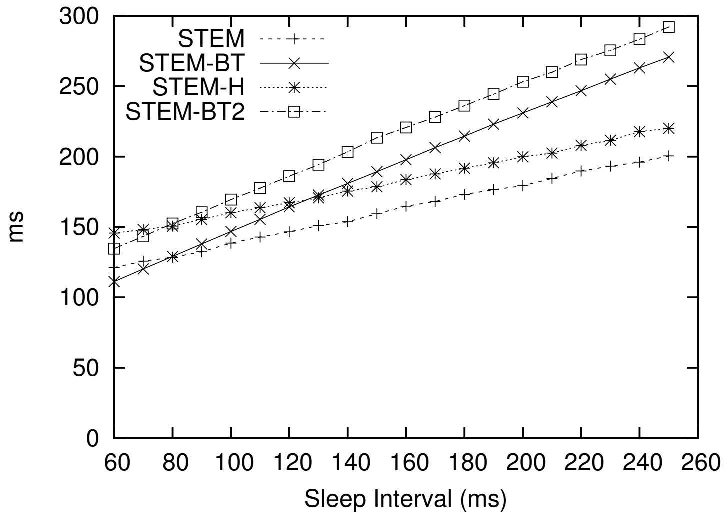

Figure 3.7: Latency of the protocols.

To gain a better understanding of the energy-latency tradeoff for the protocols, we plot the average latency versus the energy consumption for each fixed T_{ws} value. The result is shown in Figure 3.8. We can see that STEM-BT2 outperforms all of the other protocols in this metric. However, we notice that STEM-BT and STEM-BT2 show an increase in energy consumption as latency increases while STEM and STEM-H show a decrease in energy consumption. Thus, comparing the energy of STEM-H to that of STEM-BT when the average latency is about 220 ms is misleading because STEM-H has lower energy consumption at that point, but STEM-BT has lower energy consumption when the average latency is smaller.

In Figure 3.9, we test the protocols at a higher sending rate. The rate is set to three packets per second (i.e., 16% channel utilization). From this graph, we see that the relative difference in energy consumption between STEM and STEM-H decreases. This is because a higher rate reduces the amount of monitoring time between wake-up procedures. Thus, STEM does better at higher data rates due to less monitoring time per packet arrival. STEM-H, however, shows less relative improvement since its monitoring cost is already

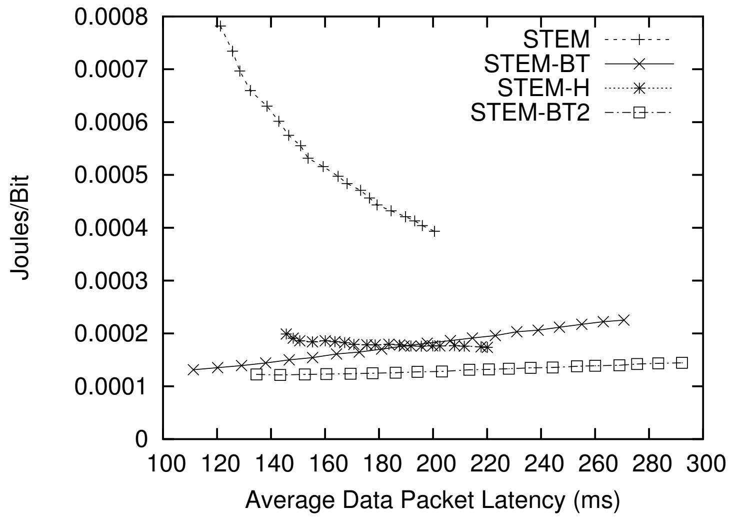

Figure 3.8: Energy versus average latency.

low by design.

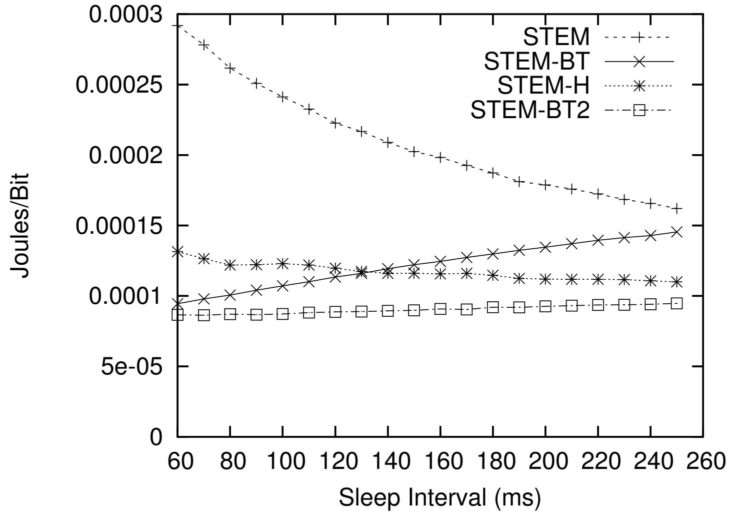

Figure 3.9: Energy consumption at a higher rate.

Also in Figure 3.9, we see that the relative difference between STEM-BT and STEM-BT2 increases at a higher rate. This is because, at a higher rate, the wake-up procedure becomes more frequent. Thus, STEM-BT2, which has a lower wake-up cost than STEM-BT, will further outperform STEM-BT.

The Effects of Spurious Wake-Ups: One of the disadvantages of doing the FILTER transmission detection on the wake-up channel in STEM-H is that interference in the frequency band may cause a node to detect the channel as busy when no FILTER is being transmitted. For example, such interference may come from other sensor nodes that are not within communication range, but still transmit with enough power to interfere. Another example of interference is other electronic devices that share the same unlicensed bandwidth as the sensors. Thus, we study how the performance of STEM-H degrades in the face of such interference.

Figure 3.10 shows the energy consumption of STEM and STEM-H. For STEM-H, we vary the probability that when a monitoring node idly listens on the wake-up channel, it erroneously detects a FILTER transmission (i.e., a false positive). For STEM-H, the percentage values shown in the key of Figure 3.10 indicate the probability a false positive occurs each carrier sensing period. For example, "STEM-H, 5%" indicates that each idle listening period on the wake-up channel a monitoring node falsely detects a FILTER transmission with probability 0.05. Thus, 0% false postive value indicates that *every* detection of the wake-up channel as busy is caused by a FILTER transmission. A 100% false positive value is the worst case where every carrier sensing period a monitoring node detects the wake-up channel as busy regardless of its actual state.

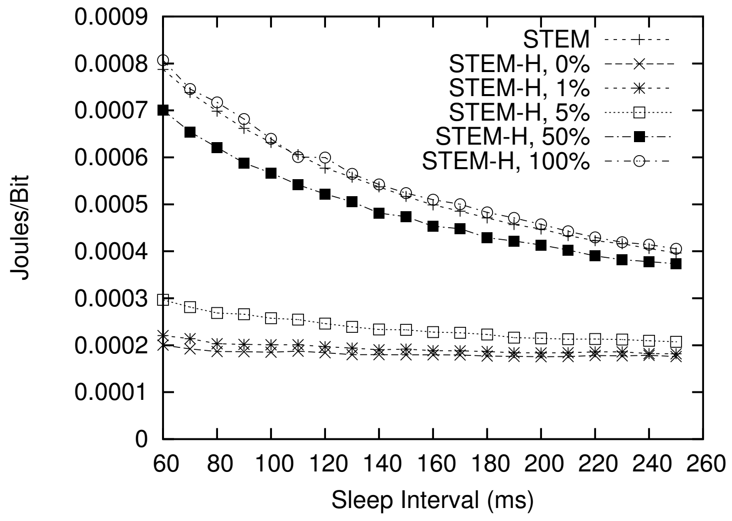

Figure 3.10: Effects of false positives on STEM-H.

From Figure 3.10, we see that a low false positive percentage (e.g., less than 5%) does not affect the performance of STEM-H much and it still significantly outperforms STEM. As the false positive percentage increase, the performance of STEM-H converges to that of STEM (the line for STEM and STEM-H with 100% false positives almost overlaps). This confirms the intuition discussed in Section 3.2.5. Thus, with completely unreliable FILTER transmission detection, STEM-H does no worse than STEM.

We do similar tests with STEM-BT and STEM-BT2. In STEM-BT, false positives affect the detection of busy tones on the wake-up channel. In STEM-BT2, false positives affect both the detection of busy tones on the wake-up channel *and* the detection of FILTER packets on the data channel. Figure 3.11 shows the results. At a low false positive percentage (i.e., 0% and 5%), STEM-BT2 saves more energy than STEM-BT because the benefits from data channel carrier sensing are significant. However, as the false positive percentage becomes large (i.e., 50% and 95%), the performances of STEM-BT and STEM-BT2 converge since both protocols are using a large amount of energy on the wake-up channel. In this situation, STEM-BT2's data channel carrier sensing will exhibit similar behavior to STEM-BT's data channel idle listening since the false positives from carrier sensing cause STEM-BT2 to frequently resort to idle listening.

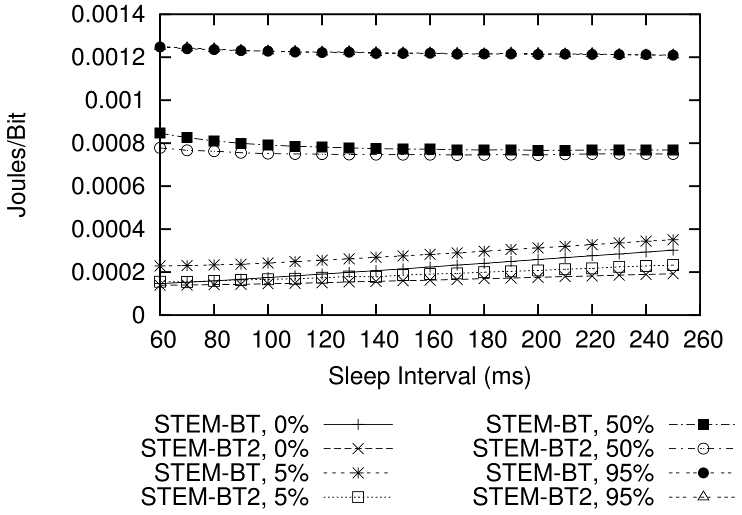

Figure 3.11: Effects of false positives on STEM-BT and STEM-BT2.

Our final experiment to test spurious wake-ups investigates the relation between the false positive percentage and energy consumption. In Figure 3.12, we fix $T_{ws} = 100\,\text{ms}$ and see that only at low percentages does STEM-H perform worse that STEM-BT. This is because carrier sensing occurs more frequently for STEM-BT than for STEM-H and, thus, shows more degradation as a higher percentage of carrier sensing periods detect false positives. STEM-BT2 also outperforms STEM-BT at low percentages, and their performance converges at high percentages. STEM is not affected by spurious wake-ups; its performance is shown for reference relative to the other protocols.

Multi-Hop Performance: We tested the protocols in multi-hop, multi-flow environments. We note that this is the first time, of which we are aware, that such tests have been done on STEM or STEM-BT. In [2,3], the simulation results are limited in that they consider only a single flow.

In Figure 3.13, we set $T_{ws} = 100\,\text{ms}$ and incrementally increase the number of concurrent flows in the network. At the low rate of one packet every 20 seconds, STEM does much worse than the other protocols for reasons discussed in Section 3.2.4. The performance of STEM-BT and STEM-BT2 is almost identical

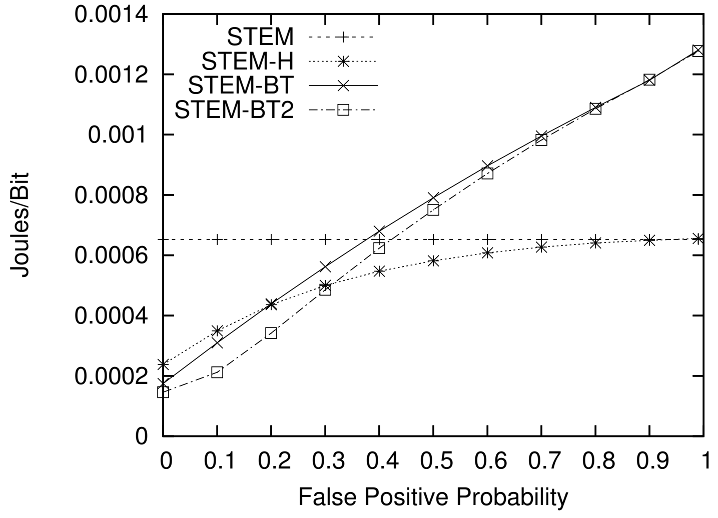

Figure 3.12: Effects of false positives on protocols.

since wake-ups are rare. STEM-H does slightly worse than STEM-BT and STEM-BT2 because, as shown in Figure 3.4, STEM-H has a higher monitoring cost since it must wake up multiple times in between sleep intervals.

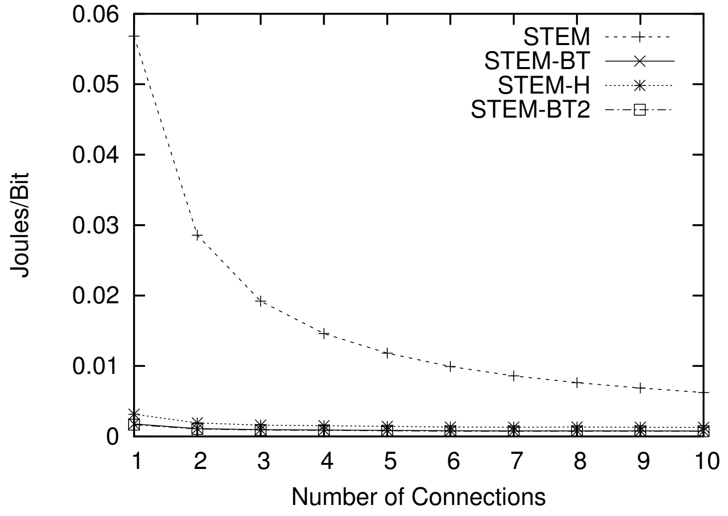

Figure 3.13: Multi-hop energy consumption.

From Figure 3.13, one could conclude that the busy tone protocols are always superior to STEM and STEM-H. However, STEM-BT and STEM-BT2 rely heavily on carrier sensing and are, therefore, more susceptible to the effects of spurious wake-ups. In contrast, STEM is unaffected by false positives since no carrier sensing is needed to determine when the radios should remain on. STEM-H shows some effects from spurious wake-ups, but as shown in Figure 3.12, large false positive percentages are not as detrimental as they are for STEM-BT and STEM-BT2. Thus, in Figure 3.14, we see that STEM and STEM-H outperform STEM-BT and STEM-BT2 when the sensors operate in an environment where spurious wake-ups are a significant problem.

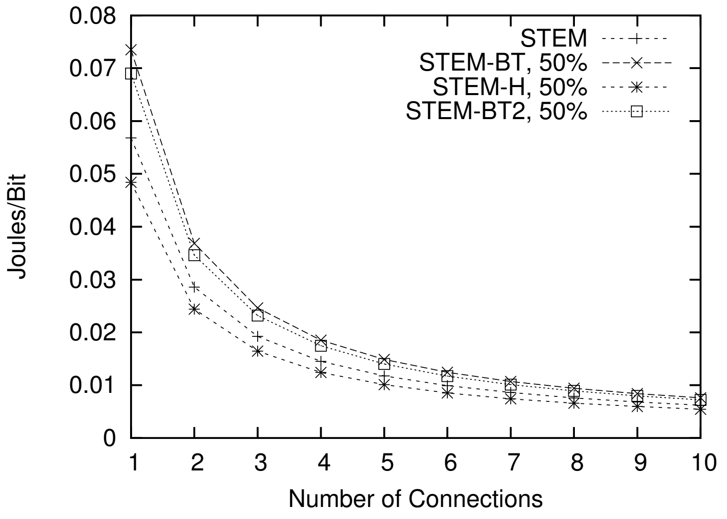

Figure 3.14: Multi-hop energy consumption with false positives.

3.3 Summary

In this chapter, we have proposed and demonstrated two methods where carrier sensing at the physical layer can reduce the energy consumption of power save protocols. The technique significantly reduces the energy spent listening for wake-up signals. We have shown how this can be applied to both an in-band protocol, IEEE 802.11 PSM [7], (Section 3.1.1) and an out-of-band protocol, STEM [2, 3], (Section 3.2.1). This technique is particularly beneficial when traffic is light and, most of the time, no packets need to be advertised.

Chapter 4

Adaptive Energy-Saving Protocols

Many power save protocols use a similar design where nodes sleep for some fixed interval and, at the end of that interval, wake up for a fixed listening interval to check for wake-up signals. This design is used in 802.11 PSM [7] as well as sensor network protocols, such as B-MAC [26] and STEM [2]. As discussed in Section 2.1.3, the disadvantage of such an approach is that is results in a "one size fits all" protocol that is agnostic of the traffic rate in the network or desired latency of an application, for example.

In this chapter we propose adaptive techniques to improve the energy efficiency of in-band power save protocols. Our previous work [27–29] has looked at adaptive intervals for out-of-band protocols. In Section 4.1, we propose dynamic listening intervals based on the traffic near a node. In Section 4.2, we explore dynamic sleeping intervals based on the desired latency requirements of an application.

4.1 Dynamic Advertisement Windows

From the description of 802.11 PSM in Section 2, we can see that the ATIM window wastes a significant amount of energy when the traffic load is low. For example, in previous work [60, 68] some typical values for the ATIM window and beacon interval are 20 ms and 100 ms, respectively. Thus, even when *no* traffic is being sent, nodes listen to the channel for 20% of the time. It is obvious that more energy could be conserved by reducing the size of the ATIM window when traffic is sparse. However, if the ATIM window becomes too small, then nodes cannot advertise their data since the window ends before they are able to access the channel and send an ATIM. Thus, our techniques reduce the overhead of the ATIM window when traffic is sparse and provide larger ATIM windows when more data needs to be advertise.

The major contribution of this work is that we dynamically re-size the ATIM window based on the number of advertisements to be sent in the current window. While the dynamic adjustment of the ATIM window has been explored previously [60, 107], this is the first work of which we are aware that achieves this in a multi-hop environment using a single channel.

4.1.1 Protocol Description

The CS-ATIM protocol discussed in Chapter 3 is more energy efficient than 802.11 PSM when there are a large number of beacon intervals in which no nodes have packets to advertise. However, if a small number of packets need to be advertised in a beacon interval, then requiring nodes to listen for the entire ATIM window wastes energy. Ideally, the ATIM window should be long enough for all the ATIMs that need to be transmitted and then the ATIM window should end right after the last ATIM-ACK is received.[1] This is what past work tries to achieve either through heuristics [60] or dynamically extending the window when packets are received [61, 107]. Unlike the previous work that dynamically extends the ATIM window based on packet reception, our goal is to have a protocol that works in multi-hop environments and does not use a second channel (e.g., a busy-tone channel). We refer to this extension of CS-ATIM as *Dynamic* CS-ATIM (DCS-ATIM).

First, we distinguish between two types of packet reception in IEEE 802.11. When a packet is received at a power level above the RX_THRESHOLD, we say that the receiver is within the *transmission range* of the sender. When a packet is received at a power level below the RX_THRESHOLD, but above the CS_THRESHOLD (carrier sense threshold), the receiver is said to be within the *carrier sensing range* of the sender. Packets received by nodes in the carrier sensing range cannot be decoded, but do cause the node's clear channel assessment to classify the channel as busy. We assume that, most of the time, a node's carrier sensing range is at least twice as big as its transmission range [108]. Thus, when S sends a packet and R is within the transmission range of S, the nodes within the transmission range of R are likely to be within the carrier sensing range of S.

We note that in several cases a node may receive a packet above the RX_THRESHOLD, but its neighbors do not receive the packet above the CS_THRESHOLD. This may occur due to short-term fading or obstructions in the line-of-sight of a node pair. While DCS-ATIM can recover from such occurrences, we assume such events are rare.[2] In the worst case, when a node detects little or no correlation between its packet receptions and a neighbor's carrier sensing of these packets, then the node can fall back to CS-ATIM to advertise packets to that neighbor.

In DCS-ATIM, *two* carrier sensing periods follow the beginning of the beacon interval:

[1]This statement assumes traffic is not so heavy that the ATIM window grows large enough that data packets can never be sent.

[2]We do not test these situations in our simulations.

- $\mathbf{CS}_1$: As in CS-ATIM, DCS-ATIM begins with a carrier sensing period of length T_{cs} during which time nodes use the protocol from Section 4.1.1 to indicate whether they have packets to advertise. We refer to this carrier sensing interval as $\mathbf{CS}_1$.

- $\mathbf{CS}_2$: DCS-ATIM adds a second carrier sensing period, $\mathbf{CS}_2$, (of duration T_{cs}) that immediately follows $\mathbf{CS}_1$. If a node wants its neighbors to use a static ATIM window, as in CS-ATIM, then it transmits a dummy packet during $\mathbf{CS}_2$. Otherwise, its neighbors use the dynamic window scheme described below. For example, a node may use a static ATIM window if it has not been able to advertise a packet for the past k intervals. This is a fail-safe mechanism when a packet is unable to be advertised after attempting for several dynamic windows.

We now describe the protocol after the above two carrier sensing periods when nodes have decided to use dynamic ATIM windows. First, we give ATIM packets a different maximum contention window size (CW_{aw}) than data packets (CW_{max}). In the IEEE 802.11 specification [7] for direct-sequence spread spectrum (DSSS), the default CW_{max} is 1023 slots and the default slot time, T_{slot}, is 20 μs. Using such a large contention window for ATIMs is unnecessary when the entire ATIM window is typically on the order of tens of milliseconds. Also, only one ATIM is sent per sender-receiver pair whereas multiple data packets may then be sent over that link after the ATIM window. Thus, the number of ATIM packets sent in the ATIM window should be less than or equal to the number of data packets sent following the ATIM window. This means there should be less nodes contending for access during the ATIM window since each sender-receiver link contends for the channel only once during the ATIM phase, but potentially multiple times during the data phase. Therefore, it is not unreasonable to make $CW_{aw} < CW_{max}$ in most scenarios. Thus, nodes that have ATIMs to send during the ATIM phase use the same protocol as 802.11 CSMA/CA, but use CW_{aw} as the maximum contention window size rather than the default CW_{max}.

At the start of the dynamic ATIM window, every node listens to the channel and sets a timer to expire after:

$$\begin{aligned} T_{idle} = {} & DIFS + T_{slot} \cdot CW_{aw} + prop_{max} \\ & + T_{atim} + SIFS + prop_{max} + T_{ack} \\ & + DIFS + T_{slot} \cdot CW_{aw} + prop_{max} \end{aligned} \tag{4.1}$$

where $DIFS$ and $SIFS$ are the DCF and Short Interframe Space as specified by IEEE 802.11 [7], respectively. The values T_{atim} and T_{ack} are the time durations required to send an ATIM and ATIM-ACK, respectively.[3]

[3]T_{atim} and T_{ack} are constant since ATIM and ATIM-ACK packets have a fixed, specified size.

The maximum propagation delay between two nodes is denoted as $prop_{max}$. T_{idle} is long enough to give a node the chance to access the channel after it was in the carrier sensing, but not transmission range, of an ATIM/ATIM-ACK handshake.

If a node sends or carrier senses a packet before the timer expires, the timer is reset to end T_{idle} time after the packet is sent or carrier sensed. To avoid starvation, an upper limit is set on the size that the dynamic ATIM window can reach. We set this upper bound equal to the default, static ATIM window size, T_{aw}, used for unmodified 802.11 PSM.

A node may transmit ATIM packets as long as it has sent a packet or received a packet above the RX_THRESHOLD within the past T_{idle} time. When one of these two conditions is met, it implies that the node's neighbors have either received or carrier sensed a packet within the past T_{idle} interval and, hence, refreshed their timers to continue listening for ATIM packets. If a node has carrier sensed a packet within the past T_{idle} time, but not sent or received a packet during that time, then it must continue to listen for ATIMs until its timer expires, but it cannot send anymore ATIMs until the next beacon interval. If a node is unable to send an ATIM for k consecutive intervals, it uses $\mathbf{CS}_1$ to let its neighbors know to resort to a static ATIM window size.

Whenever a node does not send or carrier sense a data packet for T_{idle} time, or the upper bound on the dynamic ATIM window is reached, the node ends the ATIM phase and waits for the data phase to begin. As in 802.11 PSM, if a node sent or received an ATIM during the ATIM window, it remains on for data communications. Otherwise, the node returns to sleep until the beginning of the next beacon interval. The data phase begins T_{aw} after the start of the ATIM window. It is postponed until this time to avoid sending potentially long data packets while other neighbors are trying to transmit ATIMs.

An example of DCS-ATIM compared with 802.11 PSM is given in Figure 4.1. First, DCS-ATIM has an additional carrier sensing period at the beginning of the beacon interval. Because **A** has a packet to advertise, it sends a dummy packet at the start of the beacon interval. In this example, **A** desires a dynamic ATIM window, so no dummy packet is sent during the second carrier sensing period. After both carrier sensing periods have ended, **A** sends an ATIM to **B**. In this example, **C** does not carrier sense anymore transmissions after **B**'s ATIM-ACK. Thus, with DCS-ATIM, **C** returns to sleep T_{idle} time after receiving the ATIM-ACK rather than waiting for the entire T_{aw} duration of the ATIM window. With 802.11 PSM, **C** must remain on for the entire T_{aw} time of the static ATIM window.

From this description, we see that, in the worst case, the ATIM window for DCS-ATIM uses only slightly more energy in the ATIM window than 802.11 PSM (for the carrier sensing periods) and may use much less energy when a small number of ATIMs are sent. In terms of latency, DCS-ATIM may perform worse than

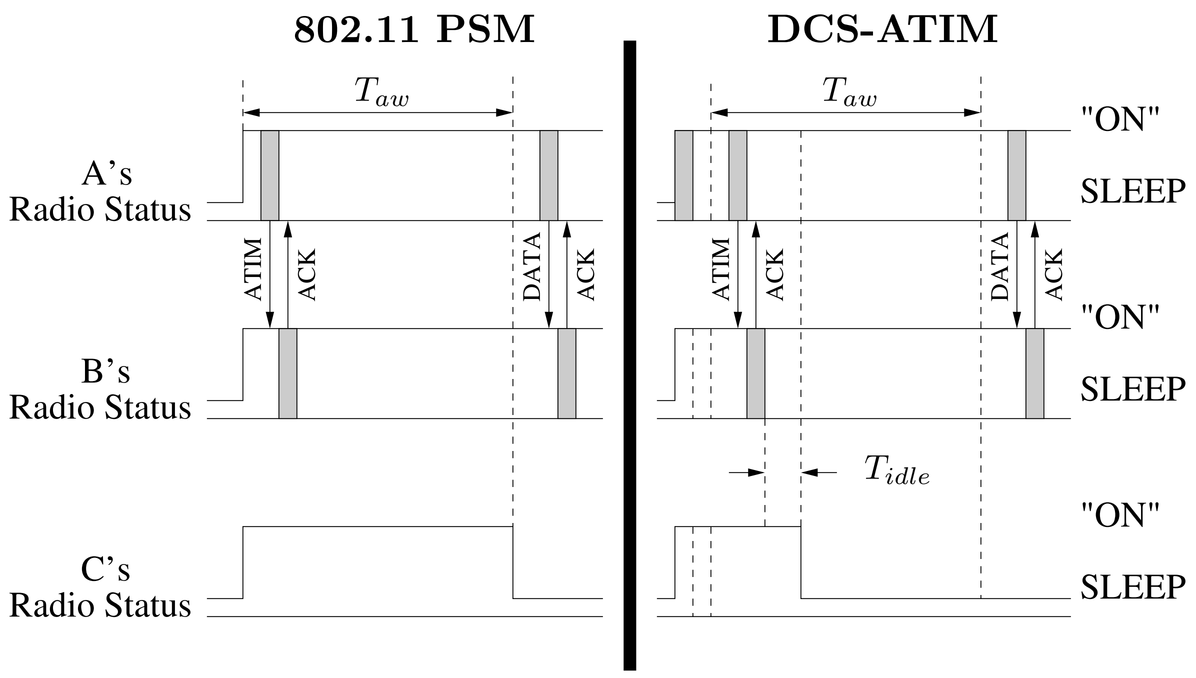

Figure 4.1: 802.11 PSM vs. DCS-ATIM.

802.11 PSM if a data packet arrives at the node towards the end of 802.11 PSM's static ATIM window. In this case, 802.11 PSM can advertise the packet and send the data in the current beacon interval. By contrast, if DCS-ATIM's dynamic ATIM window has already ended, the node may wait until the next beacon interval to advertise the packet. Additionally, DCS-ATIM may not be able to advertise as many packets as 802.11 PSM if a node with a packet to advertise does not send or receive any packets above the RX_THRESHOLD as discussed above (e.g., a node cannot transmit since it lost contention to the access). In this case, the node will wait until the next ATIM window to advertise the packet.

4.1.2 Design Discussion

T_{idle} **Length:** As shown in Equation 4.1, we set T_{idle} (the time a node waits before returning to sleep) to be a static value that is long enough for a sender to lose channel access once and still have its receiver remain listening to the channel. We chose this value to design for systems where traffic, and, hence, contention, is rather low. We feel that many power save protocols may operate in such environments where traffic is sparse, such as sensor networks. In Section 4.1.3, we show that this value works relatively well in the environments we tested.

If traffic is extremely sparse and usually only one node is transmitting in an area, then using an even smaller value of T_{idle} may perform better due to the lack of contention. However, if the environment is such that contention increases, then it may be better to choose a larger value of T_{idle}. Such a situation may occur in event detecting applications, where a geographically clustered group of sensors may begin contending for the channel at the same time in response to an event.

The value of T_{idle} should be chosen appropriately for the expected traffic patterns of the application. If the amount of traffic has a high deviation (e.g., event detection), it may be beneficial to explore dynamic values of T_{idle}. In this scenario, a sender may track a contention metric, such as how often it is unable to transmit its packet within T_{idle} time or how many other nodes in its neighborhood are contending. If the contention metric increases, then a sender can piggyback a new T_{idle} value on advertisement packets. When nodes receive new T_{idle} values from their neighbors, they will choose the largest value as the time that they use to accommodate their neighbor that expects them to remain on the longest.

In the dynamic scenario, a mechanism must be introduced to allow nodes to reduce their T_{idle} value when the contention decreases. A node could keep track of what T_{idle} value was requested by each neighbor and when. Then a node would no longer consider that value in deciding which value to use if (1) a soft timer expires without hearing from that particular neighbor or (2) the neighbor explicitly sends a new, lower T_{idle} value, which is then used in future decisions of how long to extend listening. Thus, the T_{idle} value associated with each neighbor may be changed implicitly or explicitly. Whenever a neighbor's T_{idle} value expires or changes, the node will again look at its neighbor table to determine which one has the largest T_{idle} value and, hence, should be used by the node.

The dynamic adjustment could be further optimized by a sender specifying T_{idle} times per receiver rather than using one valued for every neighbor. The main reason that we do not use such a dynamic scheme is that it significantly increases the amount of coordination and complexity of the protocol. However, future work could include incorporating such a modification to determine how much traffic variance is necessary before such a scheme offers significant benefits over our scheme.

Dynamic Thresholds: In our current protocol description, we use RX_THRESHOLD and CS_THRESHOLD as the signal strengths necessary for a node to consider its neighbors as still listening and to continue listening to the channel, respectively. We use these values because they are already specified for other purposes. However, it is not necessary to use these two values. More generally, we could refer the thresholds as Thr_{tx} and Thr_{listen} and adjust them as necessary. That is, Thr_{tx} and Thr_{listen} are not necessarily equal to RX_THRESHOLD and CS_THRESHOLD, respectively.

These thresholds could then be adjusted dynamically in response to the environment. By *increasing* Thr_{tx}, a node becomes more conservative in when it considers its neighbors to still be listening and, hence, transmits. By *decreasing* Thr_{listen}, a node becomes more liberal in when it continues listening to the channel and, hence, may receive more of a transmitter's packets, but expends more energy.

If a sender fails to successfully communicate with its receiver using DCS-ATIM and has to fall back to the original 802.11 PSM protocol, as discussed in Section 4.1.1, then it can increase its Thr_{tx} by some specified amount. Each sender could also maintain a Thr_{tx} *per receiver* based on past history. If the sender still has difficulty in communicating with the receiver, it could piggyback some information (e.g., the number of beacon intervals the packet has been delayed) in each packet in order to give the receiver an indication that it should decrease its Thr_{listen}. Unlike the Thr_{tx} value, a node can only use one Thr_{listen} value (as opposed to a separate value per sender) since the receiver does not know which senders may try to transmit in a given beacon interval.

One major design issue is when to decrease and increase Thr_{tx} and Thr_{listen}, respectively, in a dynamic scheme. The benefit of decreasing Thr_{tx} is that the sender may be too conservative in trying to communicate and unnecessarily delay packets for subsequent beacon intervals when the receiver is in fact still listening in the current advertisement window. The benefit to increasing Thr_{listen} is that the node will use less energy remaining on to listen for potential advertisements.

The difficulty with these rules are as follows. The reasons that a sender/receiver pair would try to decrease Thr_{tx} or increase Thr_{listen} are (1) the link and/or connection terminates or (2) their current communication is acceptable in terms of reliability and they want to try to improve the latency or energy consumption by changing Thr_{tx} and Thr_{listen}, respectively. In the first case, a node must keep track of the thresholds *per connection/link* and use, for example, soft timers or packet loss to determine when the connection or link terminates. At this point, the sender can stop using that receiver's Thr_{tx} value and the receiver can re-evaluate the Thr_{listen} value it is using after removing the Thr_{listen} value for that sender.

The second case is more difficult since it is necessary to know how much a modification in Thr_{tx} or Thr_{listen} would effect the reliability. One method may be for the nodes to periodically modify their thresholds for a neighbor and observe the reliability. Determining the magnitude of the change may be another issue. Also, it may be the case that, for whatever reason, the observations represent outliers when compared with the common operating environment and, thus, measure better reliability than will be experienced on averaged with the thresholds being tested.

We chose to use the RX_THRESHOLD and CS_THRESHOLD values since they are already specified. Also, choosing a specific Thr_{tx} and Thr_{listen} would vary largely by environment and, thus, is difficult to effectively test in simulation. We chose to use static values of the thresholds to simplify the protocol. As we can see from the discussion in this section, adding dynamic extensions would greatly complicate the protocol. As we show in Section 4.1.3, just using the static value for Thr_{tx} and Thr_{listen} performs well in many environments and the added complexity of dynamic values may not be worth the potential additional improvement in many circumstances.

4.1.3 Simulation Results

To evaluate our protocols, we simulated them by modifying the MAC and physical layers of *ns-2* [103]. We use the notation P_{tx}, P_{rx}, P_{listen}, and P_{sleep} to denote the power a node consumes to transmit, receive, listen, and sleep, respectively. We test the following protocols:

- **ALWAYS ON [7]:** This is the IEEE 802.11 protocol with no power save. It is the default, unmodified MAC protocol in *ns-2*. Because nodes never sleep, ALWAYS ON uses the most energy, but has the lowest latency.

- **802.11 PSM [7]:** This is the standard IEEE 802.11 protocol with power save enabled. 802.11 PSM is described in Chapter 2.

- **CS-ATIM:** This is 802.11 PSM with the carrier sensing modification described in Section 3.1.1.

- **DCS-ATIM:** This is 802.11 PSM with the dynamic ATIM modification for multi-hop networks described in Section 4.1.1.

- **802.11 MIN:** This protocol needs more explanation because we are unaware of any other work which uses it. 802.11 MIN represents the minimum latency and energy consumption possible *for the IEEE 802.11 protocol*. We do not claim, nor believe, that it is optimal across the entire range of possible MAC protocols. However, it provides a useful baseline to measure other protocols against, since energy consumption and latency are two competing metrics and the desired tradeoff between these metrics is application-dependent. The latency for 802.11 MIN is simply equal to the latency for ALWAYS ON. Generally, ALWAYS ON is better than any power save protocol in latency since a node can immediately begin contending for medium access rather than waiting for the next scheduled wake-up time for the sender and receiver. To calculate 802.11 MIN's energy, a node consumes P_{tx} power while sending a packet, P_{rx} power while overhearing a packet, P_{listen} power while deferring and backing off as required by IEEE 802.11, and P_{sleep} power at all other times. Essentially, for a given scenario, 802.11 MIN represents the lowest possible energy achievable for nodes *using IEEE 802.11* if they slept as aggressively as possible (i.e., a node sleeps whenever they are not sending a packet or attempting to access the channel). Obviously, such a protocol is not possible since it requires the receiver to have perfect, advance knowledge of when a sender will attempt to begin contending for the channel to send a packet and wake up at that time (even if the two nodes had never communicated previously).

We use 2 Mbps radios that have a 250 m range. Each data point is averaged over 30 tests. The choice of a different channel bitrate would have the following effects on the protocols. For CS-ATIM, the carrier sensing period is rate-independent [109]. Thus, if T_{aw} is normalized to the packet transmission time, the carrier sensing time will be a higher overhead at a high bitrate and a lower overhead at a low bitrate. For DCS-ATIM, T_{idle} from Equation 4.1 will be larger for a lower bitrate and smaller for a higher bitrate since it is a function of how long it takes to transmit ATIM and ATIM-ACK packets.

For each multi-hop scenario, we place 50 nodes uniformly at random in a 1000 m $\times$ 1000 m area and consider only scenarios in which every node has a route to every other node in the network. To avoid second-order effects from routing protocols (e.g., the long delay for RREQs to traverse a power save network), we use Floyd-Warshall's All-Pairs Shortest Path algorithm [110] to precompute routes for all the nodes.

We vary different parameters for each test, but the following values are used when the parameter is not being varied. The beacon interval length is 100 ms and T_{aw} is 20 ms. Five flows send 512 byte data packets at a rate of 1 kbps per flow (i.e., each flow uses about 0.05% of the channel bitrate *per hop*). We test the effects of increasing the per-flow rate. We use a relatively low rate because at high rates, power save protocols become ineffective since nodes essentially transition to the ALWAYS ON state.

The sender and receiver of each flow are chosen uniformly at random and the traffic is constant bitrate (CBR) unless otherwise noted. With CS-ATIM, the carrier sensing time, T_{cs}, is set to 1 ms, which is about 66 times larger than the 15 μs required by 802.11 compliant hardware. We set T_{cs} to be large to mitigate the effects of short-term fading. In DCS-ATIM, the maximum backoff interval size, CW_{aw}, is set to be 63 slots. For the parameters we use, T_{idle} is set to 3.19 ms according to Equation 4.1. For the power characteristics of the radio [66, 70], we use: $P_{tx} = 1.4\,\text{W}$, $P_{rx} = 1.0\,\text{W}$, $P_{listen} = 0.83\,\text{W}$, and $P_{sleep} = 0.13\,\text{W}$.

As mentioned earlier, CS-ATIM and DCS-ATIM are vulnerable to false positives when they erroneously carrier sense the presence of a signal. Thus, in some of our tests we evaluate the effect of false positives on the protocols by specifying a percentage that represents the probability that a node remains on for the ATIM window even when none of its neighbors transmitted a dummy packet. For example, a 10% chance of false probabilities means that with probability 0.1, a node running the protocols remains on for the ATIM window even though there were no dummy packets transmitted.

In this chapter, we present tests that measure energy consumption and latency by varying the following parameters:

- **Beacon Interval Time:** We vary the length of the beacon interval to increase the amount of sleep time between beacon epochs.
- **Per-Flow Rate:** We increase the rate at which each of the flows in the network is sending packets.
- **False Positives:** For our protocols, we show how false positives (i.e., erroneously detecting the channel as busy) affect the energy consumption.

In our tests, energy consumption is measured in units of Joules/bit. This is calculated by dividing the total energy consumed by all nodes in a scenario by the total number of data bits that are received by their final destination. The latency is calculated as the average end-to-end latency over all packets received by their final destination in a given scenario.

Evaluating CS-ATIM and DCS-ATIM: First we tested the power consumption and latency of CS-ATIM and DCS-ATIM. For these tests, we varied the length of the beacon interval from 40 ms to 150 ms. As shown in Figure 4.2, all of the power save protocols show a decrease in energy consumption as the beacon interval is increased since this allows nodes more sleep time between ATIM windows. We see that CS-ATIM and DCS-ATIM both perform significantly better than 802.11 PSM. CS-ATIM and DCS-ATIM use about the same amount of energy and consume anywhere from 30% to 60% less energy than 802.11 PSM for the parameters tested. All protocols do significantly better than ALWAYS ON; even 802.11 PSM consumes anywhere from 40% to 70% less energy than ALWAYS ON. When compared to 802.11 MIN, CS-ATIM and DCS-ATIM use only about 18% to 30% more energy. The standard deviation for any single data point in Figure 4.2 never exceeds 4.5% of the mean.

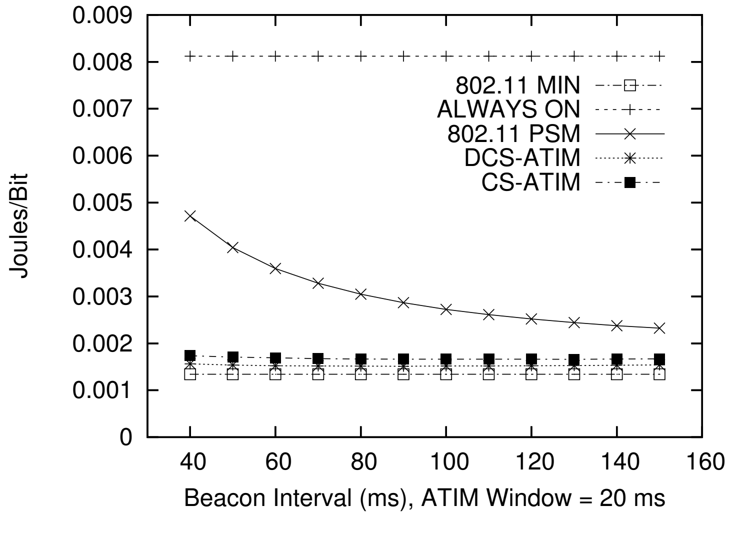

Figure 4.2: Energy vs. beacon interval.

The disadvantage of using power save protocols is evident in Figure 4.3 that shows the latency of the protocols. Just as an increasing beacon interval decreases the energy consumption, it increases the latency since there is a greater probability packets arrive outside the ATIM window and the time that these packets have to wait to be advertised increases. ALWAYS ON, and hence 802.11 MIN by definition, always do significantly better than the power save protocols. The maximum standard deviation for any single point on a curve in Figure 4.3 is between 34% and 35.8% of its mean depending on the protocol.

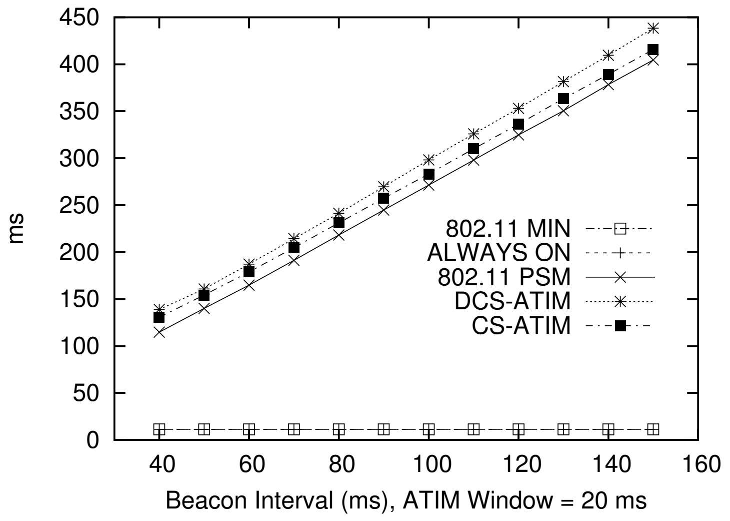

Figure 4.3: Latency vs. beacon interval.

For the power save protocols, 802.11 PSM always has the lowest latency of the power save protocols. CS-ATIM, however, tends have a slightly higher latency. The difference between CS-ATIM's latency and the latency of 802.11 PSM is relatively constant in the range of 8 ms to 15 ms. This small increase in CS-ATIM latency is due to the greater probability that packets may arrive during 802.11's longer ATIM window. DCS-ATIM has a slightly larger latency than CS-ATIM because of the extra carrier sensing period as well as the fact that sender's may occasionally have to postpone their advertisement until a later ATIM window.

In Figure 4.4 and Figure 4.5 we show how an increased sending rate affects the protocols. In these tests, the sending rate of each of the five flows is increased from 1 kbps to 10 kbps (i.e., each flow uses about 0.5% of the channel bitrate *per hop*). We see that DCS-ATIM does even better relative to CS-ATIM in this setting since a larger fraction of the ATIM windows have at least one advertisement to be sent. In this case, CS-ATIM has the same energy consumption as 802.11 PSM. However, DCS-ATIM can do better by allowing nodes to return to sleep earlier when only one advertisement is sent. The maximum standard deviation for any point on the ALWAYS ON curve in Figure 4.4 is about 0.5% of its mean; for the other curves in the figure, this standard deviation metric ranges from 8% to 12.5%. In Figure 4.5, the maximum standard deviation for any single point on a curve ranges from 34.5% to 42.7% of its mean depending on the protocol.

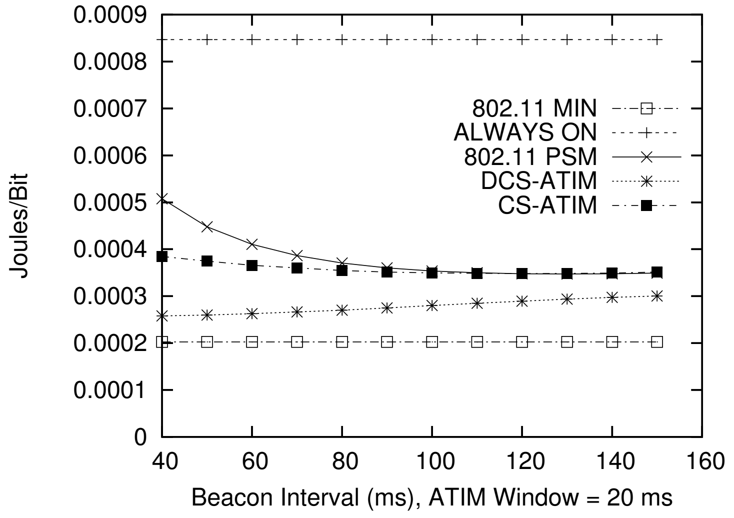

Figure 4.4: Energy vs. beacon interval with 10 kbps flows.

However, as Figure 4.5 shows, this improved relative energy consumption comes at the cost of increased latency. As the beacon intervals get longer with the higher sending rate, more contention occurs during the ATIM window and, hence, a greater chance that a node with an ATIM to send will have to delay the transmission until a later ATIM window, which significantly increases the delay of that packet.

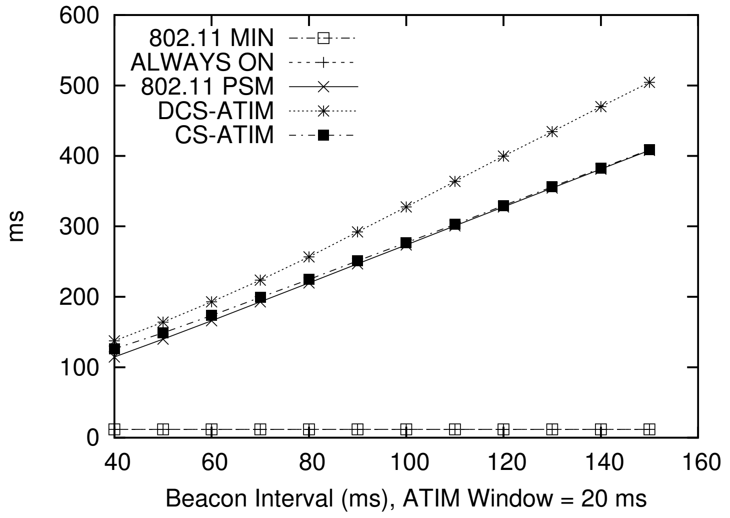

Figure 4.5: Latency vs. beacon interval with 10 kbps flows.

This increased contention and advertisement delay also explains the gradual increase in energy consumption for DCS-ATIM seen in Figure 4.4. We set DCS-ATIM to resort to CS-ATIM when a packet cannot be advertised for three consecutive ATIM windows. Thus, as DCS-ATIM uses more static ATIM windows, its energy consumption approaches that of CS-ATIM.

In Figure 4.6 and Figure 4.7, we see this trend as a function of per-flow rate. We note that the figures show the values of these metrics *relative* to 802.11 PSM (i.e., 802.11 PSM is always equal to one). CS-ATIM converges to 802.11 PSM in terms of energy and latency when the load increases to the point that packets are being advertised every ATIM window. DCS-ATIM, however, tends to plateau relative to 802.11 PSM when the rate reaches the point where packets are being advertised every beacon interval. In Figure 4.6 and Figure 4.7, the maximum standard deviation of any single point on any single curve as a percentage of its mean is 11% and 26%, respectively.

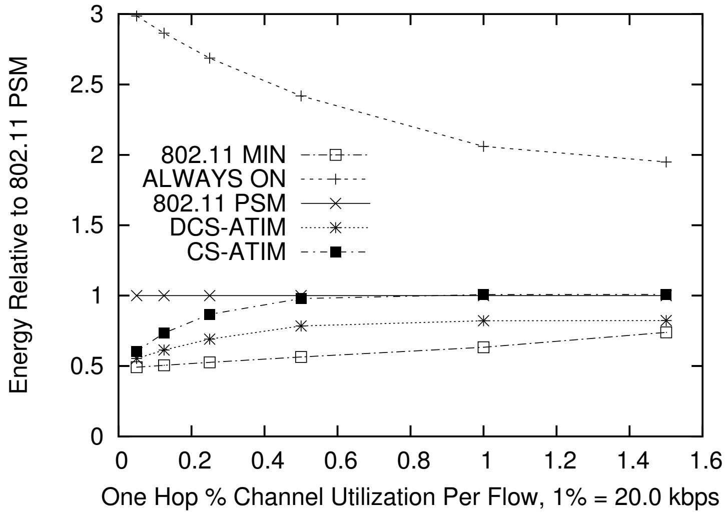

Figure 4.6: Relative energy consumption vs. per-flow rate.

False Positives As mentioned earlier, our protocols are susceptible to false positives when nodes carrier sense the channel as busy even though no dummy packet was sent. In this case, nodes waste energy by staying up for an ATIM window when no packets need to be advertised. In Figure 4.8, we see that CS-ATIM and DCS-ATIM show a linear increase in energy consumption as the false positive probability increases. In the worst case, when the false positive probability is equal to 1, the energy consumption of our protocols converges to slightly more than that of 802.11 PSM since they still have the overhead of carrier sensing. The maximum standard deviation for any single point on any single curve in Figure 4.8 is 3.6% of its mean.

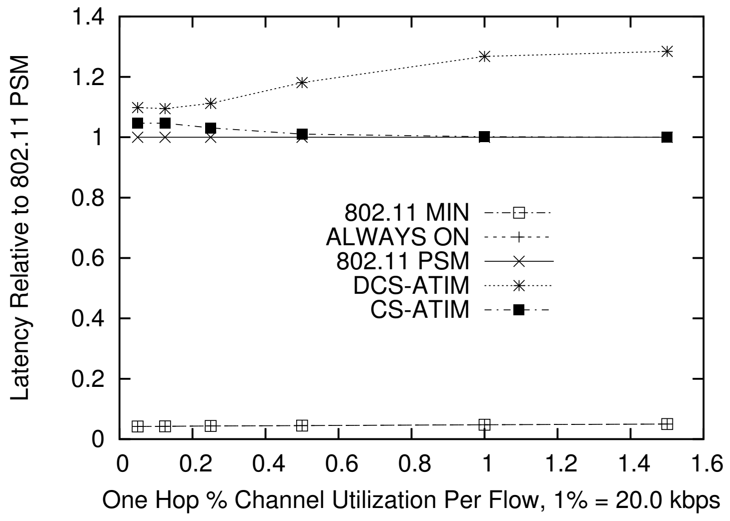

Figure 4.7: Relative latency vs. per-flow rate.

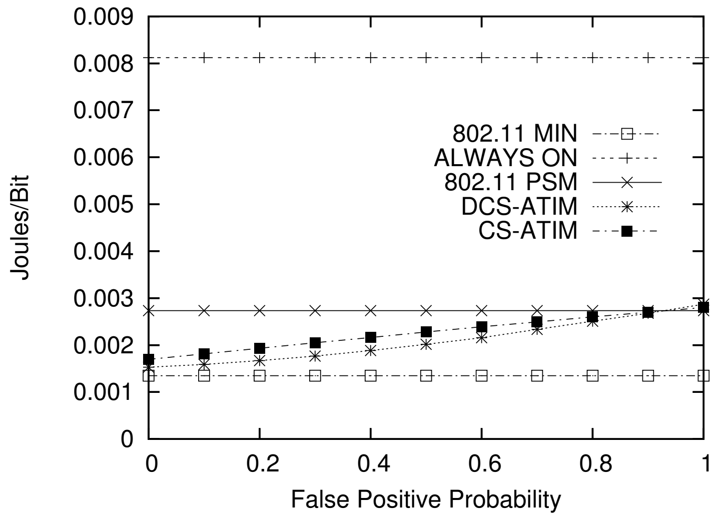

Figure 4.8: Energy vs. false positive probability.

4.2 Multi-Level Power Save Routing

In this section, we propose an adaptive sleeping approach for an in-band protocol. This is somewhat different from our previous work [27–29] on adaptive sleeping for out-of-band protocols in that we use the routing layer as opposed to only the link layer. The sleep intervals adapt in response to the desired latency of the traffic on a path.

In particular, we design a routing protocol for networks that use multiple levels of power save protocols. Each level of power save provides a different energy-latency tradeoff (i.e., a level with a lower latency requires more energy). We note that this paradigm is a generalization of the environment in [66, 67] (discussed in

Section 2.1.3) where only two levels of power save are assumed (i.e., (1) not using any power save and (2) using 802.11 PSM). This allows applications (e.g., sensor reports) to achieve an acceptable latency while reducing energy consumption in the network.

By incorporating the routing layer in the power save process, we believe that the energy-latency tradeoff can be better adapted to fit the needs of an application. For example, consider the scenario where data packets are being sent from **A** to **C** via the route $\mathbf{A} \rightarrow \mathbf{B} \rightarrow \mathbf{C}$. Using network layer information, we can design power save protocols to consider the whole route. In pure link-layer-based approaches, the power save protocol would run independently at links $\mathbf{A} \rightarrow \mathbf{B}$ and $\mathbf{B} \rightarrow \mathbf{C}$.

The idea of using multilevel design to achieve acceptable tradeoffs is prevalent in computer science (see [111] and references therein). For example, in computer architecture, accessing cache is much faster than main memory. However, main memory is cheaper in terms of cost per byte and is capable of storing much more data.

In Section 4.2.1, we give an overview of the link layer protocol that provides multilevel power save. In Section 4.2.2, we describe our routing protocol to effectively use multilevel power save. We present simulation results in Section 4.2.4.

4.2.1 Link Layer Protocol Description

First, we need to specify how the link layer power save protocols can be designed to provide k levels of power save, each with different energy-latency characteristics. Many power save protocols can be adapted to achieve this as discussed later in this section. We use 802.11 PSM [7] as the underlying power save protocol, which is described in detail in Chapter 2. This protocol is used because it is in-band and for the reasons discussed in Section 2.1.1, such as well-documented specification and widespread usage.

The 802.11 PSM protocol can be adapted to provide k levels of power save by changing how frequently a node wakes up to listen during an ATIM window based on its current power save level. We denote these k power save levels as $PS_0, \ldots, PS_{k-1}$. Without loss of generality, we assume that PS_0 corresponds to the "always on" state and PS_{k-1} uses the least amount of energy, but has the highest latency. In PS_0, the nodes never sleep and, thus, can receive a packet with the lowest latency, but also consume the most energy. The next level, PS_1 corresponds to the standard implementation of 802.11 PSM. That is, when a node is not sending or receiving any packets, it wakes up for every ATIM window and sleeps for the remainder of the beacon interval. In PS_2, nodes wake up only every other ATIM window. This allows them to save about twice as much energy as the nodes in level PS_1 while also doubling the latency to send or receive a packet.

Because we want to ensure that every node has their ATIM overlap with every other node periodically, we increase the sleep time for each level by a factor of two. This is a simple method to guarantee overlap, but more complicated schedules [41, 42] may work as well. Thus, to calculate the beacon interval for level PS_i, we have:

$$BI_i = 2^{i-1} \times BI_{base} \quad \text{, when } i > 0 \tag{4.2}$$

where BI_i is the beacon interval for the i-th power save level and BI_{base} is the base beacon interval specified for the system (i.e., $BI_1 = BI_{base}$).

Figure 4.9 illustrates the multilevel link layer protocol with $k = 4$. In this figure, AW corresponds to the ATIM window size and we show only the case in which no traffic is being sent. The beacon intervals of the four power save levels are: $BI_0 = 0$, $BI_1 = t_1 - t_0$, $BI_2 = t_2 - t_0$, and $BI_3 = t_4 - t_0$. The base interval is t_1 (i.e., $BI_{base} = t_1$).

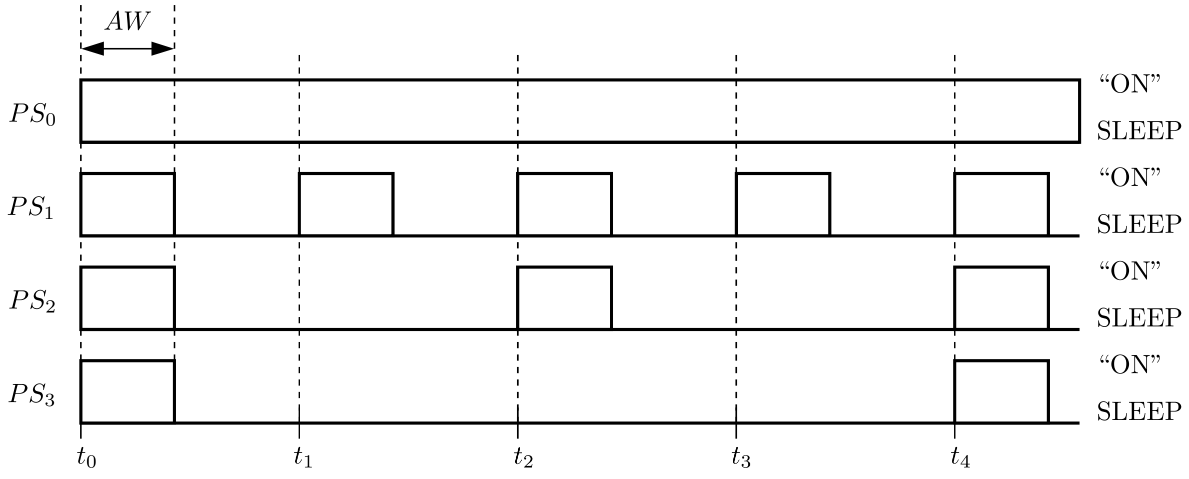

Figure 4.9: Multilevel power save with 802.11 PSM [7].

The largest possible beacon interval, BI_{k-1}, serves as the reference point for all of the nodes to ensure that they remain in phase. That is, the first ATIM window for which a node awakes in a cycle must always occur at the beginning of a reference point beacon interval (that is spaced BI_{k-1} time units after the previous reference point). The reference points in Figure 4.9 are at t_0 and t_4.

Since we assume that the nodes are synchronized, each node is initialized with the time of the previous reference point. Alternatively, if a node is added to the network later, it can learn the time of the previous reference point from older nodes in the network, along with the ATIM window size, BI_{base}, and the number of power levels the network is using via 802.11 management frames. This guarantees that for any two nodes, one with PS_i and the other with power level PS_j where $i < j$, then the node with PS_i will be awake during every ATIM interval that the node with PS_j is awake since BI_j is divisible by BI_i.

Each node keeps track of its neighbors' power save state as follows. On every data and ACK packet a node sends, it attaches its current power save level. We do not test the consistency of a node's power save table. However, our protocol could use a scheme similar to the one in [66] whereby the first time a packet transmission fails, a node sets the intended receiver's power save state to PS_{k-1}. Recall that PS_{k-1} has the longest beacon interval and all nodes, no matter what power state, are guaranteed to wake up every BI_{k-1} time units. Thus, if the neighbor still exists near the node, communication should be possible during this beacon interval. If a transmission fails again for the receiver using PS_{k-1}, then the link is considered dead and reported to upper layers.

This is just one example of how a power save protocol can be modified to achieve multiple levels with different energy-latency tradeoffs. Other examples include adjusting the time between listening periods in protocols such as STEM [2, 3] and WiseMAC [58]. Nodes using a longer sleeping time between listening periods would save more energy, but require a longer latency to be awakened by neighbors.

4.2.2 Routing Protocol Description

In Section 4.2.1, we described how to *provide* multilevel power save. In this section, we describe a routing protocol to *efficiently use* multilevel power save. If energy consumption is the only concern, the optimal adaptive sleeping strategy is simply for every node to select PS_{k-1} as their power save state. However, this results in large delays due to the power save protocol that may be unacceptable for many applications.

Thus, our protocol works by taking an application-defined latency bound and trying to find a route to achieve the bound while attempting to minimize the increase in energy consumption. We focus on only the latency induced by the power save protocol because this delay tends to be large (e.g., hundreds of milliseconds or even seconds *per hop*) relative to contention and queuing delay in non-congested networks. In highly congested networks, power save protocols would most likely not be used. A vast body of QoS research deals with congestion and queuing delay which we view as orthogonal and complementary to our work.

If we are given a set of m flows to route $(F_1, F_2, \ldots, F_m)$ and a desired latency for each flow $(L_1, L_2, \ldots, L_m)$, finding routes that minimize the overall energy consumption increase for the flows is NP-complete. A proof of this is presented in Appendix B. In this work, therefore, we consider heuristics to address the problem.

We modify DSR (Dynamic Source Routing) [77] to obtain our routing protocol. We now give a brief overview of the salient aspects of DSR. When a source, S, wants to send packets to a destination, D, it must first discover a route. To do this, S broadcasts a route request packet ($RREQ$) that is flooded throughout the network specifying that it is trying to find a route to D. Each node, other than D, that receives S's

RREQ will add itself to a node list in the packet and rebroadcast the *RREQ* (assuming that the TTL of the *RREQ* has not expired).[4] Each *RREQ* is rebroadcast only once by an intermediate node. So, if multiple copies of the same *RREQ* are received by a node, as determined by a unique sequence number for the request, the node will forward only the first one that it receives. If the *RREQ* reaches D, it generates a route reply (*RREP*) packet and sends it to the source.[5] The *RREP* packet is generated by reversing the node list in the *RREQ* and sending the *RREP* along the path specified by the node list. The entire node list is included in the payload of the *RREP* packet and is also used for source routing the packet to S. A node that receives a source-routed packet will only forward it if the node's ID is next on this node list. To do so, it transmits the packet to the next node ID specified on the list. In this manner, the *RREP* makes its way back to S. At this point, S extracts the node list from the payload of the *RREP* and uses it as the source route to forward data packets. That is, every data packet that S sends will have the node list appended to the routing header.

We modify DSR as follows. The *RREQ* sender adds its desired latency, L, for the flow to the *RREQ* packet. When forwarding the *RREQ*, each node will append its current power save state in addition to its node ID. When D receives its first *RREQ* from S, it will set a timer for some specified time, T_{delay}.[6] D will not send a *RREP* until this timer expires. While the timer is running, D will collect all *RREQ*s with the same sequence number that it receives from S. At the end of this T_{delay} time, D will evaluate all the paths it has received from these *RREQ*s and send an *RREP* along the "best" path. Next, we specify the routing metrics that are used to determine which path to use.

The goal of our routing metric is to find the path that can achieve the desired latency, L, (specified in the *RREQ*) while increasing the energy consumption in the network the least. To do this, we consider paths that have been collected during the *RREQ* reception phase. For each path, we find the node on the path whose energy consumption will increase the least by moving to the next higher energy state (and, hence, lowering the latency for that hop). We continue iterating in this manner until the path's end-to-end power save-induced latency is less than L or all nodes are in the highest energy state. At this point, we store the total energy increase for the path that was necessary for the iteration to terminate. Once this has been done for all the paths, we send the *RREP* on the path that requires the smallest total energy consumption increase. If two or more paths are tied for the minimum cost, then our protocol prefers routes with the

[4]Non-destination nodes replying to *RREQ*s using cached routes is one of *many* extensions that has been proposed for DSR. We do not use cached replies in our work.

[5]Another option in DSR is whether the destination replies to every *RREQ* it receives or just the first one. In our protocol, the *RREP* procedure is modified, but the destination will send only one *RREP* per *RREQ*.

[6]Another option is that a node replies after receiving some number, say x, *RREQ*s even if the T_{delay} timer has not yet expired. For example, if $x = 1$, then a node would just calculate the power save state changes required for the path on the first *RREQ* that it receives and use that path (and disregard all subsequent *RREQ*s for that route discovery). If $x = 2$, the node would consider only the first two *RREQ*s that it receives and cancel the T_{delay} timer if it has not yet expired.

lowest hop count.[7] Our algorithm is shown in Figures 4.10, 4.11, and 4.12.

Each node receiving the *RREP* will check the requested power save level set for it by the destination. If the requested power save level is a higher energy level than its current level, then the node switches to the new power save level. Otherwise, it will remain in its current power save state since it is sufficient to maintain the desired latency of the path.

```
FIND-ROUTE($R, L$)
 1   /*****
 2    * Find route on which to send the $RREP$
 3    * given a list, $R$, of received $RREQ$s
 4    * and latency threshold, $L$
 5    *****/
 6
 7  $isFirst \leftarrow$ true
 8  for each $r$ in $R$
 9  do $cost \leftarrow$ ENERGY-INCREASE($r, L$)
10      if $isFirst$ or $cost < min$
11         then $min = cost$
12              $minRREQ = r$
13              $isFirst \leftarrow$ false
14
15   /* Set the requested power levels for the chosen path */
16  ARRAY-COPY($psLevels[minRREQ], newPsLevels[minRREQ]$)
17
18   /* Reply using the path from $minRREQ$ */
19  SEND-RREP($minRREQ$)
```

Figure 4.10: Algorithm for determining which path to use from collected *RREQ*s.

The FIND-ROUTE function in Figure 4.10 finds the route to use based on R, the set of *RREQ*s that have been collected. For each *RREQ*, FIND-ROUTE calls ENERGY-INCREASE (discussed below) to calculate the cost of using the *RREQ*'s route in terms of how much the energy consumption of the path must be increased to reach the latency threshold, L. At the end of the **for** loop, the least costly path is found and the power save states are set to the new power levels necessary to achieve the latency threshold (*newPsLevels* is a global variable set in ENERGY-INCREASE). With these updated power levels, the *RREP* is constructed and sent along the chosen path via the call to SEND-RREP.

The ENERGY-INCREASE function in Figure 4.11 computes the minimum increase in energy consumption necessary for the path in a *RREQ*, r, to achieve the desired latency, L. First, the function makes a copy of the power save levels of the nodes in r's path ($psLevels[r]$) since our algorithm needs to change this state. The *energyCost* variable keeps a running total of the increase in energy consumption required for r's path to reach L. The **while** loop on line 21 will continue until the latency of the path is less than L (we assume that

[7]We note that other metrics such expected number of transmissions or packet loss [112] could be used instead.

```
ENERGY-INCREASE($r, L$)
 1   /*****
 2    * Find the minimum energy consumption increase required
 3    * for the path in a $RREP$, $r$, to achieve a wake-up
 4    * latency less than or equal to $L$.
 5    *****/
 6
 7   /*****
 8    * $psLevels[r]$ contains the current power save level
 9    * of each node along the path in $r$.
10    * $psLevels[r]$ is an array with an element for each node on the path.
11    *****/
12
13   /* $pathLen[r]$ is the length of the path in $r$ */
14
15  ARRAY-COPY($newPsLevels[r], psLevels[r]$)
16  $energyCost \leftarrow 0$
17   /*****
18    * We assume that PATH-LATENCY $\leq L$
19    * when $newPsLevels[r][i] = 0$ for all $i$ on the path
20    *****/
21  while PATH-LATENCY($r$) $> L$
22  do $isFirst \leftarrow$ true
23      for $i \leftarrow 1$ to $pathLen[r]$
24      do $cost \leftarrow$ ENERGY-DIFF($newPsLevels[r][i], (newPsLevels[r][i] - 1)$)
25          if $cost \neq 0$ and ($isFirst$ or $cost < min$)
26              then $min = cost$
27                   $minIndex = i$
28                   $isFirst \leftarrow$ false
29      $energyCost \leftarrow energyCost + min$
30      $newPsLevels[r][minIndex] \leftarrow newPsLevels[r][minIndex] - 1$
31  return $energyCost$
```

Figure 4.11: Algorithm for computing cost of a path to reach latency threshold L.

this will always terminate in the pseudocode). Each iteration of the **while** loop will calculate the difference in energy consumption that would result for each node in r's path if its current power save state was moved to the next lower latency power save state (i.e., moving from PS_i to PS_{i-1}). This calculation is done via the call to ENERGY-DIFF, which is discussed below. Once the **for** loop on line 23 has terminated, we have identified the node on r's path who can transition to a lower latency power save state with the smallest increase in energy consumption. At this point, we transition to the lower latency power save state (using the $newPsLevels$ variable) and increment $energyCost$ by the energy consumption increase required. When the path latency (calculated by the PATH-LATENCY function call) is less than L, the **while** loop terminates and returns $energyCost$.

The PATH-LATENCY function in Figure 4.11 (line 21) can be computed in terms of worst-case latency, average-case latency, or some other metric. We assume that a node j is using the k_j-th power level. Thus, PS_{k_j} denotes its power save level and BI_{k_j} is the length of its beacon interval. Thus, for a path of n nodes, and the worst-case latency metric, our protocol considers the route for use if:

$$BI_{k_1} + BI_{k_2} + \cdots + BI_{k_n} < L \tag{4.3}$$

```
ENERGY-DIFF($oldLevel, newLevel$)
  /*****
   * Find the difference in energy consumption for
   * switching from the lower energy $oldLevel$
   * to the higher energy $newLevel$
   *****/

  /*****
   * $atimSize$ is a parameter specified elsewhere.
   * It is the size of the ATIM window in units of time.
   *****/

 if $oldLevel = 0$ or $newLevel > oldLevel$
    then return 0

 $oldEnergy \leftarrow \frac{atimSize}{beaconIntervalSize[oldLevel]}$
 if $newLevel > 0$
    then $newEnergy \leftarrow \frac{atimSize}{beaconIntervalSize[newLevel]}$
    else $newEnergy \leftarrow 1$
 return $(newEnergy - oldEnergy)$
```

Figure 4.12: Algorithm for computing the energy consumption difference between two power save levels.

The ENERGY-DIFF function in Figure 4.12 computes an energy cost for transitioning from one power save state to a lower latency power save state. We compute the energy consumption of a power save state as the ATIM window size (*atimSize*, whose value is set elsewhere) divided by the power save state's beacon interval size (i.e., BI_i). The *beaconIntervalSize* variable is an array indexed by the beacon interval sizes for each power save state. Note that this energy consumption calculation considers only the energy consumption when nodes are not awake after the ATIM window. When nodes *are* awake following the ATIM window, the energy consumption used in the subsequent beacon interval is the same regardless of the power save state. As an example, let $atimSize = 20\,\text{ms}$, and $BI_i = 200\,\text{ms}$ and $BI_{i-1} = 100\,\text{ms}$. In this case, ENERGY-DIFF(PS_i, PS_{i-1}) will return $\frac{20}{100} - \frac{20}{200} = 0.1$.

Though we do not test this in our simulations, each node must set a soft timer for each flow for which it forwards packets so that it can revert to lower energy states whenever that flow ceases or the route fails. Because the inter-arrival time for the packets on a flow is highly application dependent, we propose letting the application specify this timeout value and piggybacking it on data packets sent by the flow. Whenever a flow times out or explicitly indicates that it will no longer use the route, the node transitions into the lowest energy power save state that is still acceptable to the flows which continue to use that node on their route, as indicated by the power save levels specified for the node in RREPs that it has received.

4.2.3 Design Discussion

Wake-Up Schedules: As described in Section 4.2.1, we use a simple link layer protocol to provide multiple levels of power save. Basically, the beacon interval either increases by a factor of two or decreases by half depending on whether the node is moving to a lower or higher energy state, respectively. An alternative is to use more complex wake-up schemes that provide overlap either deterministically or probabilistically.

In general, probabilistic protocols (e.g., [44]) are not appropriate in our design since they essentially add more uncertainty to an already unreliable channel. Additionally, these protocols make even soft real-time constraints more difficult to obtain. Thus, we do not consider probabilistic approaches for our protocol.

By contrast, protocols that give deterministic overlap in an asynchronous manner (e.g., [41, 42]) do allow soft real-time latency bounds. The basic idea is that each node wakes up according to some pattern that is guaranteed to overlap within some bound with every other node even though they may be unsynchronized. The major advantage of this approach is that it makes synchronization less necessary. However, it can greatly increase the protocol complexity since the wake-up schedules have to be chosen appropriately and nodes still must probe to find out when the overlap occurs since they have no prior knowledge. Additionally, broadcast is a problem since there is no single time where a node is guaranteed to have all of its neighbors listening. We do not use a deterministic asynchronous protocol because we are not concerned with synchronization and we need a relatively reliable and low overhead broadcast mechanism for the route discovery in our work. This also frees us from the added complexity such a scheme would add to focus on the major idea of routing with multiple power save levels.

Another option is to have one, long "master" beacon interval in which everyone is awake (i.e., PS_{k-1} in our protocol). Then, each node chooses its own beacon interval independently based on the RREPs it receives and lets each communicating neighbor know the next time it is scheduled to awake. Nodes then keep track of the next wake-up time for each node with which they are communicating. This frees the nodes from the need to use specified discrete intervals and allows them to use any interval up to PS_{k-1}. Broadcast is still possible, as in our scheme, where broadcasts are sent only during the "master", or PS_{k-1}, interval. The disadvantage of this approach is that it requires the nodes to keep more per flow state. Also, it is more susceptible to nodes returning to sleep too early since nodes waking up experience contention from data packets, not just ATIM packets as in our scheme. Data packets tend to be significantly larger than ATIM packets. In future work, one could more fully explore this idea to see under what conditions the early sleep problem makes this protocol worse than our current version.

Soft Timers: As mentioned in Section 4.2.2, we use soft timers per flow passing through a node to determine when it can revert to a lower energy state. We believe that this is acceptable since, in many environments, a node will have only a few flows passing through it. Of course, a node can always choose *not* to handle additional flows if its per flow state becomes excessive.

An alternative to this design decision is to require a sender to explicitly "delete" a flow by sending a packet along the path when it is finished. We feel that this method would be unacceptable in multihop wireless network settings due to the inherent underlying reliability of the channel and devices. Because links and flows can fail unexpectedly, a node would permanently keep state for dead flows for which the flow was not deleted. Eventually, this could exhaust the node's memory resources. Thus, overall, we feel that the per flow state required to maintains soft timers for this purpose is best for the environment we are considering.

Routing Techniques: We choose to use DSR [77], a source routing protocol, in our work as described in Section 4.2.1. An alternative would be to use a distance vector approach, like AODV [78]. The disadvantage of using AODV (or another distance vector protocol) is that nodes learn only aggregate information about the path during routing as opposed to DSR which provides *per node* information. In the algorithms discussed in Section 4.2.1, we need per node information. In this aspect, DSR provides a superset of the information that AODV does. Because our algorithms do not work with the information from AODV, we use DSR in our work.

Another choice would be to use link state routing [113], such as OLSR [114]. Nodes could flood the network whenever their power save level changes or a link breaks. The obvious disadvantage of this approach is the high overhead to flood the network if power save states are changing relatively frequently. Also, as shown if Appendix B, even if the entire topology is accurately known, it is still NP-complete to find the minimal energy consumption increase required for a desired latency. Thus, the advantage of knowing the entire topology, as opposed DSR which learns just a few paths, is not easily exploited. At the very least, we could find the k shortest paths [115], given the entire topology, and run the algorithms from Section 4.2.1 on each of these paths. In future work, it would be interesting to test a link state routing protocol versus our DSR implementation to determine which performs better under different metric change frequencies and network sizes.

4.2.4 Simulation Results

To evaluate our protocol, we simulated it using *ns-2* [103]. We test the following schemes, where the bold text is the name we use to refer to the scheme and the italicized text indicates the (Routing, MAC) tuple used:

- **Always On [7, 77]** *(DSR, 802.11)***:** This is the IEEE 802.11 protocol with no power save. It is the default, unmodified MAC protocol in *ns-2*. Because nodes never sleep, ALWAYS ON uses the most energy, but has the lowest latency.

- **802.11 PSM [7, 77]** *(DSR, 802.11 PSM)***:** This is the standard IEEE 802.11 protocol with power save enabled. 802.11 PSM is described in Chapter 2. The beacon interval for this protocol is set to the longest beacon interval for a given k value.

- **CS-ATIM** *(DSR, CS-ATIM)***:** This is 802.11 PSM with our proposed carrier sensing modification described in Section 3.1.1. The beacon interval for this protocol is set to the longest beacon interval for a given k value.

- **Multilevel PSM** *(Multilevel DSR, Multilevel 802.11 PSM)***:** This is our proposed multilevel power save protocol described in Section 4.2 using 802.11 PSM.

- **Multilevel CS-ATIM** *(Multilevel DSR, Multilevel CS-ATIM)***:** This is our proposed multilevel power save protocol described in Section 4.2 using the CS-ATIM protocol that we proposed in Section 3.1.1.

We use 2 Mbps radios that have a 250 m range. Each data point is averaged over 30 tests. The ATIM window is 20 ms and the base beacon interval, BI_{base}, is 100 ms. Our topologies are generated by placing 50 nodes uniformly at random in a 1000 m×1000 m area. Each scenario has five flows among randomly chosen source and destination pairs. Each flow sends at rate one packet per second using CBR traffic. We set T_{delay}, the time that a destination waits to collect $RREQ$s to be 500 ms. In our experiments, we set L to be the same value for all flows in the network and do not test the more general case where each flow could select its own L value.

Since our protocols are designed to only achieve soft real-time bounds on latency, it is important to consider the standard deviation of our latency results. This gives us an indication of how well the protocols are able to stay within the bounds over multiple runs. To avoid cluttering our figures with standard deviation bars, we provide the numerical values in Table 4.1 (we will refer to this table in our discussion of the results). In this table, we give the standard deviation for each protocol in each latency figure as a percentage of the mean for the corresponding data point. We use the percentage since the mean values can vary significantly which makes the absolute values of the standard deviations difficult to compare. We compute the standard deviation *averaged* over all data points for the protocol as well as the *maximum* standard deviation of any one data point on a protocol's curve. Additionally, we *have* plotted the standard deviation bars for the latency of the multilevel protocols to show their deviation relative to the desired latency bound.

Table 4.1: Standard deviation as percentage of mean for latency figures (Average | Maximum).

	Figure 4.14		**Figure 4.16**		**Figure 4.19**	
Always On	29.06	29.06	25.99	25.99	29.00	29.00
802.11 PSM	33.77	52.54	29.39	29.39	56.94	56.94
Multilevel PSM	20.03	22.75	26.33	29.04	23.46	53.28
Multilevel CS-ATIM	17.61	19.43	24.86	35.56	22.04	61.39
CS-ATIM	36.06	52.01	26.94	26.94	43.72	43.72

Figure 4.13 shows energy consumption of the protocols when $L = 300\,\text{ms}$. The horizontal axis is k, the maximum number of power save levels. Since $T_{base} = 100\,\text{ms}$, $k = 2$ corresponds to the traditional 802.11 protocol where a node can either be on or using a power save protocol with a beacon interval of 100 ms. From the figure, we see that all the power save protocols use significantly less energy than the Always On protocol.

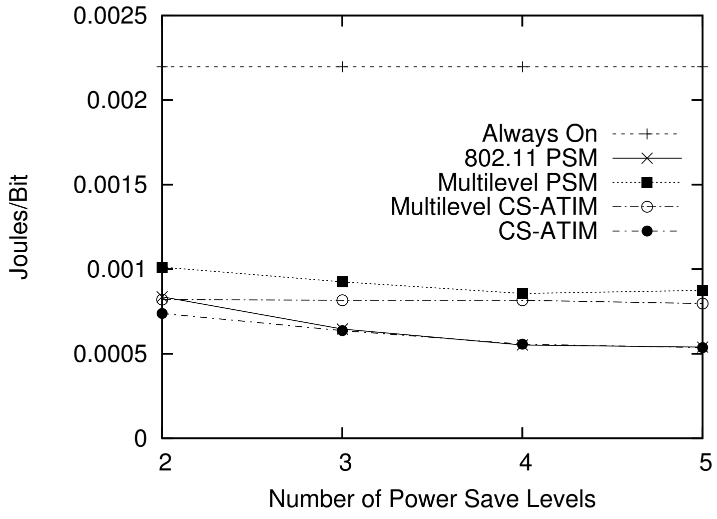

Figure 4.13: Effects of the number of power save levels on energy.

We see that the multilevel PSM protocol uses about 33% to 50% more energy than the traditional PSM protocol. However, this increase in energy comes with a huge reduction in latency as shown in Figure 4.14. In this figure, we measure only the latency for packets that are sent *after* the source has received the *RREP*.

The source queues packets while waiting for the *RREP*, which makes their delay rather large and can skew the average end-to-end delay of the rest of the packets.

The multilevel protocols achieve a delay of around 140 ms to 180 ms, which is well within the $L = 300$ ms bound that was given. By contrast, the non-multilevel protocols have a latency of just over 300 ms when $k = 2$ and increase to over 3000 ms when $k = 5$. For $k = 3$ and $k = 4$, we notice that the average latency is approximately double that of using the next lower latency power save state (i.e., $k = 3$ latency is about double that of $k = 2$ and $k = 4$ is twice as much as $k = 3$). However, when $k = 5$, the latency more than doubles over that of $k = 4$. The reason for this is that ATIM window contention causes significant delays. Since the ATIM window size is static regardless of k and the traffic rate remains the same, more packets need to be advertised in the ATIM window when $k = 5$ as opposed to, say, $k = 2$. The increased contention reaches a point where some nodes are unable to send an ATIM when they first try and must wait another beacon interval. This greatly degrades latency since the beacon intervals are longer for larger values of k. When $k = 5$, *each hop* has a wake-up latency of 800 ms plus the increased ATIM contention. With the multilevel power save protocols, the routing protocol adjusts the power save level of nodes along a path to ensure that the latency is less than L.

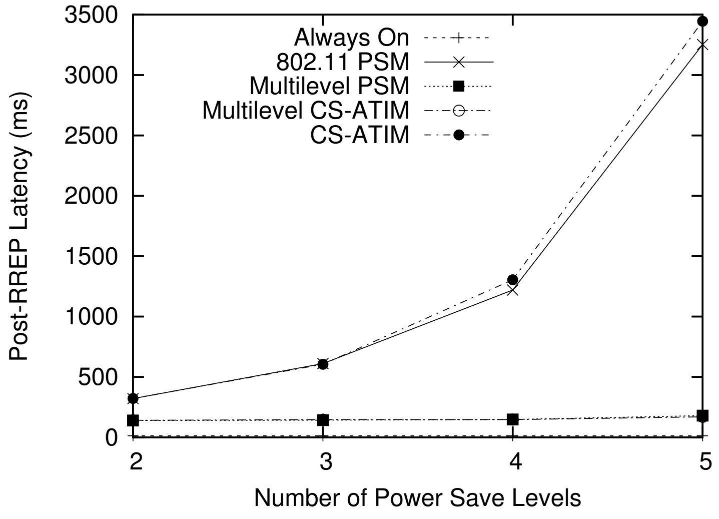

Figure 4.14: Effects of the number of power save levels on latency.

Additionally, we can see from Table 4.1 that the multilevel protocols in Figure 4.14 have a lower deviation in their latency among different runs than the corresponding protocol without the multilevel extension. The multilevel protocols have a deviation of about 20% on average, whereas PSM and CS-ATIM without the multilevel extension have about a 35% deviation on average. This is due to the fact that a wider range of average latencies are possible in the non-multilevel protocols for different topologies and traffic patterns.

From Figure 4.13, we can see that our carrier sense techniques from Section 3.1.1 integrate nicely with the multilevel power save scheme. In particular, by using CS-ATIM, we are able to achieve virtually the same latency at using PSM (and well below the L threshold) while consuming less energy than the PSM version of multilevel power save. All of the protocols seem to plateau at a point where the utility of adding more power save levels diminishes. The multilevel CS-ATIM protocol seems to reach this plateau with only two power save levels and shows only a slight decrease in energy consumption after this point.

In Figure 4.15, Figure 4.16, and Figure 4.17, we set $k = 2$ and show the effects of changing L, the desired latency, on energy consumption and the observed latency, respectively. Again, we see that the multilevel power save protocols achieve the latency bound with only a slight increase in energy. In particular, we can see that, for $k = 2$, if a latency of less than about 300 ms is desired, then the power save protocols that do not use multilevel power save cannot achieve this. Without multilevel power save, the only option would be to turn off power save which, as we can see from Figure 4.15, substantially increases energy consumption by more than a factor of two. Furthermore, in Figure 4.17, we can see that virtually *none* of the individual runs exceed the latency bound when using the multilevel extension. We note that a few of the flows *do* exceed the latency bound by a small amount. This occurs because our protocol adjusts the *power save induced* latency and does not account for transmission times and queuing delays. Thus, our protocol occasionally sets the power save states such that they are close to or equal to L, but the extra delays make the observed latency slightly higher than L.

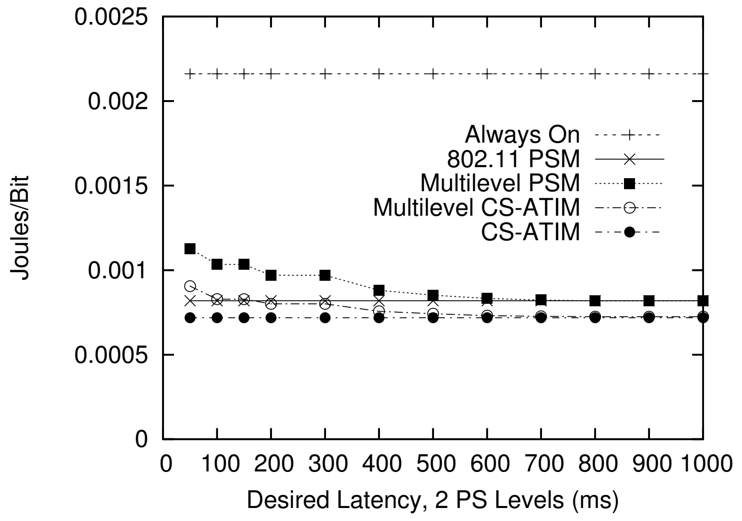

Figure 4.15: Latency threshold versus energy consumption using two power save levels.

In Figure 4.18, Figure 4.19, and Figure 4.20, we show the effects of changing L for $k = 3$. We can see that the multilevel power save protocols use slightly more energy relative to the other power save protocols

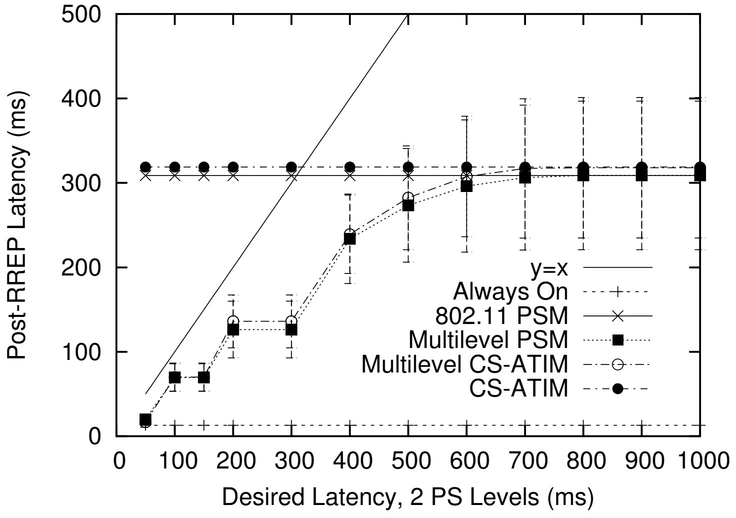

Figure 4.16: Latency threshold versus observed latency using two power save levels.

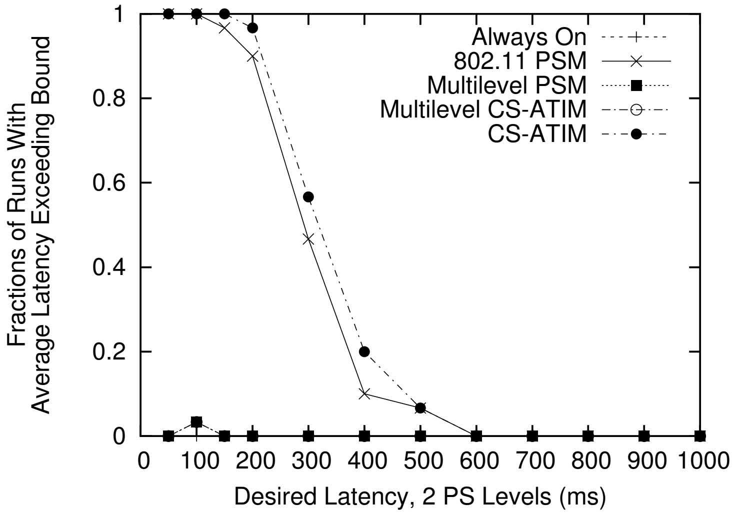

Figure 4.17: Latency threshold versus observed latency using two power save levels.

than for the $k = 2$ case. However, the multilevel power save protocols are also much more useful in achieving the latency bound. In Figure 4.19, we can see that an application with L up to about 600 ms cannot achieve its bound without the use of multilevel power save protocols or turning off power save all together. The 600 ms is a function of the average hop count in the network and beacon interval size. From Figure 4.16 and Figure 4.19, we can infer that the average hop count is approximately three in our scenarios since the latency with a 100 ms beacon interval is about 300 ms and with a 200 ms beacon interval is about 600 ms. As with the $k = 2$ case, we can see in Figure 4.20 that virtually *none* of the individual runs exceed the latency bound when using the multilevel extension. As discussed earlier, the bound is occasionally exceeded since our protocol only accounts for the power save induced latency whereas the observed value is also affected by the packet transmission time and queuing delay.

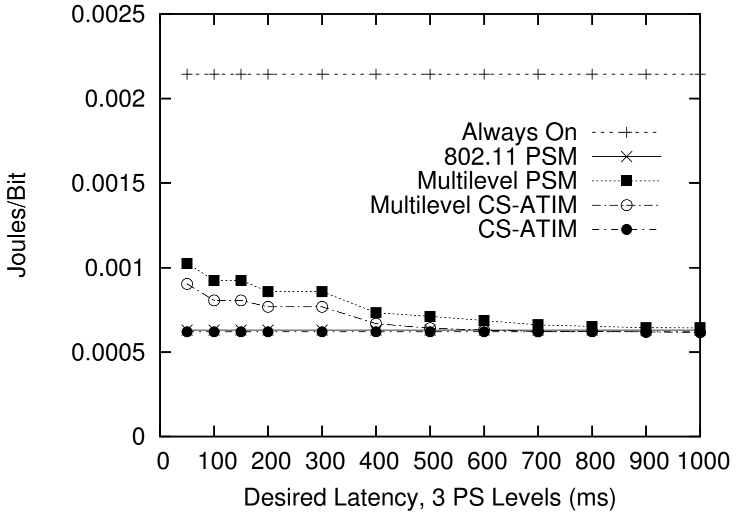

Figure 4.18: Latency threshold versus energy consumption using three power save levels.

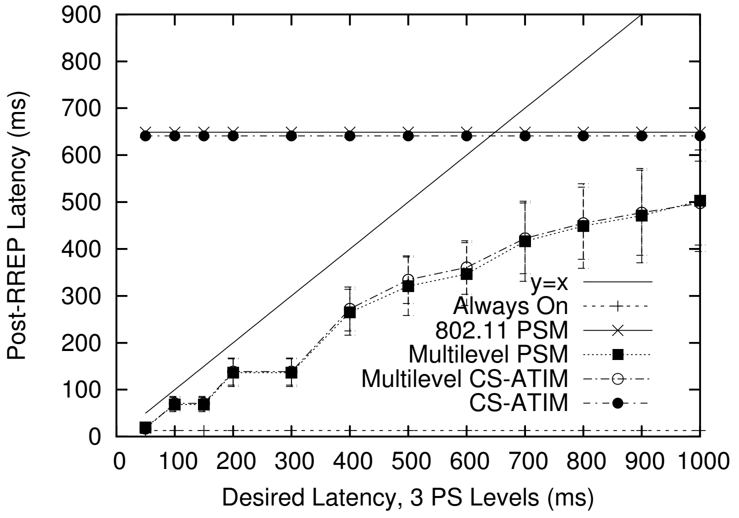

Figure 4.19: Latency threshold versus observed latency using three power save levels.

4.2.5 Extensions

Energy Load Balancing As in previous work [116–118], it is still a concern that certain nodes that are chosen to have a high energy power save state early may end up receiving a disproportionate amount of the network's traffic because they have a favorable metric. To address this, we propose that higher energy nodes periodically try to "patch" their place on the route with another node with a power level less than or equal

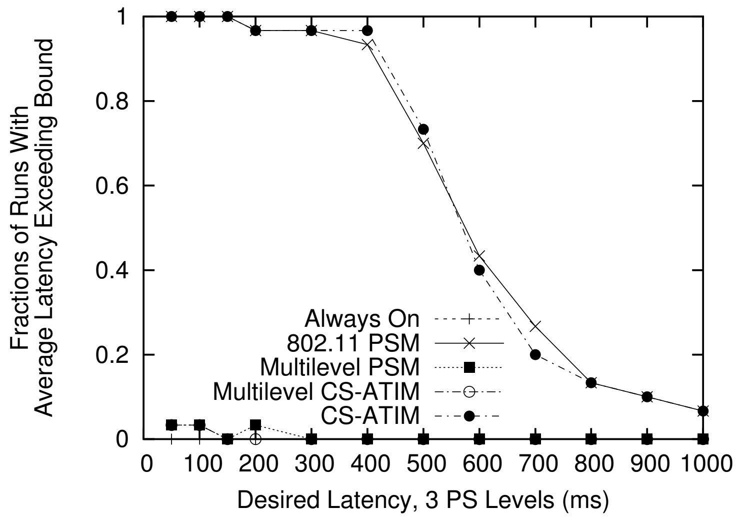

Figure 4.20: Latency threshold versus observed latency using three power save levels.

to it that can be reached by both its upstream and downstream neighbors on the route. A node could try this procedure when its residual energy falls below a specified level or when its recent energy consumption *rate* exceeds a certain level.

Such a situation may occur when two nodes, say A and B, are equivalent from a routing perspective and are in the same power save state when the *RREQ* is initially broadcast. In this circumstance, node A may be selected, for example, because it wins access to the channel before B and rebroadcasts the *RREQ* first. Thus, patching would allow A to eventually switch places with B to balance the energy consumption of the two nodes.

We note that others [67] propose delaying the *RREQ* proportional to remaining energy. However, a node with more energy at the time of the *RREQ* may eventually consume more energy than its neighbors and require load balancing. Also, such a scheme assumes a homogeneous environment where all devices have the same initial energy and/or they all consume energy at the same rate. In practice, this may not be true.

To do this, the node desiring the patch, say P, broadcasts a message that is received by both its upstream and downstream neighbors (nbr_{up} and nbr_{down}, respectively) asking them each to broadcast a packet to test which nodes are neighbors to both nbr_{up} and nbr_{down}. This packet also includes P's residual energy. Any node that receives both the packet broadcast by nbr_{up} and nbr_{down} and has more residual energy that P is a candidate to replace P on the path. Such nodes respond to P and then P can select the node with the highest remaining residual energy. Standard techniques such as choosing a backoff interval proportional to a node's residual energy can be used to ensure that nodes with a higher residual energy reply first.

The process of patching a route is shown in Figure 4.21. Here, we assume that traffic is being sent along the route $A \rightarrow B \rightarrow C$ and that B wants to try to remove itself from the path. Thus, B sends out a broadcast indicating that it wants to try to patch the route between A and C. In turn, A and C broadcast a packet to help other nodes determine their reachability. In this example, N_1, N_2, and N_3 cannot take B's place because they do not have both A *and* C as neighbors. The only two candidates to take B's place are N_4 and N_5, since both are neighbors of both A and C. In order for N_5 to take B's place, it would be necessary for it to communicate this to B via A and/or C. This is in contrast to N_4, which can communicate with B directly. This implies that the communication overhead and complexity for N_4 to be used is less than if N_5 is used. In order for N_4 or N_5 to take B's place on the route, they need to have more residual energy than B.

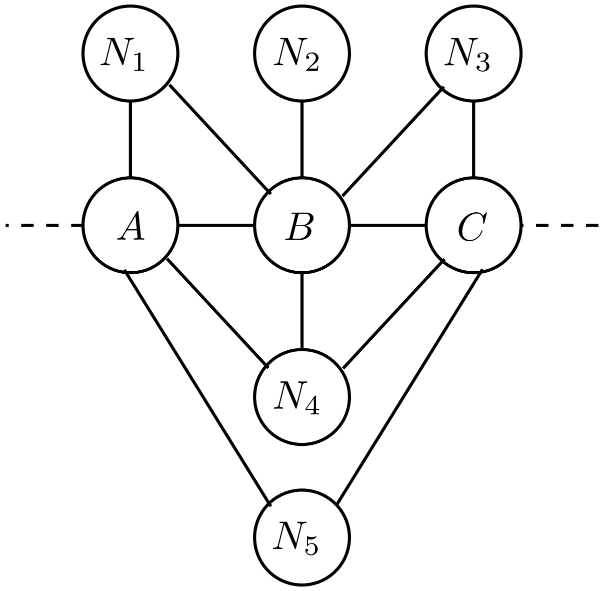

Figure 4.21: Patching a route in multilevel power save.

If a node is part of multiple, disjoint routes, it can still try this patch procedure incrementally by applying it to the path which requires the highest energy level until an acceptable level is reached. We note that in this scenario, a node may also need to account for the rate at which traffic is being forwarded on a given path since flows which require a lower energy power save level, say f_{low}, may still cause the node to consume more energy than a flow that requires a higher power save level, say f_{high}, if the f_{low} is sending at a higher rate that f_{high}. Another issue is instability in the route if two neighbors try to patch their place on the route simultaneously. If a node hears a patch request from one of its neighbors, it defers from issuing a patch request until the current one is resolved or a timeout occurs.

We have not evaluated this protocol extension. Adding it to the protocol and testing it via simulation and/or implementation is an area of future work.

4.3 Summary

In this chapter, we proposed methods of adaptive sleep and listening intervals for power save protocols. In Section 4.1, we propose an adaptive listening technique where a node adjusts the time that spend checking for wake-up signals in response to transmissions in its neighborhood. This is especially beneficial when only a few advertisement packets need to be sent in each listening interval. In this case, nodes who are not the recipient of a wake-up signal can return to sleep much sooner.

In Section 4.2, we present an adaptive sleeping technique that allows nodes to adjust their sleeping interval in response to the desired latency of data that it is forwarding. We find that our proposed protocol allows end-to-end latency bounds to be achieved with much less energy consumption than turning power save off. The protocol can maintain a desired latency bound with only a slight increase in energy consumption over traditional power save protocols.

Chapter 5

Adaptive Broadcast Dissemination Framework

In Chapter 3 and Chapter 4, we have proposed energy-efficient designs primarily for unicast traffic. In this chapter, we look at power save with respect to broadcast traffic. Multihop broadcast is used in many wireless network applications. Some common uses of multihop broadcast include discovering routing paths, sinks querying sensors for data, and distributing code updates throughout the network.

With respect to broadcast, power save protocols generally expose two options to the user. First, if no power save is used, then the broadcast can achieve a relatively low latency, but at the expense of large energy costs to listen for broadcasts. The second option is to use the power save protocol. This option conserves much less energy than the first, but has a high latency that may be unacceptable to some applications.

In our work, we propose a lightweight protocol to augment existing protocols that allows broadcast propagation to be more energy efficient while still achieving a desired latency. In this chapter, we present PBBF, Probability-Based Broadcast Forwarding, and explore the resulting energy-latency-reliability trade-offs that emerge. Additionally, we describe our implementation of PBBF on sensor hardware in TinyOS [11] in Section 5.3.

5.1 Protocol Description

We propose Probability-Based Broadcast Forwarding (PBBF) that can be used in conjunction with any power save protocol that has the following characteristics:

1. Nodes are scheduled to sleep at certain times.
2. A mechanism exists which ensures that all of a node's neighbors will be awake at the same time to receive a broadcast.

While we focus on a synchronous power save protocol in this chapter, asynchronous and out-of-band protocols with these characteristics could also use PBBF. For example, in an out-of-band protocol, nodes could be scheduled to sleep between epochs when they sample the out-of-band channel for a wake-up signal. Then, for broadcasts, a node could wake up all of its neighbors by transmitting a wake-up signal long enough that each of its neighbors has time to detect it. For deterministic asynchronous protocols, the nodes would be scheduled to sleep in certain slots and there could be one common slot scheduled at certain times.

We use IEEE 802.11 PSM [7] as the base protocol to demonstrate PBBF. Particularly relevant to this work is the fact that 802.11 PSM, unlike many power save protocols, specifies a protocol for broadcast. Additionally, some protocols designed specifically for sensors, such as S-MAC [45], are similar to IEEE 802.11 and use many of its mechanisms.

The goal of PBBF is to achieve a specified reliability, with high probability, while allowing a wide-range of tradeoffs in energy consumption and latency. Specifically, we focus on two definitions of reliability in this work: (1) the average fraction of nodes that receive a broadcast and (2) the average fraction of broadcasts received by a node.

PBBF introduces two new parameters to a power save protocol: p and q. The first parameter, p, is the probability that a node rebroadcasts a packet in the current active time even though not all neighbors may be awake to receive the broadcast. With probability $(1 - p)$, the node will wait to send the packet according to the power save protocol. The second parameter, q, represents the probability that a node remains on after the active time when it normally would sleep (the length of time that a node remains on is a parameter of the power save protocol being used). With probability $(1 - q)$, the node sleeps as it would in the original power save protocol. Even with these modifications, a node still only rebroadcasts a packet once. In Section 5.3.1, we introduce a third parameter that allows a node to rebroadcast a packet twice for added reliability.

Figure 5.1 shows pseudo-code of changes to any sleep scheduling protocol required for PBBF. The original sleep scheduling protocol is a special case of PBBF with $p = 0$ and $q = 0$. The *always-on* mode (i.e., no active-sleep cycles) can be approximated by setting $p = 1$ and $q = 1$. PBBF may be slightly different from *always-on* in this case. For example, in synchronous protocols, there may still be byte overhead (e.g., sending advertisements) and temporal overhead (i.e., PBBF cannot send data packets during the advertisement window).

Intuitively, we can see that the p and q parameters will have the following effects.

Energy: As q increases, energy consumption increases. Changing p has a negligible effect on energy consumption.

```
SLEEP-DECISION-HANDLER()
1   /* Called at the end of active time */
2   /* If $stayOn$ is true, then remain on; else sleep*/
3  $stayOn \leftarrow$ false
4
5  if $DataToSend = true$ or $DataToRecv = true$
6    then
7          $stayOn \leftarrow$ true
8    else  if UNIFORM-RAND$(0,1) < q$
9            then $stayOn \leftarrow$ true


RECEIVE-BROADCAST($pkt$)
1   /* Called when broadcast packet $pkt$ is received */
2  if UNIFORM-RAND$(0,1) < p$
3    then SEND-BROADCAST($pkt$)
4    else  ENQUEUE($nextPktQueue, pkt$)
```

Figure 5.1: Pseudo-code for PBBF.

Latency: As q increases, latency decreases, provided that $p > 0$. As p increases, latency decreases, provided that $q > 0$.

Reliability: As q increases, reliability increases, provided that $p > 0$. As p increases, reliability decreases, provided that $q < 1$. When p increases, there is a greater probability that a node rebroadcasts the packet immediately. Thus, for a fixed $q < 1$, there is a greater chance that some of its neighbors do not receive the broadcast since they chose to sleep.

If the conditions listed above (e.g., $p > 0$ for latency and reliability as q increases) are not met, then the metric is not affected in that situation. In the subsequent sections, we investigate these interactions more thoroughly.

5.1.1 Design Discussion

Connected Dominating Sets: An alternative method to address the problem of broadcast in energy saving networks is to construct a connected dominating set (CDS) (a.k.a., *virtual backbone*) and allow the selected set of nodes to remain in a high energy consumption state. Previous work [82,119,120] has presented algorithms to approximate the construction of a CDS in wireless networks.

The key aspects of a CDS are that all the nodes in the CDS are connected and all nodes *not* in the CDS are one hop away from a CDS node. Thus, if the CDS consists of high energy consumption, low latency nodes in the always on state, then a broadcast could be unicast along the CDS without incurring any power save delay. Then, each CDS node would incur one power save delay to rebroadcast the packet to all neighbors which may be in a power save state. For example, let us assume that the latency to send a packet among always on nodes is L_1 and the latency to send a packet using power save is L_2, where $L_1 \ll L_2$. If the maximum path length between any two nodes in the CDS is D, then the broadcast will reach all nodes in at most $DL_1 + L_2$ time (assuming that the broadcast initiator does not use power save mode to transmit).

The major difficulty with the CDS approach is that the formation of the CDS is NP-complete so approximation algorithms must be used [82, 119, 120]. Additionally, such an approach does not give fine-grained control over the energy consumption and latency tradeoff. Once the CDS is formed, the energy-latency tradeoff is fixed. By contrast, our design gives the user more fine-grained control of this tradeoff and does *not* require the construction of a distributed structure in the network.

5.2 Simulation Results

In this section, we present simulation results for PBBF. In Section 5.3.3, we discuss TinyOS [11] implementation results for our protocol. We simulated PBBF in two different ways. First, we used a grid topology with ideal MAC and physical layers (i.e., no collisions, packet errors, or MAC delays). This allows us to observe some fundamental aspects of the protocol behavior without second-order effects such as collisions and irregular topologies. The second method of simulation was to test the protocol in *ns-2* with uniformly random topologies. This allows us to explore how PBBF performs in a more realistic setting.

In both sets of simulations, one source that broadcasts periodically. First, we present a brief overview of the results from the ideal simulator. Then, in Section 5.2.1, we highlight some results from *ns-2* simulations. More details can be found in [1].

The metrics that we test are:

- **Joules Consumed/Total Broadcasts Sent at Source:** The sum of the total energy consumed by each node in the network divided by the total number of broadcasts sent by the source.
- **Average Per-Hop Latency:** Calculated by finding the average latency for each node divided by the node's hop count from the broadcast source. These values are then averaged over the entire network. For each broadcast, we only count the latency for nodes that *received* the broadcast.
- **Average X-Hop Latency:** The average latency of nodes that are exactly X hops away from the broadcast source. For each broadcast, we only count the latency for nodes that *received* the broadcast.
- **Fraction of Broadcasts Received:** The fraction of broadcasts received by each node averaged over the entire network.

- **Fraction of Broadcasts Received by 99% of Nodes:** For each broadcast, we determine if it was received by 99% of the nodes in the network. Then, the number of broadcasts that *were* received by 99% of the nodes is divided by the total number of broadcasts sent by the source.

In Figure 5.2, we show how the reliability increases as the values of p and q increase (e.g., PBBF-0.5 refers to PBBF with $p = 0.5$). The key aspect of this figure is that PBBF demonstrates a threshold behavior as is seen with connectivity in percolation theory [121]. In Figure 5.3, we see energy consumption as a function of q. Figure 5.4 shows the latency for various p and q combinations. By combining the data from Figure 5.3 and Figure 5.4, we are able to see the achievable tradeoff between energy consumption and latency for a given reliability in Figure 5.5. This demonstrates the wide-range of energy-latency tradeoffs for broadcast made possible by PBBF.

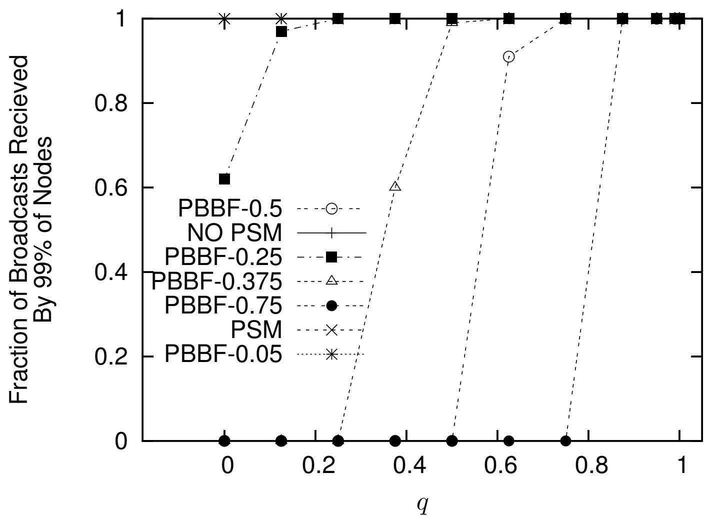

Figure 5.2: Threshold behavior for 99% reliability.

5.2.1 *ns-2* Simulations

We simulated the broadcast application at the routing layer of *ns-2*. One random node is chosen to be the broadcast source for each scenario. Broadcasts are sent deterministically at the source at a rate of 0.01 broadcasts/second. The total size and data payload of each packet are 64 and 30 bytes, respectively. Our scenarios have 50 nodes placed uniformly at random such that the expected number of one-hop neighbors per node is 10. We ran each simulation for 500 seconds and each data point is averaged over 10 runs. In Table 5.1, we give the average and maximum standard deviations for the data points on each curve as a percentage of their mean.

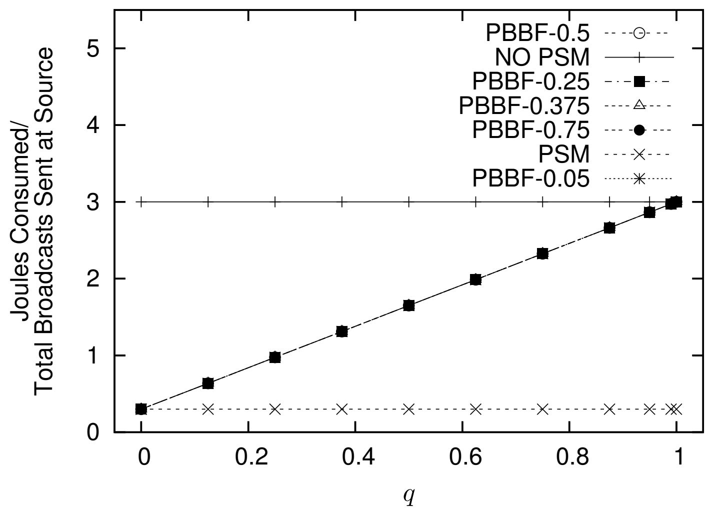

Figure 5.3: Average energy consumption.

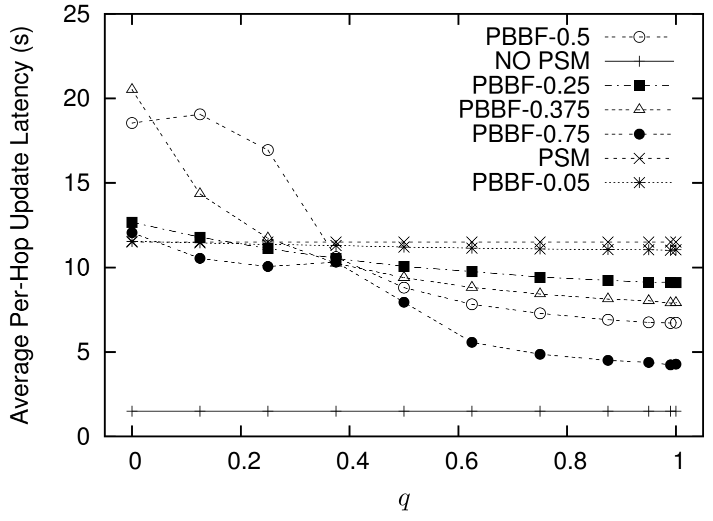

Figure 5.4: Average per-hop broadcast latency.

Table 5.1: Standard deviation as percentage of mean for Section 5.2.1 figures (Average | Maximum).

	Figure 5.6		**Figure 5.7**		**Figure 5.8**		**Figure 5.9**	
PBBF-0.5	1.25	6.80	17.59	32.65	8.47	15.53	2.88	10.59
NO PSM	0.0	0.0	30.69	30.69	11.08	11.08	0.45	0.45
PBBF-0.25	0.59	2.21	11.74	15.85	3.99	6.89	1.03	3.11
PBBF-0.1	0.49	1.93	6.07	9.01	3.16	4.31	0.63	1.55
PSM	1.54	1.54	2.27	2.27	2.20	2.20	0.44	0.44
PBBF-0.05	0.48	1.60	3.47	4.64	2.49	3.99	0.71	1.28

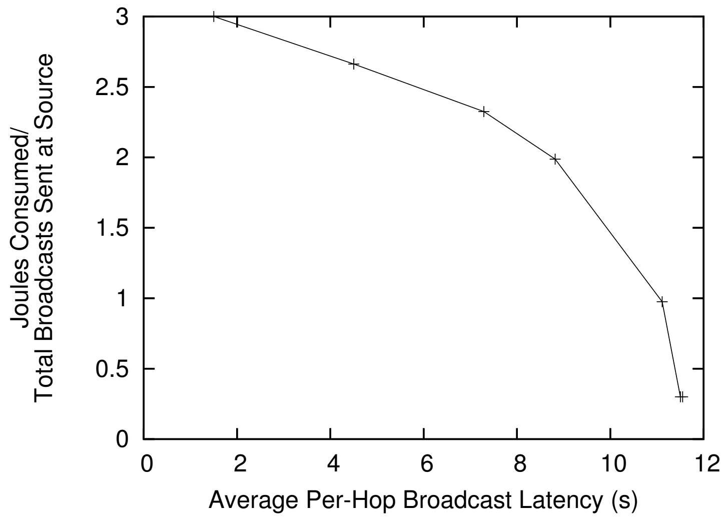

Figure 5.5: Energy-latency tradeoff for 99% reliability.

Our first simulations show how various values of q affect PBBF. Figure 5.6 shows how the average energy consumed at a node, normalized for the number of broadcasts generated, changes with q. We can see that using PSM saves almost 2 Joules per broadcast over using no PSM. The figure also shows that energy consumption increases linearly with the q value. We also observe that q dominates p in the energy usage because regardless of the p value, the PBBF lines almost overlap.

In Figure 5.7 and Figure 5.8, we see the effect of q on latency. Figure 5.7 and Figure 5.8 show the average latency of nodes that are two hops and five hops from the source, respectively. In our simulations, new packets always arrive at the source during the ATIM window, so they are sent with a delay of about AW. As expected, the latency to reach two hop neighbors is about $AW + BI$. We can see that PSM consistently has a high latency, whereas turning off PSM results in a much lower latency. PBBF does worse than PSM at small values of q, but improves significantly as q and p increase. The reason PBBF performs worse for small values of q is the amount of redundancy in broadcasts received from different neighbors is reduced. Therefore, it is more likely that a node will *not* receive the broadcast from the neighbor that would result in the smallest latency. However, as q and p get larger, the probability increases that a broadcast will be transmitted and received without waiting for the next beacon interval. From Figure 5.7 and Figure 5.8, we can also see that the cross-over q point where PBBF does better than PSM occurs at a lower value for nodes farther from the source. This is expected since there is a greater probability that at least one node between the source and a distant node is able to reduce the latency by at least one beacon interval. Whenever the latency is reduced by at least one beacon interval *and* the broadcast is received on a path with the same hop count as PSM, then PBBF does better than PSM in terms of latency. Also, potentially many more different

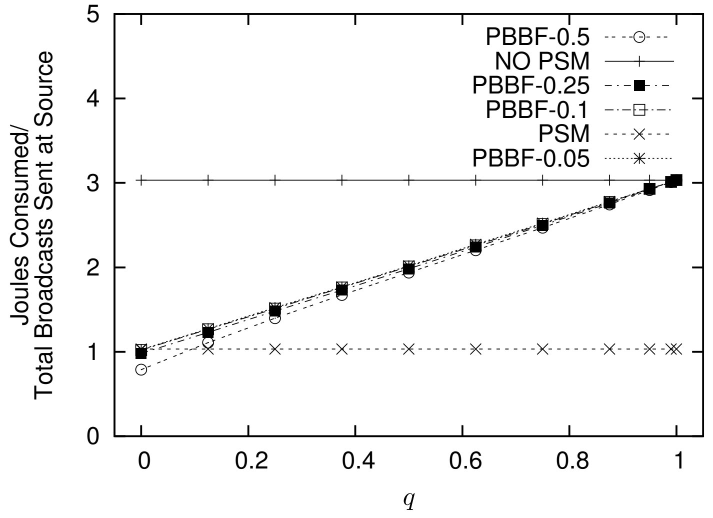

Figure 5.6: Average energy consumption.

paths exist by which the broadcast can reach distant nodes.

Figure 5.9 illustrates how the q value affects the fraction of broadcasts a node receives. We observe that setting $p = 0.5$ results in a significant degradation until q reaches about 0.5. For $p = 0.25$, a little degradation occurs and all the other p values result in less than 1% loss. These results are explained the effect that p and q have on the probability that a broadcast sent by a node is received by a neighbor. We must decrease p and/or increase q until the desired level of reliability is achieved.

5.3 Implementation

We implemented PBBF in TinyOS [11] for the Mica2 Mote [104] sensors. This serves as a proof-of-concept for the protocol and provides results from a real-world communication environment. As described in Section 5.3.2, PBBF is implemented on top of a different sleep scheduling protocol than the 802.11 PSM protocol that was the basis for the simulations in Section 5.2. This demonstrates the versatility of PBBF.

Additionally, we added an extension to the PBBF protocol described in Section 5.1. This extension is described in Section 5.3.1. In Section 5.3.2, we describe the architecture designed in TinyOS. Our experimental results are presented in Section 5.3.3. Section 5.3.4 details some lessons we learned from our implementation experience.

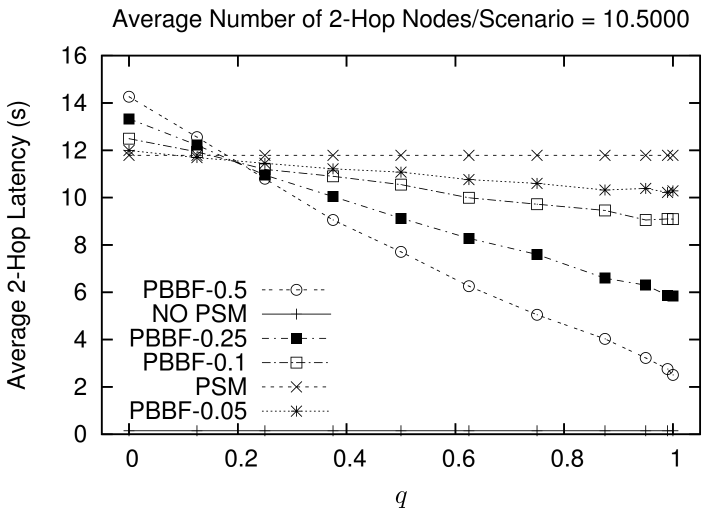

Figure 5.7: 2-hop average broadcast latency.

5.3.1 Protocol Extension

As mentioned in Section 5.1, the PBBF parameters p and q provide a tradeoff in energy consumption, latency, and reliability for broadcast dissemination. Section 5.2 quantifies this tradeoff via simulation.

We propose another parameter that can be used in PBBF that induces an overhead tradeoff in addition to the three aforementioned metrics (i.e., energy consumption, latency, and reliability). We denote this parameter as r and it behaves as follows. When a sensor decides to immediately transmit a broadcast packet according the p parameter (as described in Section 5.1), it will broadcast the packet a second time with probability r. If the packet is broadcast for a second time, then the second transmission is advertised according to the sleep scheduling protocol's original protocol. The pseudo-code for this PBBF extension is shown in Figure 5.10.

We can see that the r parameter induces an overhead tradeoff into PBBF. By increasing r, we increase the reliability of a broadcast at the expense of increasing the packet overhead in the network. At the extreme, if $r = 1$, then reliability should be close to 100% regardless of the p and q values, but each node is broadcasting every packet twice. This gives designers yet another control parameter to achieve a desired tradeoff in the energy consumption, latency, reliability, and overhead planes.

5.3.2 Design

We chose to implement PBBF in TinyOS [11] since this is a widely used open-source operating system designed for sensors. Its adoption in the research community has led to a relatively stable system with a

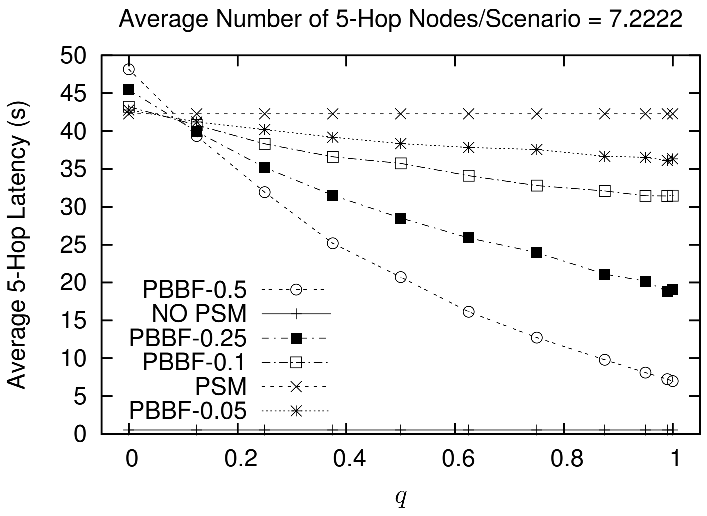

Figure 5.8: 5-hop average broadcast latency.

significant amount of documentation. For a hardware platform, the Mica2 [104] and Telos [122] Motes were available. We chose to use the Mica2 platform since it has two power save protocols implemented for it.

The two power save protocols available on the Mica2 platform were S-MAC [45] and B-MAC [26]. These protocols are both described in Chapter 2. Either would have been an appropriate choice for our PBBF implementation. We chose to use B-MAC over S-MAC for several reasons. First, B-MAC is implemented in the core of TinyOS whereas S-MAC is an add-on that must be incorporated into the TinyOS separately. Thus, B-MAC has undergone more rigorous testing since it is used by nearly everyone that downloads TinyOS and does not require an extra effort to get it working. Second, the code for B-MAC was less complex and easier to understand. Thus, it was easier to make the necessary modifications for PBBF. Finally, B-MAC, unlike S-MAC, does not require time synchronization. This eliminates a major source of potential experimental errors.

As described in Section 2.1.2, B-MAC uses preamble sampling for in-band power saving. Sensors wake up according to a specified duty cycle and carrier sense the channel. If the channel is idle, the sensor returns to sleep until the next scheduled carrier sense period. If the channel is busy, the sensor continues listening to channel in anticipation of receiving a pending data packet. When a node has data to transmit, it attaches a preamble longer than the duty cycle in order to guarantee that all nodes will carrier sense the channel at some point during the preamble and continue listening.

To implement PBBF on B-MAC, we make the following changes:

- When a node carrier senses the channel idle during its duty cycle, with probability q, it continues listening to the channel until its next scheduled carrier sensing period.

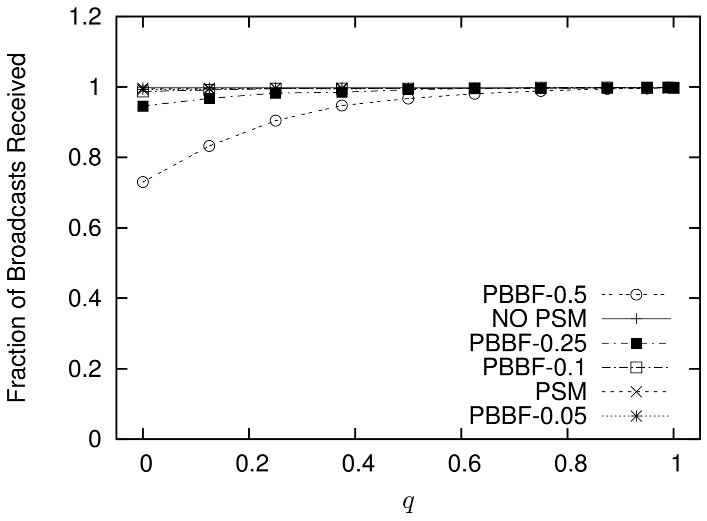

Figure 5.9: Average broadcasts received.

```
RECEIVE-BROADCAST($pkt$)
1   /* Called when broadcast packet $pkt$ is received */
2   if UNIFORM-RAND$(0, 1) < p$
3     then SEND-BROADCAST($pkt$)
4            if UNIFORM-RAND$(0, 1) < r$
5              then ENQUEUE($nextPktQueue, pkt$)
6     else  ENQUEUE($nextPktQueue, pkt$)
```

Figure 5.10: Pseudo-code for r parameter in PBBF.

- When a node has a packet to rebroadcast, with probability p, it transmits the packet without the long preamble. In this situation, most of the node's neighbors will not carrier sense the preamble and, hence, not receive the broadcast packet at that time. With probability $(1 - p)$, the node will rebroadcast the packet with the long preamble so that its neighbors will carrier sense it and receive the subsequent data packet.

- When a node rebroadcasts the packet *without* the long preamble (as discussed in the previous item above), with probability r, it will broadcast the packet a second time. This second broadcast will use the long preamble.

The architecture we used for our implementation is shown in Figure 5.11. The solid arrows in the figure represent the interface that connects two modules. The notation $A \xrightarrow{I} B$ indicates that component B implements interface I and that component A uses B's implementation of interface I. The dashed arrows indicate the message type that the connected module uses to send and/or receive via `GenericComm`. Details

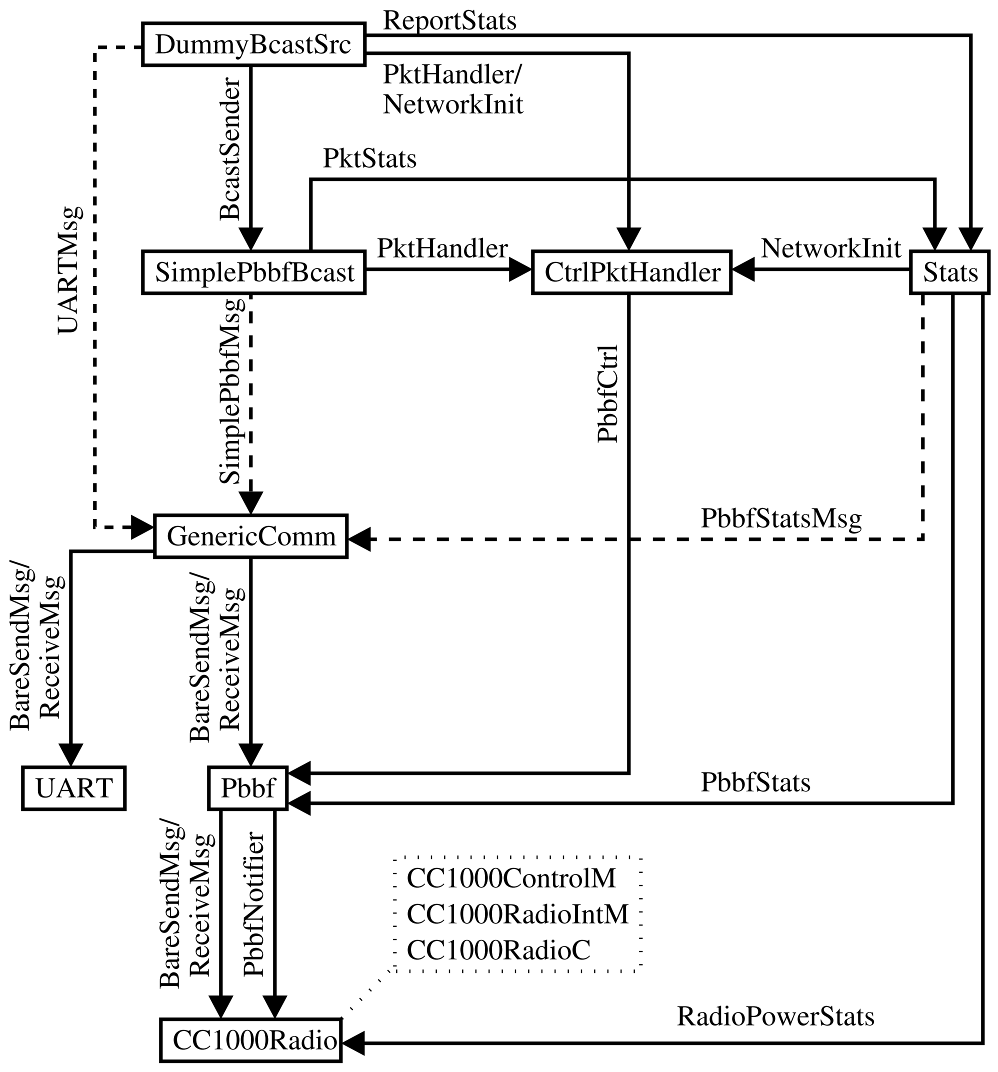

Figure 5.11: TinyOS architecture for PBBF implementation. The solid rectangles are modules (`CC1000Radio` is an abstraction for the three modules listed in the dotted lines). The solid arrows represent the major interface(s) that connect modules. The incoming dashed lines to `GenericComm` represent the message types the connected module uses.

about the interfaces and packet types are in Appendix C. The `GenericComm`, `UART`, and `CC1000Radio`[1] components are already implemented in TinyOS. We made some modifications to the CC1000Radio modules, but used these components, for the most part, in their current TinyOS instantiation. We now describe the functionality of each component from Figure 5.11.

DummyBcastSrc: This is the application that we use to test PBBF. The node with ID 0 is chosen as the broadcast source and transmits a broadcast periodically according to a desired rate. The broadcast

[1]`CC1000Radio` is an abstraction for the three modules listed in the dotted lines.

does not contain any useful data. Non-source nodes that receive broadcast packets pass information to the `Stats` module to collect experimental data.

`DummyBcastSrc` also serves as the link to the serial port (UART) for communication with a computer. The module also passes control packets (e.g., what p, q, and r parameters to use for a particular run) to `CtrlPktHandler`. Finally, it maintains all the timers for when an experimental run ends and when statistics are sent back to the broadcast source.

SimplePbbfBcast: The main functions of this module are duplicate suppression and queuing for `DummyBcastSrc`'s broadcast packets. Control packets are also passed to `CtrlPktHandler` and `Stats` is notified of *every* broadcast sent or received.

CtrlPktHandler: This module handles incoming control packets by setting the p, q, and r parameters to the values specified in the packet. This is done via its connection to `Pbbf`.

Stats: This module keeps track of statistics for our experiments and aggregates the information in packets to send back to the broadcast source. Via `DummyBcastSrc`, it keeps track of the end-to-end latency of received packet as well as the total number of unique, application-layer received packets. `SimplePbbfBcast` informs `Stats` of the total number of data packets sent and received. `Pbbf` signals to `Stats` when a packet was transmitted twice due to the r parameter (as discussed in Section 5.3.1). `CC1000Radio` signals this component whenever the radio switches to and from sleep mode to track the total fraction of time spent sleeping.

This module also provides a basic end-to-end retransmission scheme for added reliability in reporting experimental stats. We found this to be somewhat useful since the link layer retransmission scheme seemed to occasionally fail. One limitation of the link layer retransmissions in TinyOS is that no receiver is specified in the ACK packets. Thus, it is possible that both S_1 and S_2 send a packet to D at about the same time and, for whatever reason, D receives only, say, S_1's packet. However, the ACK send by D, which is intended for S_1 in this example, will also be overheard by S_2 and S_2 will considered its packet successfully received since the ACK does not specify if it is for S_1 or S_2. However, we did not run tests to determine if this was the source of occasional failures for link layer retransmissions since this was a peripheral issue that the end-to-end retransmissions seemed to fix. We mention it, though, as a possibility.

GenericComm: *(Existing TinyOS module)* This serves primarily to multiplex and demultiplex packets in TinyOS based on the packet type. Essentially, the packet type serves the role that ports do in traditional TCP/UDP communications

UART: *(Existing TinyOS module)* This component provides the lower level communication with the serial port.

Pbbf: This is the actual implementation of the PBBF protocol. It is placed between `GenericComm` and the `CC1000Radio` components. `GenericComm` is analogous to the network layer and `CC1000Radio` provides the medium access and the physical layer.

The p, q, and r values that `Pbbf` uses are input from `CtrlPktHandler`. B-MAC notifies `Pbbf` of a decision point for whether to sleep via the `PbbfNotifier` interface. At this point, `Pbbf` compares the current q value to a random number to decide whether to tell the radio to sleep, as would be normal operation, or continue listening to the channel, which is part of PBBF. For every packet received from `GenericComm`, `PBBF` decides, based on the p and r values, whether to use a long preamble and whether to transmit the packet twice, respectively. This layer also provides link layer retransmissions since this feature is not implemented in lower layers (i.e., `CC1000Radio`).

CC1000Radio: *(Existing TinyOS modules)* These components provide the lower level communication with Chipcon's CC1000 radio [123] found on Mica2 Motes. Additionally, the B-MAC [26] implementation is integrated into these components.

5.3.3 Results

To test our implementation, we set up the following experiments. The broadcast source, with ID 0, was attached directly to a laptop via a MIB510CA board. This sensor also served as the sink for reporting statistics back to the laptop. Our Mica2 Motes used the 433 MHz frequency. We were constrained to using only nine Motes total, so the other eight Motes served as broadcast receivers.

Initially, we planned a multihop topology. However, statistics reporting proved far too unreliable for the environment in which we attempted this (see Section 5.3.4 for more details). Thus, we only experimented on a topology where all of the devices were within range of the broadcast source (and each other). This setup was also beneficial since a limited number of Motes were available and PBBF relies on some amount of density to operate efficiently. Furthermore, this simple scenario is sufficient for demonstrating some of the key properties of PBBF.

In our experiments, the source transmitted a broadcast every 2.5 s. Each experiment ran for 30 s, which results in 11 packets being sent per run (the first packet is not sent immediately when the test commences). Each data packet uses the standard TinyOS format with 2 synchronization bytes, 5 header bytes, 2 CRC bytes, and a payload of 29 bytes. The default preamble adds an additional 8 bytes, though, as described in

Section 2.1.2, B-MAC increases the preamble length according to how much power saving is desired. In our tests, we set the B-MAC parameters to have a duty cycle of 135 ms and preamble size of 371 bytes. We note that when a sender decides to transmit immediately, according to the p parameter in PBBF, the preamble size is set to the default 8 bytes for that particular packet. We also note that the version of B-MAC we used carrier senses the channel for 8 ms once every duty cycle. If the channel is not carrier sensed idle, then B-MAC extends the time that it is awake for 32 ms. At the end of this 32 ms interval, B-MAC carrier senses again and will sleep or extend its listening for another 32 ms depending on if the channel is idle or busy, respectively. For statistics collection, once the sensor has run the experiment for the specified 30 s length, it switches power save off for 10 s and reports its data.

The metrics that we measured are:

- **Fraction of Time Not Sleeping:** Obtaining fine-grained energy measurements for the Motes requires special equipment. Thus, we use a coarse-grained metric where we track how much time a node spends with its radio not in the sleep state over the course of an experiment. Thus, the larger the fraction of time not sleeping, the more energy is generally being consumed by the radio.

- **Average Broadcast Latency:** This is the average latency from the time a packet is sent at the sender's application layer until the *data* begins transmission over the radio (i.e., after the preamble and synchronization bytes have been transmitted). For this, we use the time stamping implementation described in [40]. Again, this is not as fine-grained of a metric as we would like. However, this technique obviates the need for time synchronization among the nodes which would induce a large amount of complexity and overhead to our implementation. We only compute the latency for nodes that received a given broadcast.

- **Unique Data Packets Received:** We measure the average fraction of broadcasts sent by the source that are received by listening nodes.

- **Total Data Packets Received:** This is a measure of the receive overhead of the protocol. It is the average total broadcasts received divided by the number of broadcasts sent by the source. Since sensors filter duplicate broadcast packets (with respect to the source and sequence number), the total data packets received is greater than or equal to the unique data packets received.

- **Total Data Packets Sent:** This is a measure of the sending overhead of the protocol. It is the average total broadcasts sent by a node (excluding the broadcast source) divided by the number of broadcasts send by the source.

To test the effects of p, q, and r, we set their values to 0.0, 0.3, 0.7, and 1.0 and ran one experiment (with multiple broadcasts) for each of the 64 possible combinations of these three variables using these four values. For clarity of presentation, we omit some of the combinations in our graphs.

In Figure 5.12, we show the effects of p on energy consumption. When $q = 1$, obviously no energy savings occurs. When $r = 0$, the energy consumption decreases with p because less packets are being received by the nodes (it is proportional to the corresponding curve shown in Figure 5.14). If a sensor is receiving only a few packets, then it will be sleeping most of the time. When $r = 1$, every packet transmitted immediately is also transmitted a second time using the power save protocol. This results in increased overhead and reliability. However, since more packets are being sent and received, the energy consumption is greater than for $r = 0$.

The curve is approximately flat when $q = 0$ and $r = 1$ because the number of transmissions that follow the B-MAC power save protocol remain the same. In B-MAC, such transmissions, with their long preambles, consume significantly more energy than the immediately sent packets. Thus, the dominating energy consumption component, the packets sent using the power save protocol, remains constant. The number of immediate sends increases as p increases, but the effect on the curve is small.

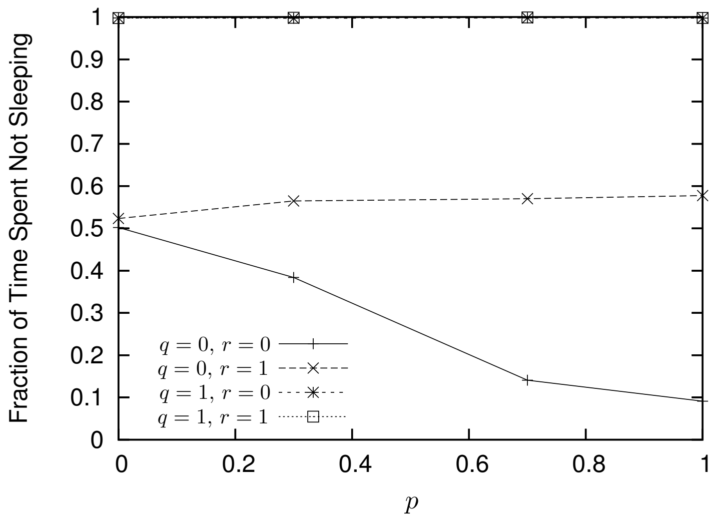

Figure 5.12: Energy consumption.

The effect of p on latency is shown in Figure 5.13. We can see how p improves the latency when $q = 1$. This improvement comes at the expense of energy consumption. The sharp drop-off in the $q = 0$, $r = 0$ case occurs because of a large decrease in reliability (shown in Figure 5.13). This is due to the fact that the latency is only computed for sensors that receive a broadcast. So, the few sensors that happen to receive the broadcast will do so with a small latency when p is high and $q = 0$. As an example, when $p = 0.3$, the reliability of the broadcast is about 80%. However, the slight decrease in latency when $p = 0.7$ comes at the expense of achieving only a 20% reliability. The $q = 0$, $r = 1$ case actually shows an increase in latency because the second packet being transmitted is what is usually being received. The second transmission occurs only after the first transmission that uses a short preamble.

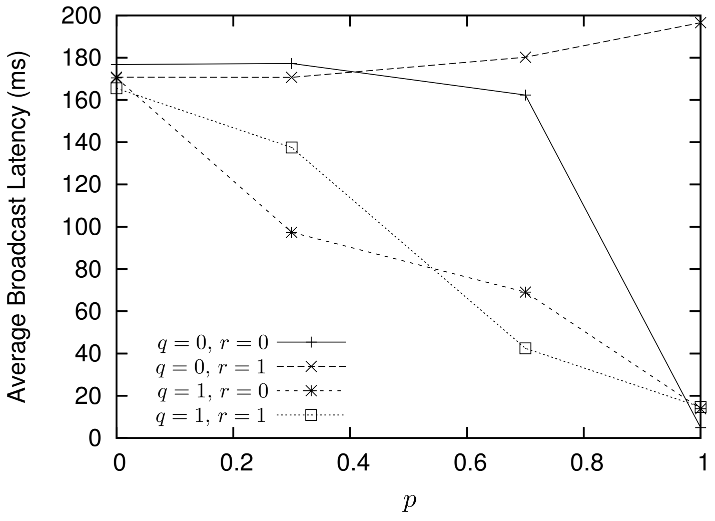

Figure 5.13: Average broadcast latency.

Figure 5.14 shows, as expected, that if $q = 1$ or $r = 1$, the reliability is 100%. However, if both q and r are zero, then the reliability steadily decreases with p to the point of almost 0% reliability.

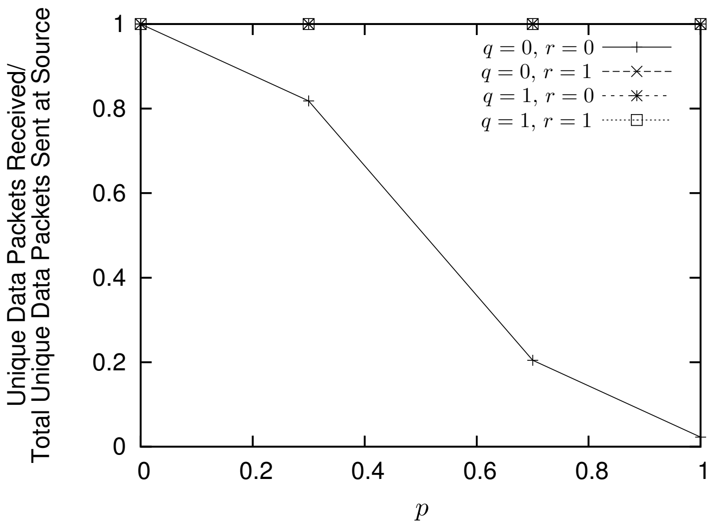

Figure 5.14: Reliability of broadcast.

The overhead for receiving and sending in the protocols can be seen in Figure 5.15 and Figure 5.16, respectively. As expected, when $r = 1$, we get twice the overhead for both sending and receiving when compared to $r = 0$. In both figures, we see that the overhead increases as p increases when $r = 1$ since more packets are sent according to the p parameters and, hence, more packets are sent a second time according to the r parameter. In the case where $q = 0$ and $r = 0$ in these figures, we see a decrease in overhead since more packets are being transmitted according to the p parameter and less neighbors are listening at that time (since $q = 0$).

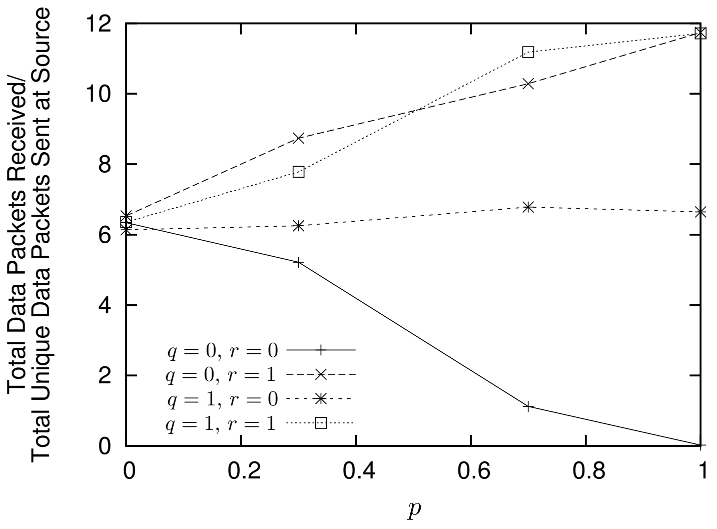

Figure 5.15: Reception overhead.

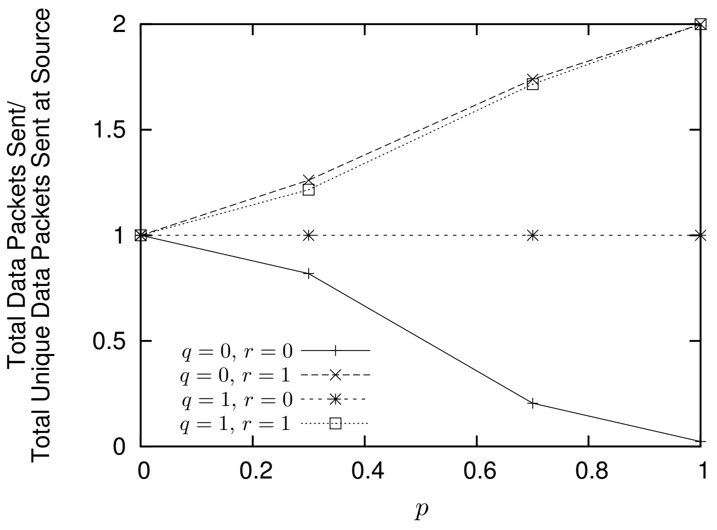

Figure 5.16: Transmission overhead.

Consistent with our results in Section 5.2, Figure 5.17 shows that an increase in q causes a linear increase in energy consumption. The $p = 1$ case uses significantly less energy than the other three cases at lower values of q. This is due to a decreased reliability compared with the other three cases. When no packets are being transmitted using the power save protocol, sensors sleep more since they are receiving fewer packets. This is similar to what was seen in Figure 5.12.

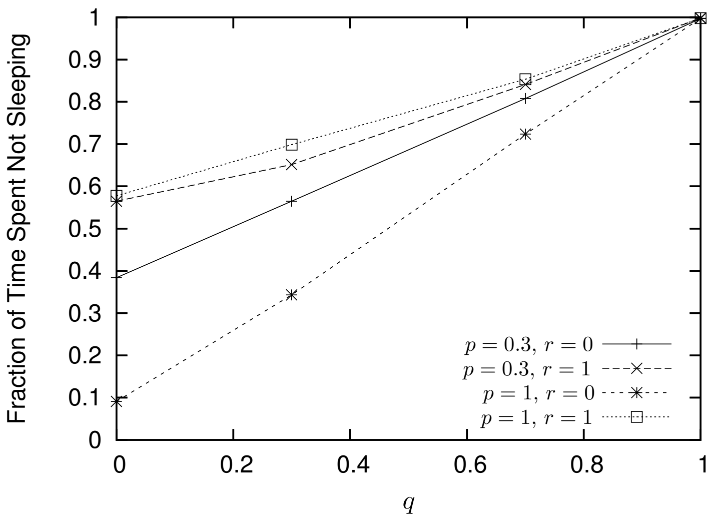

Figure 5.17: Energy consumption.

Figure 5.18 shows the effect of q on latency. In the $p = 1$, $r = 1$ case, the latency shows a linear decrease with q. This is because when $q = 0$, most of the broadcasts received are from the second transmission. However, when $q = 1$, the broadcasts received are from the first transmission instead of the second rebroadcast (as determined by the r parameter). In the $p = 1$, $r = 0$ case, when $q = 0$ the latency is low due to the low reliability (as shown in Figure 5.19). After this point, however, there is a gradual decrease in latency as more broadcasts are received directly from the source rather than rebroadcasts by a neighbor. When $p = 0.3$ and $q = 0$, we can see that $r = 1$ can actually have a *negative effect* when compared with $r = 0$ due to the increased contention from the extra overhead induced.

The fraction of broadcasts that are received is shown in Figure 5.19. The interesting cases are only $p = 1$, $r = 0$ and $p = 0.3$, $r = 0$ since the other two curves have enough redundancy to give 100% reception. In both cases we can see how the reliability improves at q increases (the other two curves are flat at 100% regardless of q's value).

Finally, we tested the effects of r in our implementation. Figure 5.20 shows energy consumption. When $p = 1$, $q = 0$, we can see that the nodes use more energy due to the increase in reliability that the increasing r is providing. The reliability improvement with r is illustrated in Figure 5.21.

Figure 5.22 and Figure 5.23 show the overhead for receptions and transmissions, respectively. These results show that the overhead doubles when $p = 1$ and $q = 1$ as r goes from 0 to 1. This occurs because when $p = 1$, each sensor will transmit each broadcast once when $r = 0$ and twice when $r = 1$. When $p = 0$,

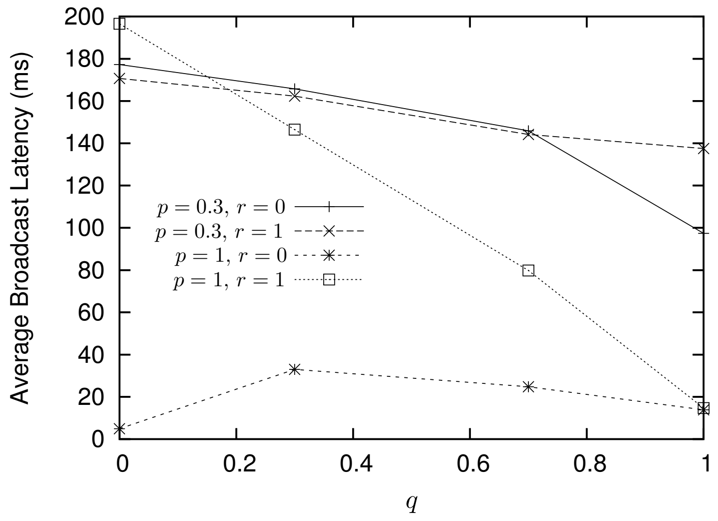

Figure 5.18: Average broadcast latency.

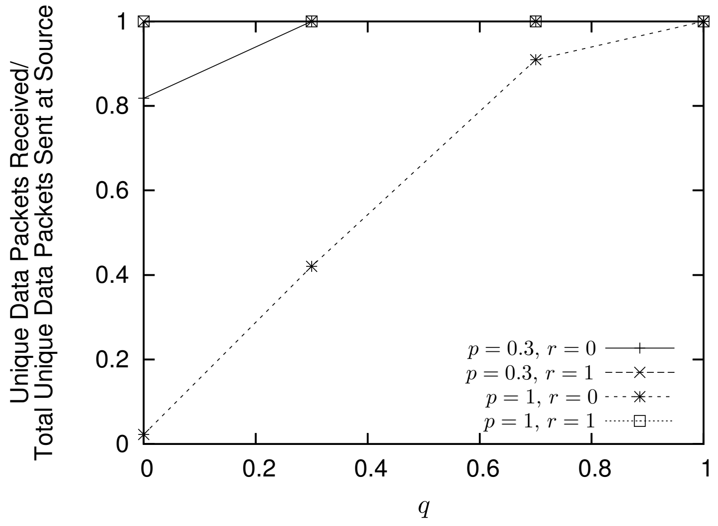

Figure 5.19: Reliability of broadcast (all curves overlap except for $(q = 0, r = 0)$.

we see no effects on the overhead, with respect to r, as expected. When $p = 1$ and $q = 0$, then the overhead is zero when $r = 0$ due to the lack of reliability. The increasing reliability with r causes the overhead to increase linearly.

From these results, we see that our implementation in TinyOS shows the same trends as we observed in simulation for energy consumption, latency, and reliability as a function of the p and q parameters. Additionally, we have shown the effects of a new parameter, r, which can improve reliability at the expense of extra packet overhead. Our figures show the quantitative performance of these three parameters on sensor hardware.

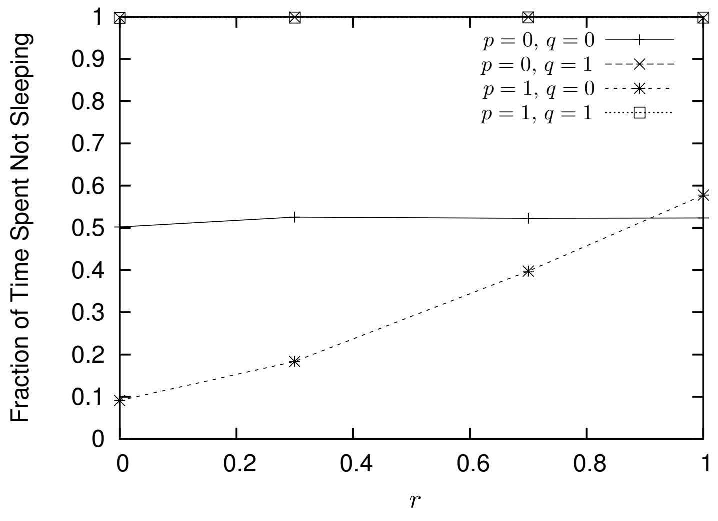

Figure 5.20: Energy consumption.

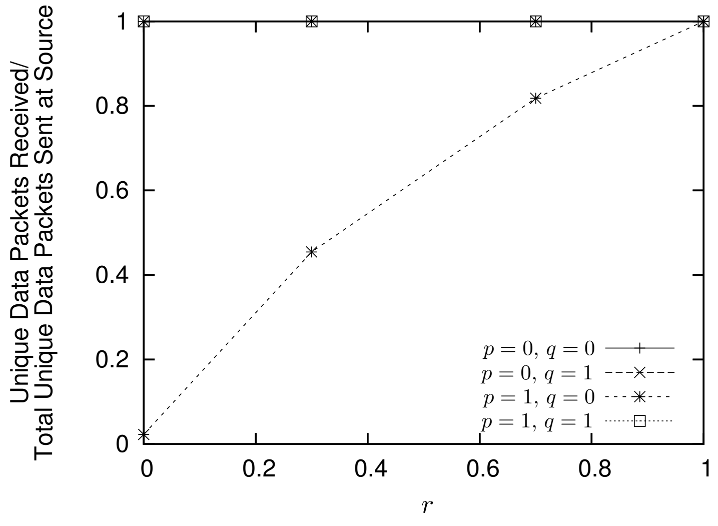

Figure 5.21: Reliability of broadcast.

5.3.4 Lessons Learned

In this section, we list some lessons that we learned as a result of our implementation.

Lesson 1: *A small fraction of seemingly trivial tasks will take a large fraction of your time.*

Getting a reliable serial connection between a Mote and the laptop proved extremely time consuming. The need for root access limited our PC choices to laptops. Most laptops are equipped with only USB ports and not serial ports. However, the USB-to-serial adapters that we tried tended to produce non-deterministic errors where serial communication would succeed about 10-20% of the time. After devoting significant time working under the assumption that the operating system needed configured

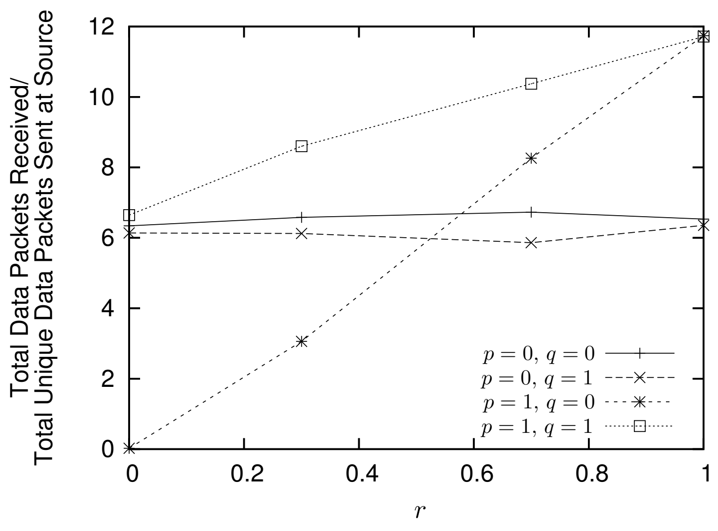

Figure 5.22: Reception overhead.

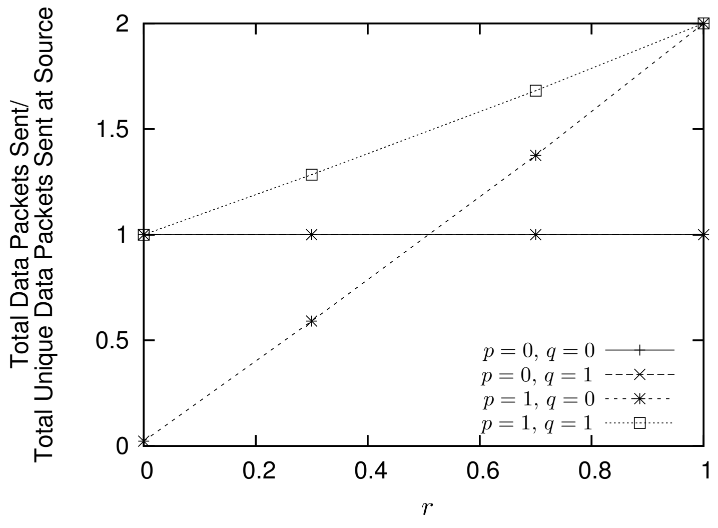

Figure 5.23: Transmission overhead.

correctly, we eventually had to purchase a laptop with a native serial port.

Similarly, some aspects of TinyOS code are poorly documented and commented (though, overall, the documentation is good relative to other open source projects). Thus, some questions that could be answered quickly by someone familiar with the system took a much longer time to figure out by perusing code, documentation, and running applications. Examples include finding that link layer retransmissions were not implemented and discovering how to code them correctly. Another example is determining the differences among the routing protocols provided in TinyOS and discovering how to use these components correctly.

Lesson 2: *Multihop topologies are much more difficult to create than in simulation.*

We found this particularly difficult in an indoor setting. Contrary to our assumption that one could just place devices in a large seminar room spaced by a fixed distance, we discovered that this task requires much more in the way of measurement studies in a specific location to create a topology. Factors such as the height of a device, its distance to other objects, and asymmetry of communication links proved extremely complex in our chosen environment. A much more rigorous measurement study of a location or, better, a specifically designed testing area are needed to create multihop topologies.

Lesson 3: *Statistic collection and software updates are extremely difficult without a wired backplane.*

A significant amount of effort was needed to create a system for collecting statistics that did not interfere with experimental runs. Without an out-of-band channel available, each node had to unicast its statistics for a run back to the sink using the same channel on which the experiments were run.

This was exacerbated by the fact that the experiments required power save protocols to be used, which decreased reliability and increased latency for packets. Our solution was to use local timers with a large "fudge" factor to account for synchronization errors so that all nodes would report their statistics at about the same time and could turn power save mode off while this was being done. Thus, one node's statistics collection was not interfered with by another node's statistics reporting.

Another major difficulty with the lack of a wired backplane is that every time a change is made to the code, each sensor must be manually connected to the laptop and receive the uploaded code. This approach is obviously neither scalable nor desirable during the debugging phase.

Lesson 4: *Debugging is difficult.*

Essentially, the only output available is three LEDs. No `gdb` or `printf` statements can be used when things do not go as planned. The simulator that is available with TinyOS is of some use, but some lower level hardware abstractions are not available (e.g., the time-stamping mechanism) or significantly differ from the Mica2 implementation (e.g., the channel bitrate in the simulator is hard-coded to be twice that of the Mica2 hardware). Again, debugging is an area that would greatly benefit from a wired backplane. Though, even this is made difficult by the fact that every module that needs debugged must be wired to the backplane component and it must create a new packet type to communicate its data.

Lesson 5: *Buffer management is difficult.*

In TinyOS, when an upper layer sends a packet to a lower layer, it is responsible for protecting the memory allocated to that packet until lower layers signal that they are finished handling it. Lower layers are responsible for sending back pointers to packet memory locations when they signal that they are finished. Coding must be done carefully to ensure that upper layers never reuse memory being handled by lower layers and that lower layers return memory pointers consistent with what upper layers are expecting.[2]

5.4 Summary

In this chapter, we have proposed a lightweight protocol, PBBF, that makes lower latency broadcast propagation in power save networks more energy efficient than not using any power save protocol. Using the protocol, a user has more fine-grained control over the energy consumption for a broadcast to achieve a desired latency and reliability. Our simulations show the effectiveness of the protocol and allows us to quantify the energy-latency-reliability tradeoff. Additionally, our implementation in TinyOS [11] demonstrates the protocol on sensor hardware.

[2]For example, an upper layer has two packet queues from which it is sending. The pointers to the packets at the heads of these queues are q_1 and q_2, respectively. Thus, if an upper layer sends the packet at pointer q_1, it would expect that the corresponding signal to indicate that the send is done will return pointer q_1 so that it knows which queue just finished being serviced. If a lower layer erroneously returns a different pointer, the upper layer cannot correlate which queue is being signaled as serviced.

Chapter 6

Leveraging Channel Diversity for Secure Key Distribution

As sensor networks become more ubiquitous, security becomes a major concern. It is also an excellent opportunity for researchers to integrate security in the initial stages of protocol design, rather than an added afterthought as has occurred in many previous network protocols. To this end, an important aspect of sensor network security is key establishment. Using secure keys is the foundation of many other aspects of security such as encryption for confidentiality and message authentication for integrity.

Certain properties of sensor networks make the key establishment problem unique compared with protocols for other types of networks:

- *Sensors are resource constrained:* The sensors are generally assumed to be small devices. This implies that resources such as memory, computation power, and transmission rates are much more constrained than in desktops and laptops. As an example, Mica Motes [25] have a CPU speed of 4 MHz, a few hundred kilobytes of memory, and a bitrate of 19.2 kbps. From a security perspective, this means that symmetric keys are preferable to asymmetric keys and, ideally, only a small portion of memory is devoted to key material [6, 89, 90, 93, 95].

- *Packets are broadcast over the air:* This is true for wireless networks of all types and means that it is much easier for an adversary to tap into a device's communication channel. Thus, it is assumed that an attacker can overhear any packet transmitted in its vicinity.

- *Deployment may be large in scale:* Networks may be on the order of thousands of sensors. Thus, it is preferable to localize key establishment as much as possible.

- *Topology may be uncontrolled:* In some cases, it is envisaged that sensors may, for example, be tossed out of an airplane to monitor an area. In such a case, there is no advance knowledge of which sensors will be neighbors in the network. Thus, it is conceivable that a device is equally likely to be neighbors with any of the N sensors being deployed. If a sensor wishes to share a secret key with each of its neighbors, it needs to store $N - 1$ in memory, creating a scalability problem given the memory size of the sensors and the potentially large scale of the network.

- *Deployment may be in hostile territory:* Sensors may be used to monitor enemy areas, therefore it is possible that an adversary is able to populate the network with a significant number of its own devices.
- *Planned incremental additions may be desired:* Sensor networks may be long-lived and require the owner to deploy new sensors as older ones fail (whether maliciously or due to battery exhaustion). Additionally, it may be desirable for an owner to occasionally "upgrade" the network by increasing the density of the sensors to get better sensing coverage. In this work, we assume that incremental additions are relatively rare events that can be planned reasonably well in advance.

Given these properties, we design a key establishment protocol based on symmetric cryptography that can scale to hundreds or thousands of sensors without any prior knowledge of sensor locations and demonstrate resilience to the threat model described in Section 6.1.2. In this chapter, we focus on distributing pairwise keys among one-hop neighbors in a sensor network. Pairwise keys are important in sensor networks for a couple of reasons. First, when sensors are sending their data to a sink, it allows secure aggregation of data at each hop since a sensor shares a secret key with each of its children as well as its parent. Second, pairwise keys can be used to authenticate a hash chain commitment that can then be used to do, for example, authenticated broadcasts [124].

The design space for pairwise key distribution lies between two extremes. On one end, each sensor could be loaded with $N - 1$ keys before deployment such that it shares a secret key with every other sensor in the network. This scheme offers the most security since no information about the keys is ever broadcast and a compromised sensor gives an attacker no information about keys being used by other sensor pairs. However, this scheme suffers greatly from a scalability viewpoint since a sensor may need to store thousands of keys, of which it probably uses only a small subset.

At the other extreme, each sensor is given one key which it shares with every other sensor in the network. From a scalability viewpoint, this scheme is excellent since a sensor stores only one key regardless of the size of the network. However, this scheme offers little resilience to an attacker since if one device is compromised, the communication between every pair of sensors is also compromised.

The major contributions of this work are as follows. First, we present a distributed protocol that requires a small amount of memory for storage and, with high probability, ensures each link in the network shares a unique key. Through analysis, we show the properties of our protocol and demonstrate that it is feasible within the resource constraints of current sensor hardware.

Second, we show that diversity of sensor channels and location can be a benefit to sensor network security. While such diversity has been widely used to improve performance in wireless networks (e.g., increasing spatial reuse, decreasing bit errors), to our knowledge, this is the first work to apply these concepts to symmetric key establishment. In particular, we show that the location diversity of randomly deployed sensors greatly improves resilience to adversarial hardware that is deployed in the same manner. It is also demonstrated that using only *one* extra channel during initialization (i.e., using two channels instead of one) greatly improves security. Finally, we characterize the tradeoff in security and energy that is possible when power saving is used.

6.1 Background

6.1.1 System Model

We assume that the sensors are deployed such that their locations are distributed in a desired area. In our model, the network is relatively dense (e.g., more than ten one-hop neighbors per sensor) so a sensor can overhear multiple neighbors.

Our analysis also assumes a radio model that can be represented by unit disks and that links are symmetric; so if A can hear B, then B can also hear A. We assume that the radio can communicate on multiple, non-interfering channels, but can listen to or transmit on only a single channel at any given time. For example, devices such as Mica Mote sensors [25] and any 802.11 compliant hardware [7] can use frequency-division multiple access (FDMA) to achieve this.

6.1.2 Threat Model

In this work, an attacker's primary objective is to learn the link key that a legitimate pair of sensors is using for communication. If the attacker is able to learn this key, then the encryption and authentication for the link is no longer secure. As in previous work [6, 89, 90, 93, 95], we consider denial-of-service to be beyond the scope of this work. See [125] for a discussion of possible solutions to address such attacks in sensor networks.

The attacker has two means by which it tries to learn link keys. The first is by compromising legitimate sensors and learning all the key material that is stored on the device. In this case, obviously all communication with the compromised sensor becomes insecure. However, learning the key material of the compromised sensor may also assist the attacker in learning the link keys being used by non-compromised sensors. In this work, we assume that attackers can compromise sensors *any* time after deployment. We note that this is a stronger attack model than is assumed in some key establishment protocols [97] in which an attacker can compromise sensors only *after* some initialization time.

The second method by which an attacker may try to learn link keys is listening to the plaintext keys exchanged during the initialization phase as discussed in Section 6.2. Since all the information needed to create link keys is broadcast in plaintext at some point during the initialization, the attacker may be able to reconstruct link keys by eavesdropping.

As in [6], we assume that the hardware that an attacker deploys is similar to the legitimate sensor hardware in the network. Thus, any radio hardware an attacker uses has a receive threshold equal to or larger than that of the sensors in the network. That is, for a packet transmitted at a certain power level, the attacker's radio cannot receive a packet if it is farther away than the distance that sensors can receive the packet. As another result of this assumption, the attacker cannot execute wormhole attacks whereby colluding devices can propagate data across the network via an out-of-band channel.

We do not investigate coordinated channel assignment and switching protocols that an adversary may use with multiple radio devices or colluding single radio devices. Such scenarios are an area for future work. However, we *do* consider the effects of an attacker adding more devices that collude by combining their knowledge of overheard packets.

6.1.3 Bloom Filters [4]

In our protocol, we use Bloom filters [4] to communicate sets of keys. In this section, we give a brief overview of the operation of Bloom filters. For more information, the reader is encouraged to peruse any of the numerous tutorials or surveys available on the subject (e.g., [126]).

Each sensor is preloaded with the same h one-way hash functions [127], $H^1_{BF}, H^2_{BF}, \ldots, H^h_{BF}$.[1] Given an input, each of these hash functions maps an object to a value between 1 and s according to a uniform distribution. The Bloom filter is a bit vector of s bits. Initially, every bit is set to 0. Given an object, v_i, it is placed in the filter by setting the bits $H^1_{BF}(v_i), H^2_{BF}(v_i), \ldots, H^h_{BF}(v_i)$ to 1. Thus, for each object (say, $i = 1$ to n) added to the filter, up to h bits are set to 1. After receiving a Bloom filter, a sensor can check to see if an object, v_j, is in the filter by testing if the bits $H^1_{BF}(v_j), H^2_{BF}(v_j), \ldots, H^h_{BF}(v_j)$ are all set to 1. If this test is true, then the object is considered to be in the filter. False positives occur when each of the h values for an object, v_j, maps to a bit that was set to 1 by the hash function for some object other than v_j.

[1]A one-way function, H, is easy to compute (i.e., given object x, $H(x)$ can be computed in polynomial time), but hard to invert (i.e., given $H(x)$, no polynomial time algorithm exists to find y such that $H(y) = H(x)$).

In 6.3, we investigate what values of h and s are appropriate for our scheme to avoid false positives with high probability. However, our scheme is robust against occasional false positives as described in Section 6.2.5.

6.1.4 Merkle Trees [5]

Another cryptographic primitive used in our protocol is Merkle trees [5]. We use Merkle trees to provide authentication that the set of keys being broadcast by a sensor was generated by a trusted source before deployment.

Assume that a trusted source has m objects that it wishes to distribute among untrusted sensors. The goal of a Merkle tree is that when a sensor claims to have a certain object, the value of that object can be authenticated without contacting the trusted source. To do this, the trusted source generates the Merkle tree for the objects before deployment and then loads each sensor with the root of the tree as well as the $\lg m$ interior nodes of the tree needed to verify the authenticity of each object distributed to the sensor.

Figure 6.1, gives an example of a Merkle tree for four objects: v_1, v_2, v_3, v_4. First, each leaf node of the tree is generated by hashing one of the objects. For example, $C = H_M(v_1)$, where H_M is a one-way hash function [127]. Each of the interior nodes of the tree is generated by hashing a concatenation of the node's left and right child. For example, $B = H_M(E||F)$. This continues until the root, R, is generated.

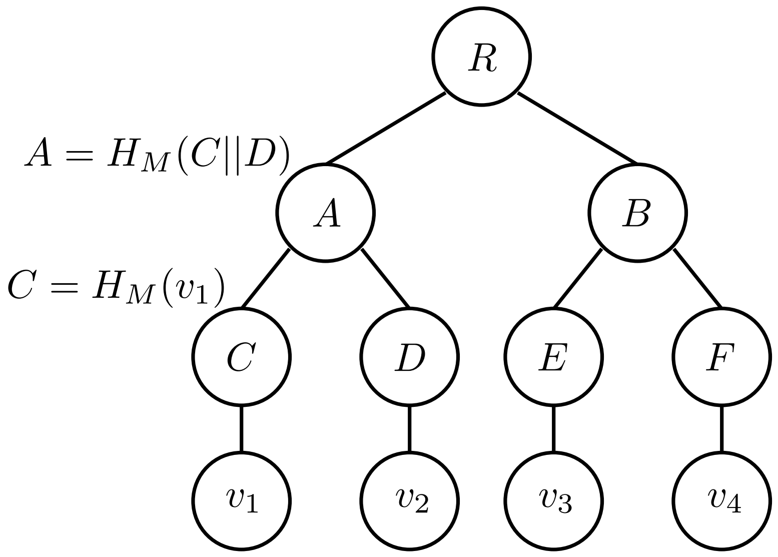

Figure 6.1: An example Merkle tree [5] for four objects: v_1, v_2, v_3, v_4. The leaf nodes are generated by doing a one-way hash, H_M, on the corresponding object. The interior nodes are generated by doing a one-way hash, H_M, on a concatenation of the children of a node.

Each sensor can verify the authenticity of objects by being loaded with R by the trusted source. As an example, consider a sensor that is given object v_2 by the trusted source. The sensor is then also loaded with the values necessary to authenticate v_2 (i.e., C and B). When the sensor wishes to verify the authenticity of v_2, it can do so by transmitting v_2 along with the values of C and B. With these values, any sensor that knows R can verify the authenticity of v_2 by checking that $R = H_M(H_M(C||H_M(v_2))||B)$.

6.2 Protocol Description

6.2.1 Overview

We begin with a brief overview of the details that are presented in Section 6.2.2 through 6.2.4. Table 6.1 provides a key for the notation used in this section.

Table 6.1: Protocol notation.

Notation	Description
c	Number of non-interfering channels available
α	Number of keys broadcast by each sensor during initialization
λ	Maximum number of advertisement keys from any one of a sensor's neighbors
γ	Cumulative number of keys a sensor includes in its advertisement from neighbors
η	Minimum number of advertised keys a sensor must share with a neighboring sensor to engage in communication
h	Number of hash functions per Bloom filter
s	Size of the Bloom filter (bits)

Prior to deployment, each sensor is loaded with a unique, non-overlapping set of α keys by a trusted source. Unlike previous work [89, 90], this set of keys is known to only that sensor and *not* part of a larger shared pool of keys. Along with these keys, the sensors are loaded with the Merkle tree nodes necessary for others to authenticate the Bloom filter of their α keys (as discussed in Section 6.2.2, the actual keys could be authenticated rather than the Bloom filter at the cost of increased overhead).

The sensors are then deployed non-deterministically over a given area. On a common channel, each sensor broadcasts the Bloom filter of the α keys with which it was loaded along with the Merkle values necessary to authenticate the filter. This allows sensors to verify that future keys received from a given neighbor were given to that neighbor by the trusted source. In Section 6.2.2, we discuss this aspect of the protocol in depth as well as how to deal with attacking devices that try to rebroadcast legitimate keys and generate arbitrary keys.

During initialization, sensors switch their radios to a channel chosen uniformly at random and choose a non-deterministic amount of time to listen to the channel before switching to another channel. Also during this time, the sensors choose non-deterministic times to broadcast each of their α keys in plaintext. The key broadcast times are independent of the channel switching times. This procedure continues until all sensors have had a chance to broadcast all of their keys. Each key is sent on the channel to which the sensor is listening at the chosen broadcast time. During this initialization phase, sensors store every key that they transmit as well as every key that they overhear in broadcasts by their neighbors.

At the end of the initialization phase, all sensors switch their radio to a common channel, and perform a key discovery phase during which a sensor tries to establish a unique key to communicate with each neighbor. For the discovery phase, each sensor hashes all its known keys into a Bloom filter and broadcasts the filter. Every time a sensor overhears a filter, it searches the Bloom filter of its own keys to determine which keys it has in common with the sender of the overheard filter.

After the key discovery phase, the key establishment phase is done. During key establishment, a sensor broadcasts a separate message for each filter it overheard in the key discovery phase. In this message, the sensor includes a Bloom filter indicating the keys it believes it has in common with the sender of the original Bloom filter along with a random nonce encrypted by a link key composed of those shared keys. If the sensor receives a single acknowledgment with a properly encrypted, incremented nonce value, a link key has been established with that neighbor. This process continues until a sensor has a unique key for each of its neighbors.

At this point, with high probability, the sensor shares a secret key with each of it neighbors and needs to store only its link keys, α preloaded keys, and Merkle nodes for authentication [128].

6.2.2 Predeployment Phase

We now describe how keys for sensors in the initial deployment are loaded onto each device. In Section 6.6.1, we discuss how this phase of the protocol can be extended to allow incremental sensor additions after this initial deployment.

A trusted authority generates α keys per sensor that are loaded on each sensor before deployment. The keys generated for each sensor are unique to that sensor, and *not* part of a global key pool as has been done in previous work [89, 90] (i.e., a sensor's set of keys do not overlap with another sensor's set of keys). Once the key sets are generated for the sensors, the trusted source computes the Bloom filter for each sensor's set of keys as described in Section 6.1.3. Finally, the trusted source creates a Merkle tree, described in Section 6.1.4, as follows. The leaves of this Merkle tree are generated by hashing a concatenation of the sensor's ID with the Bloom filter of its keys (discussed in Section 6.2.2), denoted as BF.[2] Thus, there is one

[2]Alternatively, the leaves of the Merkle tree could be a hash of *each of the sensor's keys* instead of the Bloom filter of the keys. This would increase the protocol's storage and communication overhead, but eliminates the possibility of false positives and increases robustness against attacks that generate arbitrary keys. Our work does not explore this approach.

Merkle leaf for each sensor its value for that sensor is $H_M(ID||H_M(BF))$.[3]

At this point, each sensor is loaded with the α keys that the trusted source generated for that device, the root of the Merkle tree, and the $\lg N$ interior Merkle nodes needed to authenticate the sensor's Bloom filter of its keys, where N is the number of sensors deployed. After deployment, every sensor listens to a common channel. During this period, each sensor broadcasts the Bloom filter of its α preloaded keys along with the $\lg N$ Merkle values needed to authenticate its Bloom filter and ID. Every sensor stores the ID and Bloom filters for each of its neighbors for the duration of the key distribution procedure.

When the key broadcasting begins, a sensor only accepts a key from a neighbor if the key exists in the Bloom filter associated with the neighbor's ID. This scheme is vulnerable to two attacks. First, an attacker may try to assume another sensor's identity by rebroadcasting the packets containing a legitimate sensor's ID, Bloom filter, and Merkle values and then rebroadcasting keys that it hears the legitimate sensor broadcast during key initialization. In Section 6.2.4, we explain why an attacker could benefit from such a strategy. However, such an attack can be *detected* if any legitimate sensor overhears a key broadcast twice claiming to be from the same ID since each key is to be broadcast only once. Alternatively, the malicious device can be detected if the sensor that is the victim of identity theft overhears the attacker using its keys and ID. Such detection will happen with high probability in relatively dense sensor networks. It is an area of future work to quantify the optimal strategy for an attacker to try rebroadcasting legitimate packets while remaining undetected.

The second type of attack is for the malicious devices to generate arbitrary keys that hash to all 1's in a legitimate sensor's Bloom filter. In 6.3, we analyze the effort required by an attacker to generate these arbitrary keys and see that somewhere on the order of tens to hundreds of arbitrary keys must be generated, on average, for an attacker to create such a key. Also, increasing the size of the Bloom filter can greatly increase the effort required to generate arbitrary keys.

Another way to combat such an attack is to keep track of the number of keys received from any one neighbor and alert an authority when the number of keys received exceeds some threshold that may indicate misbehavior. In applications that require greater immunity to this attack at the expense of storage and communication overheard, each *key* could be placed as a leaf in the Merkle tree, resulting in $\alpha \times N$ leaves for the tree instead of only N leaves. This would allow each broadcasted key to be authenticated individually.

[3]If a sensor's post-deployment location is known *prior* to deployment, then the techniques from [129] can be used to reduce the amount of communication required to verify that a sensor's leaf is legitimate.

6.2.3 Initialization Phase

The goal of the initialization phase is to ensure that each pair of neighbors knows a unique subset of keys at the end of the process. The length of the initialization process depends on how large α must be to achieve the desired probability that all links have a unique set of keys and the amount of channel contention. We assume that broadcasts are sent using CSMA/CA to reduce collisions. In Section 6.3, we analyze what values of α are appropriate for a deployment.

Neighbor pairs are expected to share a unique subset of keys because of the channel and spatial diversity in the network. The primary form of diversity is from the channel switching of the protocol. The only constraints we make on the channel switching algorithm are: (1) each of a sensor's α keys are broadcast during the initialization phase, (2) that each broadcast from a sensor is, on average, overheard by a subset of d/c neighbors, where d is the expected number of one-hop neighbors of the sensor and c is the number of channels available, and (3) a different subset of neighbors overhears each of a sensor's broadcasts, with high probability. Thus, channel selection gives diversity in the subset of neighbors that overhears each of a sensor's key broadcasts.

This is illustrated in Figure 6.2 where E is a neighbor of A, B, and C. We refer to a key that is known by both A and B as a *shared key*. In Section 6.2.4 and Section 6.2.5, we elaborate on how the shared keys are used to generate a *link key* for the sensor pair. The link key is the secret symmetric key that A and B use for secure communication. When C broadcasts a key, the probability that A and B are both listening to the channel on which C broadcasts is $(1/c)^2$. Given that A and B are listening to the same channel on which C broadcasts, the probability that E is *not* listening to that channel is $1 - (1/c)$. When this occurs, A and B have a shared key that can keep the link secure against eavesdropping by E.

The spatial diversity comes from the fact that two neighbors, say A and B, may overhear broadcasts from different sets of common neighbors. Consider the scenario in Figure 6.2 where A and B are one-hop neighbors that are both within range of C and D. However, C and D are *not* within range of each other. Thus, for example, if one of A and B's shared keys comes from a broadcast by C, then D will not know that key and, hence, A and B's link can use the overheard key from C to secure the link from being compromised by D. Similarly, if one of A and B's shared keys comes from D's broadcast, then the link can be made secure against eavesdropping by C. Therefore, over the course of several broadcasts, the link between A and B is expected to eventually be secure against eavesdropping by both C and D with high probability.

Another form of diversity from the wireless channel comes from uncorrelated packet loss. Thus, even if A, B, and E all listen to the channel on which C broadcasts a key, E may receive a packet in error that A and B receive correctly. In this case, A and B learn a key that E does not know. Packet loss can occur, for

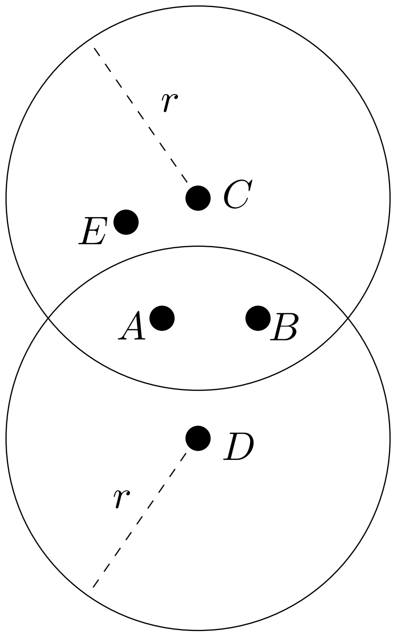

Figure 6.2: One-hop neighbors A and B can both overhear broadcasts from C and D. However, C cannot overhear broadcasts from D and vice versa. E is a one-hop neighbor of A, B, and C.

example, due to noise and interference degrading the received signal.

If two sensors happen to end up close to each other, our protocol still provides some security from diversity. Two cases need considered. The first is when two sensors are close enough to have the same set of one-hop neighbors, but still far enough apart to have uncorrelated packet loss. In this case, the sensors may still receive a different set of keys since they are switching to different channels *and* experiencing different packet losses. The second case is when the sensors are within the coherence distance [130] of each other. In this case, the diversity in key sets still exists since the sensors are switching to a different set of channels.

6.2.4 Key Discovery Phase

When the initialization phase completes after the specified time, to determine a link key, sensors must discover which keys are known by its neighbors. To reduce communication overheard, we use Bloom filters [4] to advertise key sets in a compact manner.

A sensor creates a Bloom filter for advertising its keys by including each of the α keys that it transmitted *and* some of the keys it overheard from the broadcasts of others. We denote the set of keys that a sensor u overheard from other neighbors as K_u. In Section 6.2.6, we discuss how to deal with an attacker who sends out a filter packet that spoofs the address of a legitimate sensor (e.g., claiming the source of the filter is sensor u when it is not). Sensor u then chooses a key subset of size γ to advertise in its filter (i.e., γ is a predetermined constant that, with high probability, is less than $|K_u|$). Since the sensor knows the source of each of its overheard keys, an upper limit, λ, is placed on how many keys in this advertised subset come from any one neighbor. This avoids giving disproportionate influence to any one neighbor in establishing link keys. The sensor creates its advertisement by placing each of these $\alpha + \gamma$ keys into an s-bit Bloom filter. Thus, to place key k in the filter, a sensor sets the bits $H^1_{BF}(k), H^2_{BF}(k), \ldots, H^h_{BF}(k)$ to one. A sensor also computes and stores the hash values associated with *each* of the keys that it overheard or transmitted during the initialization phase so that it can check its key hashes for inclusion in the Bloom filters of its neighbors. Using CSMA/CA the sensors broadcast their s-bit filters.

Upon receiving a Bloom filter from one of its neighbors, a sensor checks to determine which subset of keys it shares with the sender of the filter. Assume that sensor u overhears a Bloom filter advertisement from sensor v. Node u checks to see which of its known keys are included in v's filter. For each key for which all h associated bits are set in the filter, u adds the key to the list of keys it potentially shares with v. Because Bloom filters are susceptible to false positives (but not false negatives), u may add a key to the list that is unknown to v. In Section 6.2.5, we describe how this list of keys is verified. After a filter has been received from each of u's neighbors, it has a list of keys that it believes to be shared with each neighbor. For security, we can require that a sensor only participates in the key establishment phase with neighbors that share some minimum number of keys, η, with the sensor. η can be determined based on the expected number of keys a link should share such that, with high probability, their subset of shared keys is unique. We note that η is similar to the q value in [90].

Now, u tries to determine a unique subset of keys that it shares with each neighbor and creates a link key that will be used for future communication. The set of keys that sensor u believes it shares with sensor v is denoted as K_{uv}. To try creating a shared key with v, u chooses a subset of K_{uv} of size η. It creates the link key, k_{uv}, as the hash of these η shared keys: $k_{uv} = hash(k_1||k_2||\ldots||k_\eta)$, where $k_1, k_2, \ldots k_\eta \in K_{uv}$. Node u will try to verify k_{uv} during the key establishment phase described in Section 6.2.5.

If a sensor determines that it does not share at least η keys with some neighbor, then a few options exist. First, it can choose not to use the link. In a dense network, not being able to use a few links may be acceptable. Alternatively, the sensors can request that the sensors do another initialization procedure at a later time to try adding the links,[4] though such a technique would require some authentication mechanism for the request to avoid attackers spuriously sending such requests. Another option is to use one of the multipath reinforcement mechanisms described in [90] if enough trust exists among its neighbors to do so.[5]

[4]If another link key initialization procedure occurs, sensors that established link keys previously do not try to re-establish a new link key.

[5]Multipath reinforcement [90] allows a sensor pair, A and B, that do not share η keys to use m shared neighbors, $nbr_1, \ldots, nbr_m$ to establish a key if A and B both already share η with each of these m neighbors.

6.2.5 Key Establishment Phase

The final phase of the protocol is the key establishment phase. At this point, each sensor knows of a subset of keys that it believes it shares with each of its neighbors (sometimes this belief may be erroneous as discussed below). Furthermore, as we show in Section 6.4, with high probability, this subset of keys is unique to that sensor pair. Each sensor has a list of unique filters received in the key discovery phase. We refer to this as the *filter list*. In key establishment phase, a sensor, u, challenges its neighbor, v, with the key k_{uv} that it generated in the previous phase. Specifically, u sends a random nonce, RN, encrypted by key k_{uv} to v along with a Bloom filter containing the η keys that compose k_{uv}. We refer to this packet as a Link Request ($LREQ$). Thus, $LREQ = (v||u||\{RN\}_{k_{uv}}||BF(k_{uv}))$, where $BF(k_{uv})$ is a Bloom filter containing the hashes of the keys that compose k_{uv}. We note that the size of $BF(k_{uv})$ should be much smaller than the Bloom filters send during the advertisement phase since, typically, $\eta \ll \alpha + \gamma$. In Section 6.2.6, we discuss how to deal with an attacker who sends out a $LREQ$ packet that spoofs the address of a legitimate sensor (e.g., claiming the source of the $LREQ$ is sensor u when it is not).

When v receives the $LREQ$, it searches K_v to determine if it believes it knows every key in $BF(k_{uv})$. If v can decrypt the $LREQ$ correctly using the keys that u used to compose k_{uv},[6] it will reply with a Link Reply packet ($LREP$). The form of this packet is $LREP = (u||v||\{RN+1\}_{k_{uv}})$. Once u receives the $LREP$ and verifies that the incremented value of RN is correct, the link key is established and, with high probability, k_{uv} is known only to u and v.

It is possible that v is not able to decode u's $LREQ$ correctly for one of two reasons. First, there could have been a false positive when u determined its shared keys from v's advertisement and it used the key that caused this false positive to compose k_{uv}. The second reason is that a false positive occurred when v determined the keys that composed k_{uv} from the Bloom filter in u's $LREQ$. If one of these two event occur, then v can reply with its own $LREQ$ to u using some different subset of keys that it believes the two sensors share. If the sensors are unable to agree on a shared key after some specified number of $LREQ$ exchanges, then can resort to one of the methods described in the previous section (i.e., do not establish the link, request another link key initialization procedure, or use multipath reinforcement [90]).

6.2.6 Neighbor Address Authentication

We note that an attack could occur during the key discovery or key establishment phase whereby an attacker spoofs the identity of another node in the filter, $LREQ$, or $LREP$ packet. A malicious device could initiate

[6]We assume well-known fields are in the encrypted part of the $LREQ$ so that a sensor can verify that it correctly decrypted the random nonce.

a Sybil attack [92] by spoofing multiple addresses in this manner. In this case, a legitimate node, v, may receive two different packets claiming to be from a neighbor, u. One of the packets is actually from sensor u while the other one is spoofed by the adversary. In this case, v has no way of identifying which set of known keys are actually associated with u. If the v chooses the wrong set, it may end up establishing a pairwise key with the malicious device thinking that it is sensor u.

To guard against this attack, we propose using Merkle trees and one-way hash chains for a challenge-response system. Each sensor is loaded with the root of another Merkle tree (i.e., separate than the key authentication tree discussed in Section 6.2.2). The leaves of this Merkle tree consist of hashing the concatenation of the sensor's ID with a hash chain commitment (described below) for the sensor. Thus, the leaf for sensor u is $H_M(u||HC_u)$, where HC_u is the chain commitment for u. The sensor is also loaded with the $\lg N$ values necessary to authenticate the leaf with its ID and hash chain commitment, as discussed in Section 6.1.4.

The hash chain works as follows. The commitment of the chain is obtained by applying multiple one-way hashes to an initial value known to only the sensor associated with the commitment. For example, suppose that RN_u is a random nonce known to only u initially. In this case, $HC_u = H_c(H_c(\ldots H_c(H_c(RN_u))\ldots))$, where H_c is a one-way hash function for generating hash chains. Thus, if we denote RN_u as v_0, then the i-th value ($i > 0$) of the hash chain, v_i, is $v_i = H_c(v_{i-1})$. Thus, for a chain of length l, $HC_u = v_{l-1}$.

The hash chain is then revealed, as needed, in order starting with the value preceding HC in the chain. That is, the first time a sensor needs to authenticate itself using the chain, it will send the value v_{l-2}. Recall that $HC = v_{l-1} = H_c(v_{l-2})$. Thus, since H_c is a one-way function, it is computationally infeasible for any sensor other than the owner of the hash chain to derive v_{l-2} prior to its initial broadcast. Additionally, any sensor than knows the HC for the broadcasting node can verify v_{l-2} by testing that $HC = H_c(v_{l-2})$. Choosing a sufficiently long value for the hash chain represents a tradeoff in the number of challenges that a device can respond to on one hand and the computation required to compute/verify chain values on the other hand. We note that hash chains have been used for symmetric key authentication in other wireless networks applications [131, 132].

Each device broadcasts it hash chain commitment value along with the $\lg N$ Merkle tree nodes to authenticate the commitment to its neighbors. When the authentication of a filter packet, $LREQ$, or $LREP$ is needed, a device will reveal the next value on its hash chain. If the most recent value that the sensor broadcasted was v_i, then it will next broadcast v_{i-1}. A sensor can pessimistically authenticate each filter, $LREQ$, or $LREP$ packet that it sends, or do so only in response to a challenge by a neighbor. Such a challenge could come, for example, in response to a sensor receiving two different packets claiming to be from node u, or if it receives a single packet which it suspects may spoofed.

Alternatively, instead of using hash chains, we could use another two-level Merkle tree as described in Section 6.6.1. In this approach, each sensor would have k leaves in its Merkle tree and each leaf would be the hash of the sensor's ID with one of k random nonces. To authenticate in this scheme, the sensor would include a previously undisclosed random nonce and broadcast the Merkle tree nodes necessary for authentication.

We note that such an approach is not infallible. In particular, an attacker could learn the necessary authentication values for a sensor, u, in one part of the network and transmit them out-of-band to another part of the network for a malicious device to assume u's identity. Such an attack is beyond the scope of our work. Another shortcoming is when packet loss occurs, an adversary may rebroadcast a legitimate node's most recent hash chain value. A sensor that did not receive the legitimate node's original disclosure of the hash chain value, but did receive the adversary's rebroadcast of the value will consider the adversary's packet authenticated. So, while this approach is not perfect, it does provide another level of security that makes address spoofing more difficult.

6.3 Analysis

In this section, we present analysis, followed by simulation in Section 6.4. We use the notation in Table 6.1 and Table 6.2. This analysis is approximate and only considers a limited case (i.e., where only nodes within range of each other are considered and the attacker is passive) in order to see the trends that arise. We consider two sensors, u and v, that have d_{uv} common neighbors (d_{uv} excludes u and v) that are *not* within range of an eavesdropper, w. We are interested in the probability that the link key generated by u and v is known by w in the system. Thus, our analysis does not consider multiple, colluding adversary devices. Nor does it consider an attacker that is broadcasting its own set of α keys during the initialization phase. However, the simulation in Section 6.4, addresses these scenarios.

In our analysis, w has d_{uvw} neighbors that are shared by u, v, and w (d_{uvw} excludes u, v, and w). Thus, the total number of shared neighbors for u and v is $d_{uv} + d_{uvw}$. In our analysis, we consider only sensors u, v, and the d_{uv} and d_{uvw} shared neighbors of u and v. In a real network, there may be many sensors within range of one of the sensors (i.e., u or v), but not the other one; this is not captured by our analysis. In Section 6.4, we simulate the protocol in a more realistic setting. We then derive the probability that u and v share at least one key in their key establishment phase that is unknown to w (and, hence, the link is secure against eavesdropping by w). In the analysis, we assume an ideal MAC layer with no collisions. In Section 6.4, we simulate the protocol using a more realistic MAC layer.

Table 6.2: Analysis notation.

Notation	**Description**
d_{uv}	Shared one-hop neighbors of sensors u and v (excluding u and v) that are not within range of w
d_{uvw}	Shared one-hop neighbors of sensors u and v that are also shared by sensor w (excluding u, v, and w)
p_e	Probability of a packet loss
p_h	Probability a sensor in overhears a key that is broadcast by one of neighbors
X_{uv}	Random variable indicating the number of keys known to both u and v that originated from either u or v's α keys.
$X_{d_{uv}}$	Random variable indicating the number of keys known to both u and v that originated from their d_{uv} neighbors.
$X_{d_{uvw}}$	Random variable indicating the number of keys known to both u and v that originated from their d_{uvw} neighbors.
E_{comp}	Event that u and v's link is compromised by w (i.e., w overheard *all* of u and v's η keys).
E_{conn}	Event that u and v's link is connected (i.e., u and v share at least η keys).
f	False positive rate for Bloom filter
p_a	Probability an attacker creates an arbitrary key that will be accepted as a legitimate key

The probability that a sensor hears a key broadcast by one of its neighbors is: $p_h = \frac{1}{c}(1 - p_e)$, where c is the number of channels and p_e is the packet loss probability. Now, we look at the probability a link is connected and not compromised for given values of α and η. We do not consider the effects of λ or γ in our analysis (refer to Table 6.1 for parameter meanings). Instead, we assume that a sensor advertises all its keys and places no maximum on the number of keys included from any one neighbor (i.e., λ). These assumptions simplify the analysis. In Section 6.4, we incorporate the γ parameter to avoid having arbitrarily large advertisement packets.

Recall that η is the number of keys that u and v must share to have a link. Thus, we first determine the probability with which u and v share this many keys. We partition the keys known by both u and v into three sets: (1) those keys that came from either u or v's set of α keys (indicated by the random variable X_{uv}), (2) those keys that came from their d_{uv} neighbors that are *not* within range of w (indicated by the random variable $X_{d_{uv}}$), and (3) those keys that came from their d_{uvw} neighbors that *is* within range of w (indicated by the random variable $X_{d_{uvw}}$). We now present equations of the probability density functions for these three random variables.

Each random variable has a binomial distribution [133]. For X_{uv}, there are a maximum of 2α keys possible and each one is in the shared set with probability p_h. This is because the originator of the key (either u or v) knows that key with probability 1 and the non-originator will have heard the key with probability p_h.

Thus, we have:

$$\Pr[X_{uv} = x] = \binom{2\alpha}{x} p_h^x (1 - p_h)^{2\alpha - x} \quad \text{, when } 0 \le x \le 2\alpha \tag{6.1}$$

The expressions for $X_{d_{uv}}$ and $X_{d_{uvw}}$ are similar to Equation 6.1 but have different parameters. The maximum number of keys that can come from these sets are $d_{uv}\alpha$ and $d_{uvw}\alpha$, respectively. The probability (for both $X_{d_{uv}}$ and $X_{d_{uvw}}$) that a key is in u and v's shared set of keys is p_h^2 since both u and v overhear such a key with probability p_h. This gives us:

$$\Pr[X_{d_{uv}} = x] = \binom{d_{uv}\alpha}{x} p_h^{2x} (1 - p_h^2)^{d_{uv}\alpha - x} \quad \text{, when } 0 \le x \le d_{uv}\alpha \tag{6.2}$$

$$\Pr[X_{d_{uvw}} = x] = \binom{d_{uvw}\alpha}{x} p_h^{2x} (1 - p_h^2)^{d_{uvw}\alpha - x} \quad \text{, when } 0 \le x \le d_{uvw}\alpha \tag{6.3}$$

Now, we can derive the probability that u and v are connected, or $\Pr[E_{conn}]$ (i.e., they share η keys overheard during the initialization phase).

$$\begin{aligned} \Pr[E_{conn}] &= \Pr[X_{uv} + X_{d_{uv}} + X_{d_{uvw}} \ge \eta] \\ &= \sum_{i+j+k \ge \eta} \Pr[X_{uv} = i] \cdot \Pr[X_{d_{uv}} = j] \cdot \Pr[X_{d_{uvw}} = k] \end{aligned} \tag{6.4}$$

Next, we find the probability that a link is compromised given that it is connected (i.e., $\Pr[E_{comp}|E_{conn}]$). For this, we note that if u and v share even one key from a common neighbor that is not w's neighbor, then w cannot compromise this link since, by design, such neighbors are are not overheard by w. Therefore, we need to sum over the event space where $X_{d_{uv}} = 0$ and $(X_{uv} + X_{d_{uvw}}) \ge \eta$.

$$\Pr[E_{comp}|E_{conn}] = \left(\Pr[X_{d_{uv}} = 0] \cdot \sum_{i=0}^{2\alpha} \sum_{j=\max(\eta-i,0)}^{d_{uvw}\alpha} p_h^i p_h^j \Pr[X_{uv} = i] \cdot \Pr[X_{d_{uvw}} = j] \right) \Big/ \Pr[E_{conn}] \tag{6.5}$$

Given these expressions, we can now compute the probability that u and v's link is connected and not compromised:

$$\Pr[\overline{E_{comp}} \wedge E_{conn}] = (1 - \Pr[E_{comp}|E_{conn}]) \cdot \Pr[E_{conn}] \tag{6.6}$$

Based on these expressions, we generated some numerical results. Due to the $O(\alpha^3)$ complexity of Equation 6.4, we had to make a simplification to achieve an acceptable computation time. The simplification is that we do not consider u and v's keys to be a separate partition of the key space. Instead, we include them in the d_{uvw} set of keys to reduce the algorithmic complexity to $O(\alpha^2)$. Again, since our analysis is approximate, the numerical results only serve to show trends whereas the simulation results in Section 6.4 capture effects that are ignored in our analysis. To generate these numerical results, we set $\eta = 10$, $d_{uv} = 0$, $d_{uvw} = 11$, and $p_e = 0.01$.

Figure 6.3 shows the fraction of links that are connected as a function of α and Figure 6.4 shows the fraction of links the are connected *and* not compromised.[7] We note that the curves in Figure 6.3 and Figure 6.4 are the virtually the same for all but the $c = 1$ case. This means that, for the curves where $c > 1$, a link is secure with high probability given that it is connected. For the $c = 1$ case at lower α values, we can see that the connectivity is approximately one, but a significant fraction of links are compromised. Thus, for the $c = 1$ case, connectivity does not correlate to a link not being compromised as strongly as it does when $c > 1$.

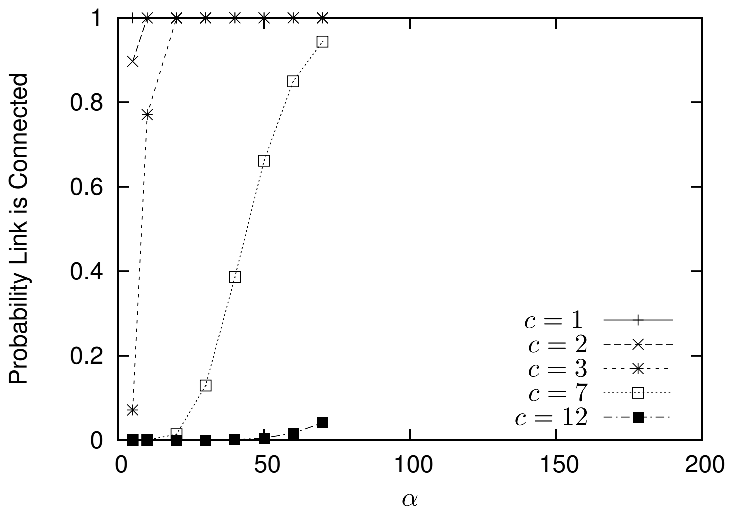

Figure 6.3: Connectivity of legitimate sensors vs. α.

We note that the trends in Figure 6.3 and Figure 6.4 are the same as the corresponding Figure 6.7 and Figure 6.8, respectively, in Section 6.4. However, there are some differences when compared to these simulation results. First, the security of the $c = 1$ case is higher in the analysis than in simulation. This is because in the analysis we only consider one eavesdropper, whereas 30% of the sensors are colluding adversaries in the simulation results. Also, the connectivity of the $c = 2$ and $c = 3$ cases is higher in the analysis. This is because the analysis only considers an ideal MAC layer with no collisions. In the simulation, when the number of channels is relatively low, the contention is higher and, hence, the number of packets

[7]These figures are plotted on the same scale as Figure 6.7 and Figure 6.8, respectively, for purposes of comparison. Due to computation time, we were not able to extend the curves as far in the analytical numerical results.

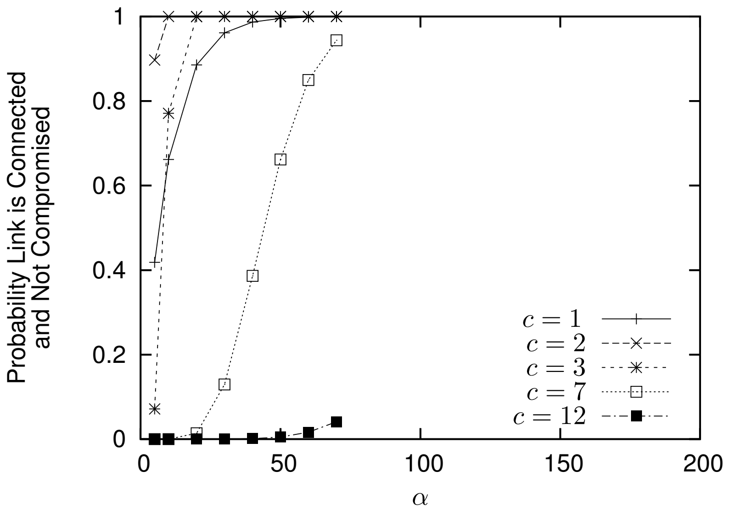

Figure 6.4: Secure connectivity of legitimate sensors vs. α.

a node receives is reduced. Finally, the connectivity for the $c = 7$ and $c = 12$ cases is *lower* in the analysis than in simulation. When the number of channels is higher, contention is less of a factor. However, the fact that we are not treating u and v's keys separately in the numerical analysis reduces the connectivity. This is because the keys from u and v's sets of α keys have a p_h probability of being in the pair's set of η keys whereas in our numerical analysis, such keys only have a p_h^2 probability of being in the shared set of keys.

If $d_{uv} > 0$, then the security of the protocols improves to the point where virtually every connected link is not compromised. This is shown in Figure 6.5, where we use $d_{uv} = 3$ and plot the probability that a link is compromised given that it is connected $(1 - \Pr[\overline{E_{comp}} \wedge E_{conn}]/\Pr[E_{conn}] = \Pr[E_{comp}|E_{conn}])$. We note that for the $c = 1$ case, our tools do not have enough precision and all probabilities are rounded up to one (i.e., $\Pr[\overline{E_{comp}} \wedge E_{conn}] = 1$ and $\Pr[E_{conn}] = 1$, which forces $\Pr[E_{comp}|E_{conn}] = 0$ by our numerical method). Therefore, we do not plot the $c = 1$ curve. We see that, for the cases we can plot with sufficient precision, the conditional probability is at least 0.99999. Thus, when $d_{uv} = 3$, $\Pr[E_{comp}|E_{conn}]$ is virtually zero.

Figure 6.6 shows the fraction of links that are connected and not compromised as a function of the number of channels. This is comparable to Figure 6.10 in the simulation section (both figures have the $\alpha = 25$ and $\alpha = 50$ cases). In the analysis, when compared to the simulation, the connectivity is lower at a higher number of channels and higher at a lower number of channels. The reasons for this are explained in the previous paragraph in relation to Figure 6.4.

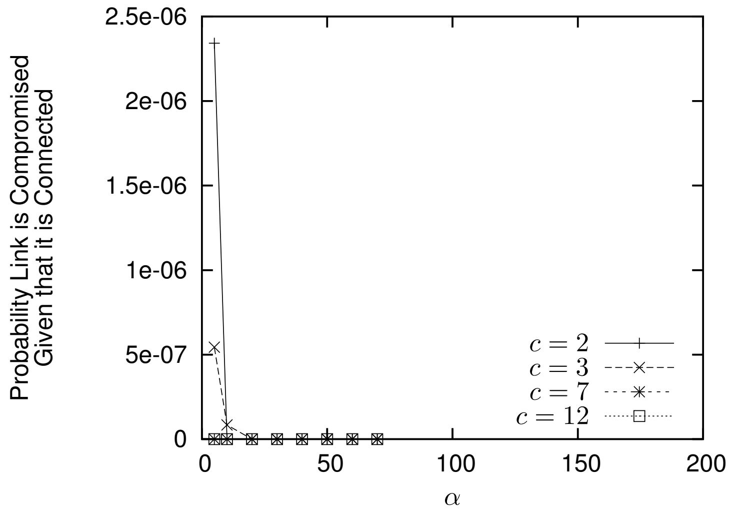

Figure 6.5: Probability of link compromise given the link is connected vs. α for $d_{uv} = 3$. Note that the lack of precision in our numerical method causes the $c = 1$ case (not plotted) to always be zero.

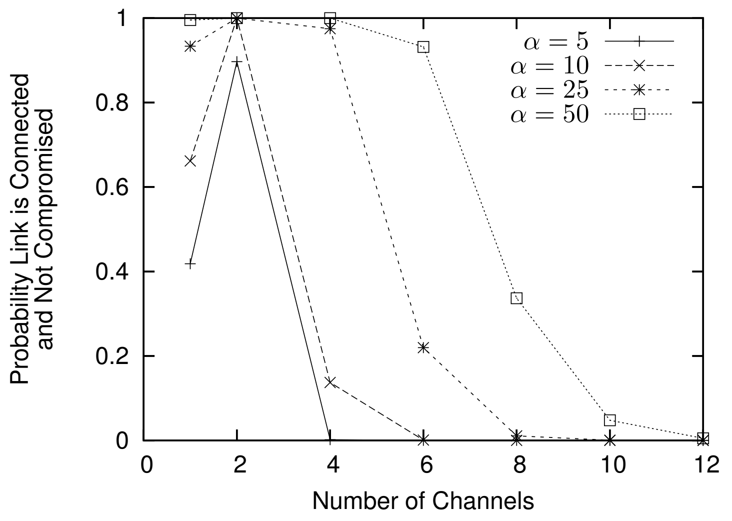

Figure 6.6: Secure connectivity of legitimate sensors vs. the number of channels.

6.4 Simulation Results

We simulated our protocol with *ns-2* [103]. In each test, 50 sensors are placed uniformly at random such that the density of the network (i.e., the expected number of one-hop neighbors per sensor) is 10. Each data point is the average of 30 test runs. The standard deviation for the figures is shown in Table 6.3 and Table 6.4. The default values used in the simulations for the protocol are: $\alpha = 100$, $\gamma = 100$ (i.e., the advertisement size, $\alpha + \gamma$, is 200 keys), and $\eta = 10$ (i.e., a sensor pair must share at least 10 advertised keys for their link to be considered "connected"). We set $\lambda = \alpha$; in future work, we plan to thoroughly investigate the effects of the λ parameter.

Table 6.3: Standard deviation as percentage of mean for Section 6.4 figures with channel curves (Average | Maximum).

	Figure 6.7		**Figure 6.8**		**Figure 6.11**		**Figure 6.12**		**Figure 6.13**	
$c = 1$	0.00	0.00	43.62	45.00	65.38	224.82	0.45	0.91	44.50	65.42
$c = 2$	0.44	3.11	2.40	12.52	9.08	31.44	0.57	1.24	9.73	33.31
$c = 3$	2.91	20.35	3.69	24.45	3.08	13.56	0.68	1.40	7.06	21.36
$c = 7$	6.01	37.03	5.99	37.03	0.51	3.16	0.57	0.86	12.20	60.10
$c = 12$	15.89	81.48	15.89	81.48	0.22	0.78	0.49	0.81	14.20	45.14

Table 6.4: Standard deviation as percentage of mean for Section 6.4 figures with α curves (Average | Maximum).

	Figure 6.9		**Figure 6.10**	
$\alpha = 25$	13.51	50.08	20.29	49.92
$\alpha = 50$	3.29	12.23	9.10	37.72
$\alpha = 75$	1.33	5.94	6.38	33.26
$\alpha = 100$	0.52	2.95	4.872	29.32

To implement the channel switching and key broadcasting discussed in Section 6.2.3, we use the following algorithms. Sensors are assumed to be synchronized and at fixed intervals (we use 2 s in our tests), all of the sensors switch to a new channel uniformly at random. Within each fixed interval, if a sensor has not broadcast all of its α keys, it chooses a time uniformly at random to broadcast *one* of its remaining keys.

When the parameter is fixed, we set 30% of the sensors, chosen uniformly at random, to be controlled by a malicious entity. In our tests, these malicious sensors follow the same protocol as the uncompromised sensors, however, they collude in their knowledge of plaintext keys learned during the initialization procedure. Thus, collusion among malicious devices is global; we do not restrict them to being neighbors to share their knowledge. Thus, the attacker is able to compromise a link between two legitimate sensors if their set of shared keys is found within the union of the sets of keys known by *all* the colluding malicious sensors.

In our tests, we vary α, c (the number of channels), and the percentage of colluding malicious sensors and consider the following metrics:

Connectivity: defined as the fraction of links between legitimate sensors in the network that share at least η advertised keys.

Secure Connectivity: the fraction of links between legitimate sensors that use a key set with at least one key that it unknown to the colluding malicious sensors.

In Figure 6.7, we see how increasing α improves the connectivity of legitimate sensor pairs for different values of c. This is expected since sensor pairs will share more keys when their total number of known keys increases. We also note that the connectivity improves with a smaller number of channels, since the probability of a sensor overhearing a neighbor's broadcast is increased.

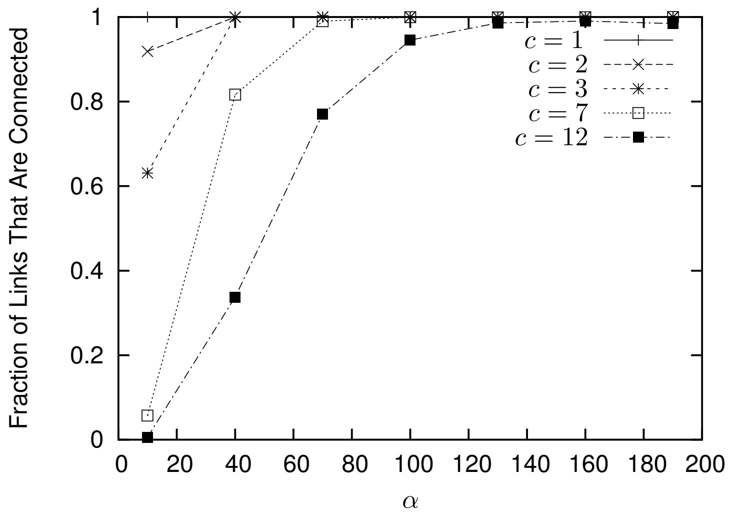

Figure 6.7: Connectivity of legitimate sensors vs. α.

However, as shown in Figure 6.8, using a smaller number of channels is not always good from a security perspective. In Figure 6.7, using $c = 1$ gave a connected topology for all values of α. However, in Figure 6.8, we see that less than half of the connected links for $c = 1$ remain uncompromised by the attacker. All of the other values of $c > 1$ are much more resilient to attacker compromise, though they require a larger value of α for all of the links to be connected. We note that the reason the $c = 1$ case does not converge to 70%, as might be expected since 30% of the sensors are malicious, is due to the high network density. Intuitively, if all sensors were within range of each other, then one would expect virtually all links to be compromised provided at least one attacker is in the network.

In Figure 6.9 and Figure 6.10, we look more closely at the effect of the number of channels on connectivity and security. As expected, the connectivity of the network drops as the number of channels increases as shown in Figure 6.9. The more interesting result is in Figure 6.10, which shows the large benefit obtained from channel diversity. The reason that the $c = 1$ case is lower than the $c = 2$ case is because, despite the high connectivity of the $c = 1$ case, the fraction of secure links for the $c = 1$ case is much lower. Thus, by adding *one* extra channel (i.e., $c = 2$), the protocol's security is greatly increased. We note that the benefit from using multiple channel diversity is not possible in the Anderson's work [6], which uses only one channel to broadcast plaintext keys.

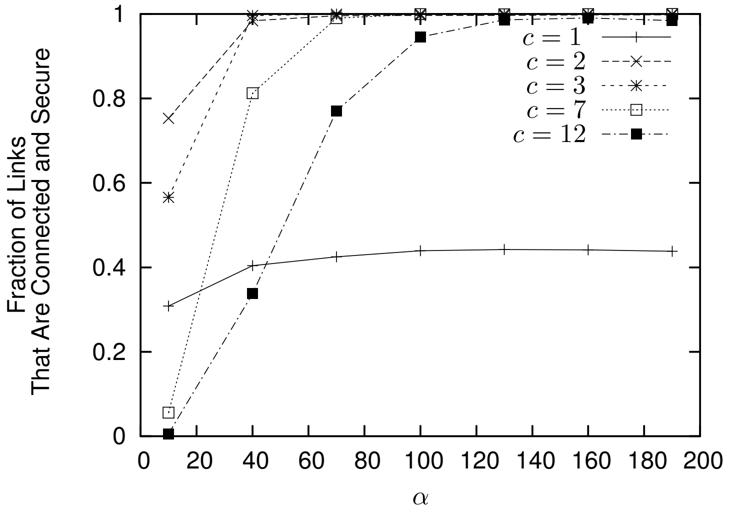

Figure 6.8: Secure connectivity of legitimate sensors vs. α.

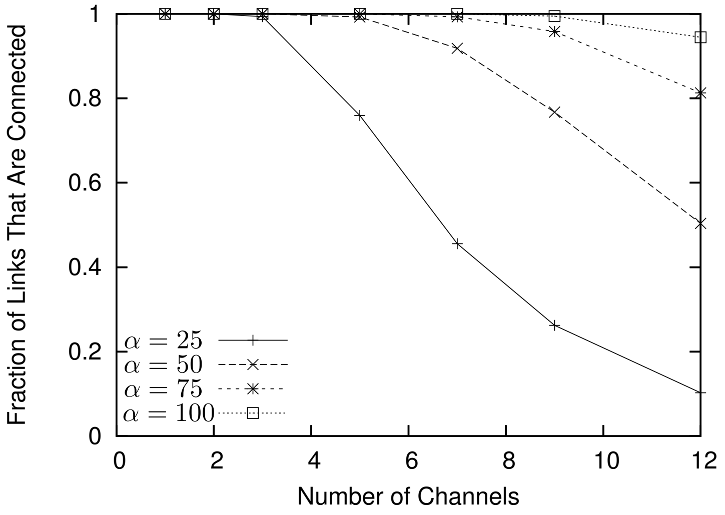

Figure 6.9: Connectivity of legitimate sensors vs. the number of channels.

At this point, it is evident that setting $c = 2$ provides the significant gain in link security (compared with $c = 1$) while maintaining very high network connectivity (compared with larger values of c). Thus, one may wonder what is the utility of setting $c > 2$. To answer this question, we refer the reader to Figure 6.11, which shows the fraction of connected links between legitimate sensors that are secure against the colluding malicious devices as a function of the number of such devices in the network. Though we omit the data, we note that the connectivity is greater that 90% in each of these tests and for $c \leq 7$, the connectivity is greater than 99% for each test. Thus, all the values of c shown in Figure 6.11 provide much higher connectivity than is seen in the results for some other key predistribution schemes (e.g., [89, 90]).

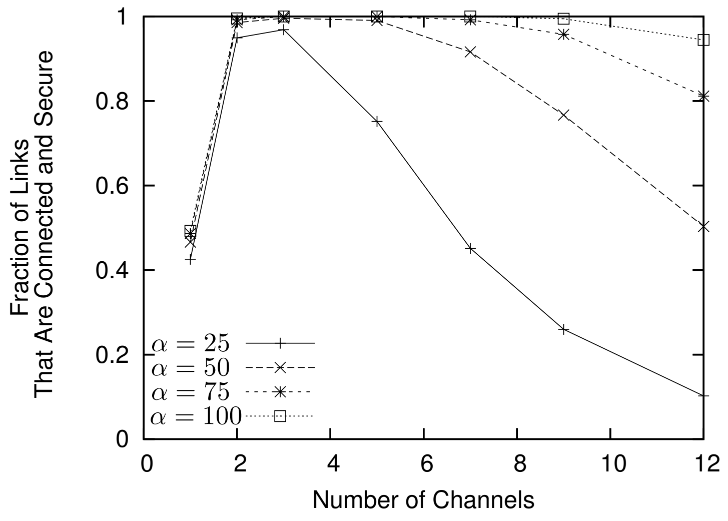

Figure 6.10: Secure connectivity of legitimate sensors vs. the number of channels.

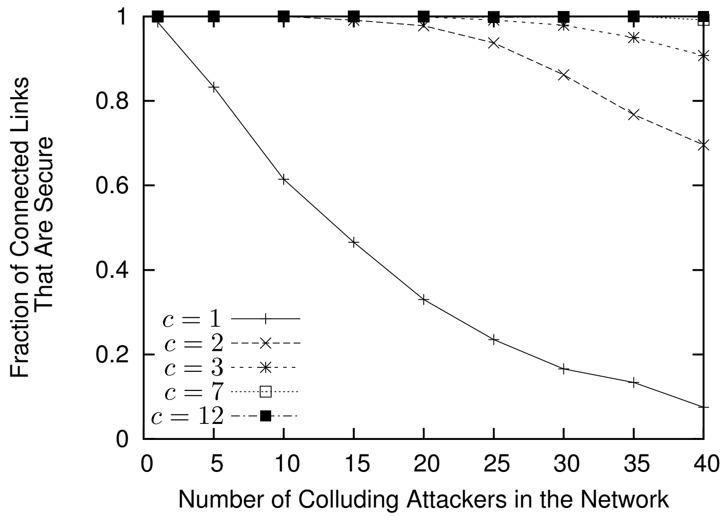

Figure 6.11: Fraction of links between legitimate sensors that are secure vs. the number of colluding attacker sensors.

Figure 6.11 shows that having more channels allows connected links to be more secure against the colluding malicious devices. In particular, for $c = 2$, when about 15 malicious devices are in the network (30% of the sensors), some of the links are no longer secure. For $c = 3$, about 25 malicious devices, 50% of the sensors, are necessary to start compromising some of the legitimate links. For $c = 7$ and $c = 12$, even with 40 malicious devices in the network, virtually all of the links between legitimate sensors are secure. We would like to emphasize that this last scenario corresponds to ***80%*** of the sensors in the network being malicious, which is an extremely hostile setting. Recall that in Anderson's work [6], only up to about ***3%*** of the sensors were malicious.

To test the effect on energy-saving on our protocol, when sensors switch channels once per fixed interval, it also decides to sleep during the remainder of the interval with probability p. By increasing p, we are able to consume less energy during the initialization, however the legitimate sensors will also overhear less keys. The malicious sensors never sleep, so they can still overhear all broadcast keys.

In Figure 6.12, we see that the fraction of the time the legitimate sensors sleep has a linear relationship with their average power consumption, as one would expect. In Figure 6.13, we show how the security of the legitimate links decreases as the time spent sleeping increases. Finally, by combining the data from Figure 6.12 and Figure 6.13, we generate Figure 6.14 to characterize the effects of energy-saving on the security of the protocol. This figure shows that using a small number of channels (i.e., $c = 2$ or $c = 3$), gives us the desirable property that the security increases significantly for small increases in energy consumption, until a certain point. At this point, to get a security level where virtually all of the legitimate links are connected and secure requires a much larger increase in energy consumption in our protocol.

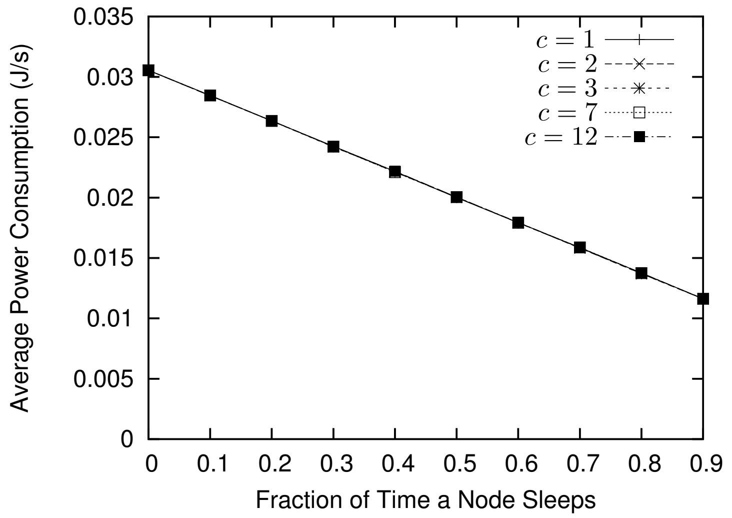

Figure 6.12: Average power consumption vs. the fraction of time legitimate sensors spend sleeping.

6.5 Discussion

We now discuss our protocol in relation to the key predistribution method (e.g., [89, 90, 93, 95]) as well as the approach of Anderson et al. [6]. In some scenarios, these methods may be preferable to our protocol. However, we believe properties of our protocol make it desirable in many environments.

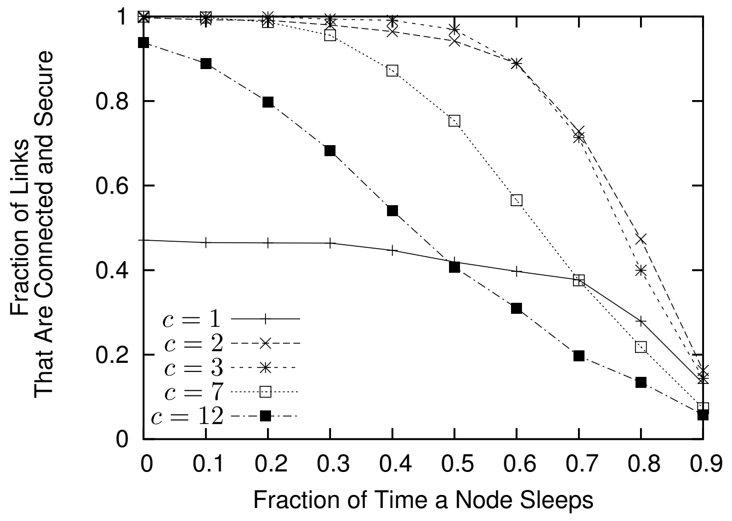

Figure 6.13: Secure connectivity of legitimate sensors vs. the fraction of time legitimate sensors spend sleeping.

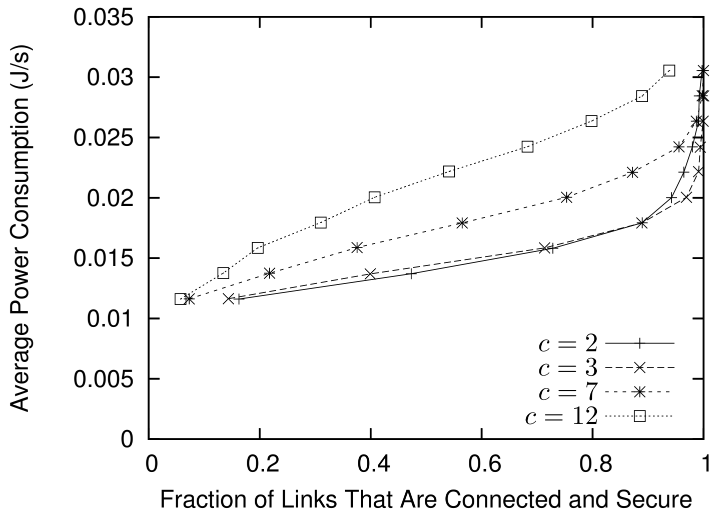

Figure 6.14: Average power consumption vs. the secure connectivity of legitimate sensors.

6.5.1 Comparison with Predistribution Schemes

We begin with some comparative advantages of our scheme:

- *Network Connectivity:* As shown in Section 6.4, our protocol is able to achieve close to 100% network connectivity in many settings *without using multipath reinforcement*, which requires a sensor pair to rely on other sensors to establish their link key. Allowing a sensor to communicate with all of its neighbors is desirable from a performance perspective since it gives more options for forwarding a packet over a high quality link [134].

- *Localizing Damage from Sensor Compromise:* In the other predistribution protocols, every time a sensor is captured, the entire network becomes slightly less secure since the attacker learns more about the network's global key pool. By contrast, our protocol localizes the damage caused by compromised devices. If an attacker captures many sensors in one region of the network, it does not learn anything about the link keys being used in another region of the network. This is because our protocol forms link keys based on the key sets of nearby sensors rather than from a network-wide key set.

Some comparative disadvantages of our scheme include:

- *Multihop Key Sharing:* In some applications (e.g., [135,136]), it may be desirable for key establishment to result in shared keys between sensors that are multiple hops away from each other. Predistribution schemes provide this property since a sensor is as likely to share keys with one-hop neighbors as it is to share keys with sensors in other regions of the network. Our protocol is localized and, therefore, establishes keys among only neighboring sensors. Adapting our protocol to establish keys with sensors multiple hops away is an area for future work.

- *Key Set Authentication Overhead and Vulnerability:* While key predistribution schemes do have significant overhead to advertise their key sets, our protocol has the added overhead of sending Merkle nodes to authenticate the Bloom filter of the key set that will be broadcast by a sensor. However, we note that other sensor protocols have been proposed with $O(\lg N)$ overhead associated with Merkle trees (e.g., [129]) and suggested methods to improve this overhead if location information is available. Additionally, our protocol introduces a vulnerability that is not present in key predistribution schemes whereby an attacker could generate arbitrary keys that will be accepted as legitimate when broadcast. However, we have discussed methods to address this problem, such as increasing the Bloom filter size or increasing the Merkle tree size, in Section 6.2.2.

6.5.2 Comparison with Anderson et al. [6]

Compared to Anderson's protocol, our protocol is more complex and has more overhead. Additionally, our protocol requires the availability of multiple channels, which we do not view as a disadvantage since current sensors [25] already have this capability. We feel that our protocol offers significant comparative advantages:

- *Greatly Increased Security:* Any adversary that compromises our protocol could also compromise Anderson's protocol [6]. However, in many cases, Anderson's protocol is compromised but our protocol is not. Anderson's protocol can provide security no greater than the $c = 1$ case that is simulated in

Section 6.4. In the same section, we show that $c > 1$ significantly improves resilience to colluding malicious sensors.

- *Increased Link Authentication:* From the description in Chapter 2, it is easy to see that Anderson's scheme is vulnerable to identity theft whereby a malicious device claims a legitimate sensor's ID and creates link keys with neighbors using this ID. In our protocol, we preload the sensors with data necessary to authenticate their ID and key set by a trusted source. We note that the authentication mechanisms that we use could be adapted for use in Anderson's protocol.

6.6 Extensions

6.6.1 Incremental Deployment

We assume that incremental sensor deployment is done in a planned manner rather than a completely ad hoc fashion. When the network is initially set up, the owner is assumed to have accurate knowledge of how many incremental deployments will occur during the lifetime of a sensor as well as the maximum number of new sensors that will be deployed each time. The new sensors are deployed in batches rather than individually.

When new devices are added after the initial deployment, sensors start another link key initialization procedure. A link key initialization procedure after the initial deployment could be triggered by one of several means. It could be at regular, predetermined intervals when the incremental deployment will happen. Alternatively, the new sensors could request the initialization procedure on demand by broadcasting their authenticated Bloom filters. Finally, a trusted source could send packets to the sensors telling them when to start the procedure. As mentioned earlier, sensors that already have established link keys do not try to create a new link key during subsequent link key initialization procedures.

Having discussed how the initialization, key discovery, and key establishment phases can be triggered for incremental deployment, we now propose a modification to the predeployment phase to allow authentication among existing and new sensors as well as provide a new key set for existing sensors in a space-efficient manner.

First, we focus on authentication among existing and new sensors since it has a relatively simple solution. Because incremental deployments are planned, it is assumed that the network administrator plans no more than I incremental deployment over the lifetime of a sensor. Furthermore, for the i-th incremental deployment, the network owner knows in advance that no more than N_i sensors will be deployed at that time (we let $i = 0$ be the initial deployment). Thus, whenever a sensor is deployed along with its associated Merkle tree values, it is also loaded with the roots of the next $I - i$ Merkle trees that will be generated by the trusted source for future deployments. The trusted source is able to generate these $I - i$ Merkle roots in advance since the maximum number of sensors that will be associated with each of these $I - i$ Merkle trees is known in advance. This allows an existing sensor to authenticate newly deployed sensors. To allow newly deployed sensor to authenticate existing sensors, the new sensors are loaded with the previous i Merkle roots that were generated for prior deployments. Given that sensors are loaded with these roots by the trusted source, they can now authenticate the keys of sensors deployed over the previous i generations or $I - i$ generations in the future.

The second issue that needs to be addressed is how an existing sensor can generate a new set of authenticatable keys to broadcast in subsequent deployments. If a sensor continues using the same α keys for every initialization phase, this gives the attacker the chance to learn more of the sensor's preloaded keys. If the attacker records previously heard $LREQ/LREP$ handshakes, then it may be able to break existing link keys as more of the preloaded keys are learned. Also, we seek to avoid using $\alpha \times I$ extra storage per sensor. Thus, we present a scheme which requires only $\alpha + 2I - 1$ extra storage per sensor. To do this, the Merkle tree generated in Section 6.2.2 is modified to be a two-level Merkle tree as follows.

Each sensor is again loaded with α unique, secret values. However, rather than use these values directly as keys to broadcast, they are used to create a hash chain to generate key values. Specifically, each sensor is loaded with the secret values $sv_1^1, sv_2^1, \ldots, sv_\alpha^1$. The first set of keys a sensor broadcasts is generated by applying a one-way hash function, H_{key}, to each of these α secret values. Thus, the first key broadcast when the sensor is initially deployed is $H_{key}(sv_1^1)$ and the last key broadcast during the sensor's first initialization procedure is $H_{key}(sv_\alpha^1)$. When a new batch of sensors is deployed, the existing sensor must generate keys to use for its second initialization procedure using a new set of secret values, $sv_1^2, sv_2^2, \ldots, sv_\alpha^2$. This is done by applying a hash function (not necessarily one-way), H_{sv}, to each of its secret values to generate a new set of secret values. That is, $sv_j^2 = H_{sv}(sv_j^1)$ for $j = 1, \ldots, \alpha$. The keys for the second initialization phase are then created by applying the one-way hash function H_{key} to each of the sv_j^2 values. So, the first key broadcast for the second initialization procedure is $H_{key}(sv_1^2)$ and the last key is $H_{key}(sv_\alpha^2)$. Figure 6.15 illustrates this process. Note that in between initialization procedures, a sensor needs store only α secret values rather than $\alpha \times I$ keys.

Now that we have specified a way for a trusted source to generate each of a sensor's I sets of α keys each in advance while using only α storage on the sensor, we must create the Merkle tree used to authenticate the Bloom filter for each set of keys. To do this, we extend the Merkle tree discussed in Section 6.2.2 by making each leaf node the root of another Merkle tree.[8] Thus, each sensor in a given deployment generation

[8]Though this structure is actually just one larger Merkle tree, we refer to it as a two-level tree for ease of explanation.

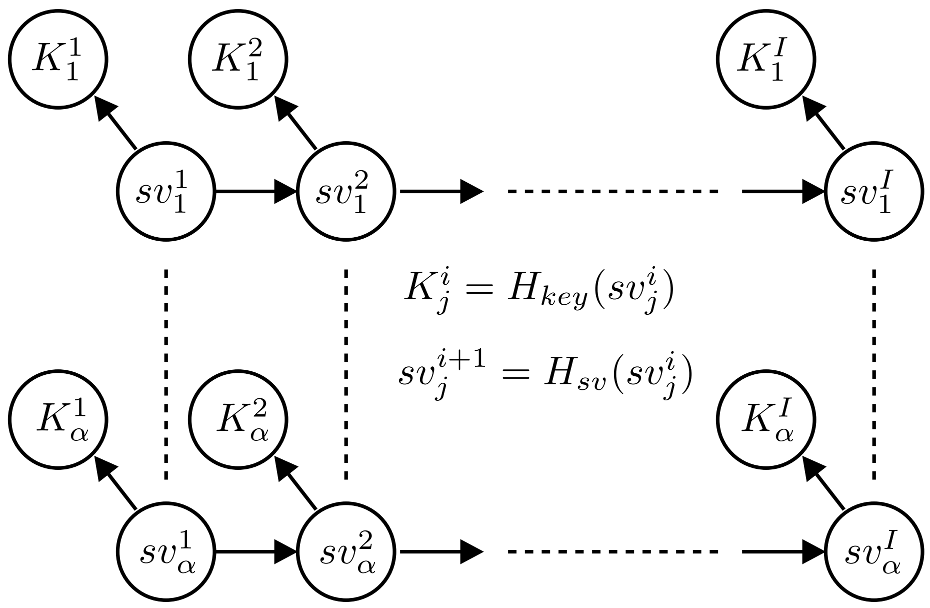

Figure 6.15: Key generation from secret values for incremental deployment. K_j^i denotes the j-th key broadcast by a sensor in the i-th initialization phase. H_{key} is a one-way hash function and H_{sv} is a different hash function that is not necessarily one-way.

has its own unique second Merkle tree. This second Merkle tree has I leaf nodes, one for each of the I authenticated Bloom filters corresponding to its key sets. Each sensor is then loaded with the $2I - 1$ nodes from its second Merkle tree to authenticate its Bloom filters along with the $\lg N$ nodes from the primary Merkle tree to authenticate the root of its second Merkle tree.

An example of a two-level Merkle tree is shown in Figure 6.16. In this example, $N = 4$ and $I = 4$. Without loss of generality, consider the second sensor in the deployment. It is loaded with the four Bloom filters necessary to authenticate its key sets, BF_1^2, BF_2^2, BF_3^2, and BF_4^2. The filters are then hashed to form the leaves of its local Merkle tree. This local Merkle tree is constructed as described in Section 6.1.4 to generate root R_2. This process is repeated for the three other sensors as well. Using these four roots as leaves, the primary Merkle tree is constructed with root R_0. The second sensor is then loaded with all the nodes from the subtree rooted at R_2 as well as the $\lg N$ nodes from the primary tree necessary to authenticate root R_0. The sensor must be loaded with all of the nodes rooted at R_2 in order to ensure each of its I Bloom filters can be authenticated.

6.6.2 Path Diversity

In this section, we explore another type of diversity that can be used to further improve our protocol. Ideally, we would like to take scenarios where our protocol achieves, say, 99% secure links and augment the protocol

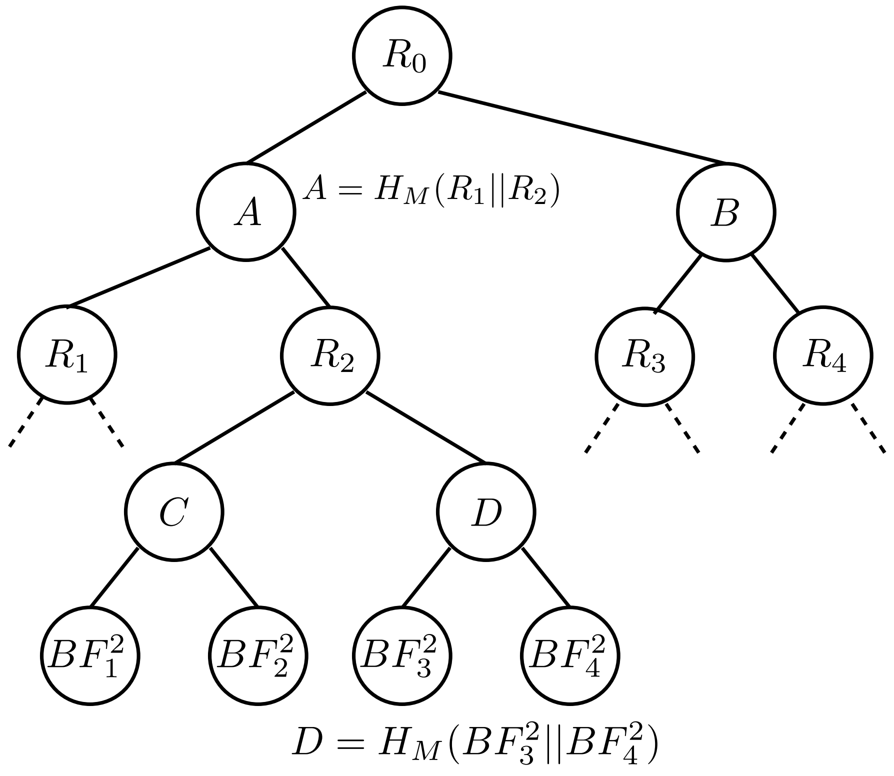

Figure 6.16: Two-level Merkle tree for incremental deployment. In this example, we have four sensors and four Bloom filters per sensor for the sets of keys on that device. BF_j^i refers to the Bloom filter for the keys of the i-th sensor for the j-th initialization procedure.

to get, say, 99.9999% secure links. Thus, we consider the idea of path diversity, illustrated in Figure 6.17. In this example, assume that the link between U and V is compromised, but all the links on the paths $U \rightarrow A \rightarrow V$ and $U \rightarrow B \rightarrow C \rightarrow V$ are secure. U can send a key, k_1, along the $U \rightarrow A \rightarrow V$ path and a key, k_2, along the $U \rightarrow B \rightarrow C \rightarrow V$ path. V could then create a secure link key, k_{uv} by combining k_1 and k_2 (e.g., $k_{uv} = k_1 \oplus k_2$). Provided that the intermediate sensors are trusted, U needs to have only one path to V that has all secure links in order to form a secure key.

We note that this is similar to the multipath reinforcement scheme proposed in [90]. However, in [90], multipath reinforcement was proposed to combat *node* compromise. By contrast, our path diversity scheme aims to address *link* compromise. Thus, generally, multipath reinforcement algorithms need to discover node disjoint paths whereas we need discover only link disjoint paths. A node disjoint algorithm guarantees link disjointness, but a link disjoint algorithm need not necessarily find a node disjoint path. However, we have found that there is little benefit to considering paths longer than two hops (i.e., using a shared neighbor of U and V) and the algorithm complexity increases significantly. In this case, node disjointness and link disjointness between two nodes are equivalent.

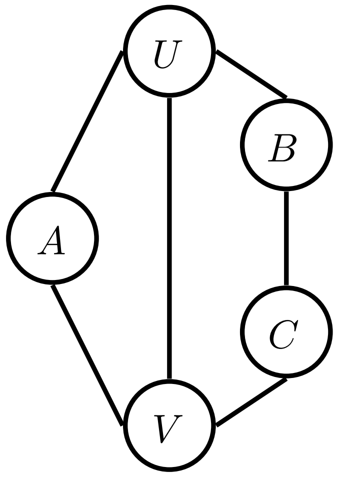

Figure 6.17: Example topology for demonstrating path diversity.

Analysis: Given a that links are secure with probability[9] p, the probability of using one shared neighbor for path diversity increases the probability that the link is secure to:

$$\begin{aligned} \Pr[\text{Link secure using one shared neighbor}] &= 1-(1-p)(1-p^2) \\ &= p(1+p-p^2) \end{aligned} \tag{6.7}$$

If 2-hop paths via N shared neighbors are used, we get:

$$\Pr[\text{Link secure using } N \text{ shared neighbors}] = 1-(1-p)(1-p^2)^N \tag{6.8}$$

The benefit from using longer link disjoint paths can be considered by looking at the scenario where there are N shared neighbors and M 3-hop paths:

$$\Pr\left[\begin{array}{c} N \text{ shared neighbors are compromised and} \\ \text{at least one of } M \text{ 3-hop paths is secure} \end{array}\right] = (1-p^2)^N(1-(1-p^3)^M) \tag{6.9}$$

Simulation Results: To test path diversity, we augmented our *ns-2* code to determine the link security if shared neighbors are used. We did some testing using 3-hop paths between two nodes. To do this, we performed the Hopcroft-Karp matching algorithm [110] on the bipartite graph generated with one set of vertices consisting of u's one-hop neighbors and the other set of vertices consisting of v's one-hop neighbors (where u and v are the nodes that are using path diversity). Hopcroft-Karp finds the maximal matching on

[9]We note that this analysis assumes independence among link compromises which is not always true. Later we use simulations to determine the effectiveness of path diversity when correlations may exist in link compromises.

a bipartite graph, resulting in the discovery of the maximum number of 3-hop paths between u and v. Our simulations showed virtually no improvement from using 3-hop paths over using two-hop paths, so we did not further pursue methods of finding longer link disjoint paths.

In Figure 6.18, we show an example scenario from one run where using path diversity does effectively reduce link compromise from over 10% to zero. Note that in this figure, the y-axis is inverted from our graphs in Section 6.4 and now shows the fraction of *compromised* links. The x-axis is the number of shared neighbors a link is allowed to use to try to improve their security. This demonstrates that it can be useful to use more than one shared neighbor.

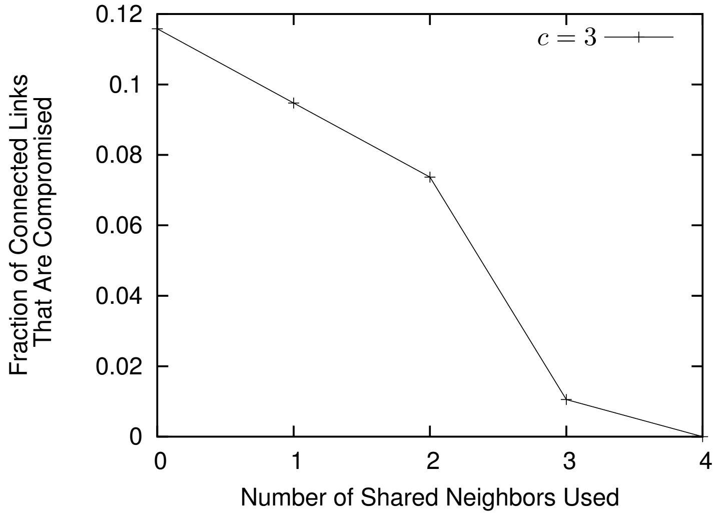

Figure 6.18: Example topology that benefits from path diversity.

However, when averaging over multiple runs, we found that it is difficult reach 100% security when some fraction of the links are initially compromised. Figure 6.19 shows an example where the fraction of secure links in the network plateaus regardless of how many paths are used. The x-axis is the maximum number of shared neighbors that a link is allowed to use to try to improve their security.

Unfortunately, we discovered that it is difficult to reach 100% link security even when the initial fraction of secure links is relatively high. To investigate why, we looked in more detail at the $(c = 2, \alpha = 40)$ case. From Figure 6.8, we see that these parameters give a high fraction of secure connectivity that is still less than one.

Table 6.5 shows how path diversity affects five individual runs. Each run used the same input parameters except for the random seeds used to generate the topology and run the simulations. Of the three runs that did not result in all links being securely connected, two of them improve to one when just one shared neighbor is used. However, in Run 2, the topology never becomes completely securely connected regardless

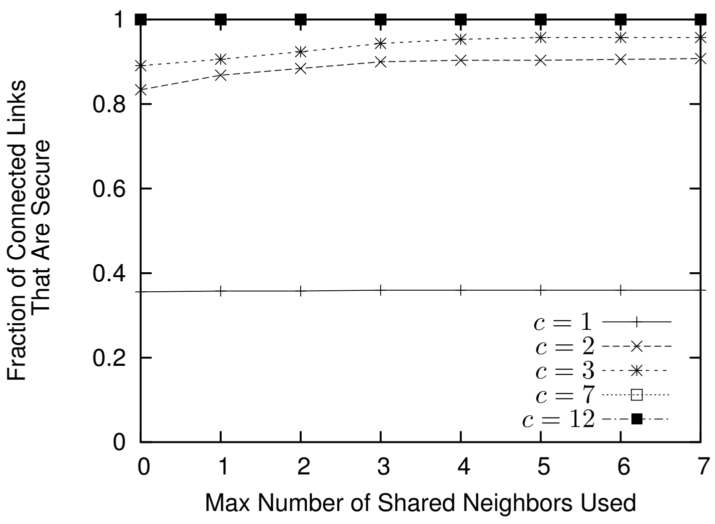

Figure 6.19: Security improvements using path diversity.

of how many shared neighbors are used. Instead, it plateaus after one shared neighbor is used at 0.978.

Table 6.5: Fraction of connected links that are secure using shared neighbors.

Run Number	**Maximum number of shared neighbors used**			
	0	**1**	...	**7**
1	1.0	1.0	...	1.0
2	0.968	0.978	...	0.978
3	1.0	1.0	...	1.0
4	0.989	1.0	...	1.0
5	0.982	1.0	...	1.0

The reason for this is that some topologies are compromised such that some sensors are partitioned from their neighbors with respect to secure links. For example, in Run 2, we discovered the topology shown in Figure 6.20. The dashed lines represent compromised links and the solid lines represent secure links. In this example, we see that A can used shared neighbors to form a secure key with B, but cannot form one with E or F since the compromise is such that there exists a partition. If E, F, and H do not have any other neighbors, then it is impossible for A to form a secure key with any of them regardless of how much path diversity is allowed.

In conclusion, our research in path diversity has shown that:

- Path diversity can improve the security of some links in our key distribution protocol. Depending on the topology, in can make all links secure.

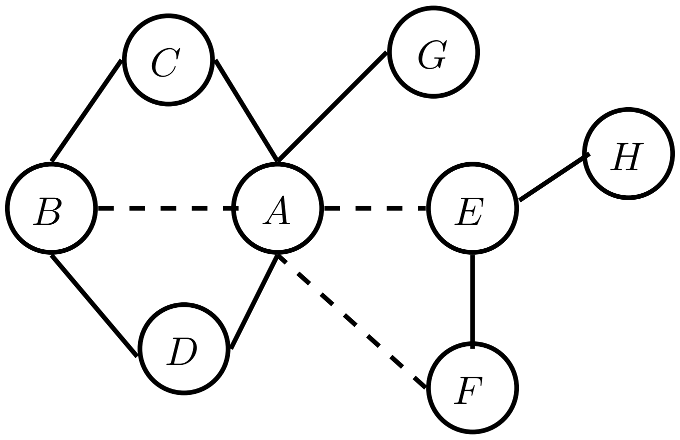

Figure 6.20: Example topology of network security partition. Dashed lines are compromised links and solid lines are secure links.

- Path diversity's utility in making all links secure in a wide variety of networks is limited due to the security partitions which can occur in large randomly deployed topologies.
- Finding the maximum number of link disjoint paths beyond two hops increases algorithm complexity significantly and gives little improvement over exclusively using two-hop paths.

6.7 Summary

In this work, we have proposed a novel method of symmetric key establishment for a sensor network that uses channel diversity, as well as spatial diversity, to create link keys for one-hop neighbors. Establishing such keys is important because public keys are computationally expensive for many sensors. Sharing a symmetric key with neighbors allows for secure aggregation as well as hash chains authentication, for example.

Via analysis and simulation, we show that our protocol performs well in network connectivity and resilience to colluding malicious devices compared with previous work. One result is that using even *one* extra channel for broadcasting keys during the initialization phase significantly improves security. From a numerical perspective, our simulations demonstrate that our protocol can achieve over 90% connectivity among neighboring sensors with link keys that are uncompromised even when 80% of the devices in the network are malicious and collude.

Chapter 7

Conclusion

In this dissertation, we have considered the problems of saving energy and security in multihop wireless networks. In Chapter 1, we discuss that wireless networking is a rapidly growing area and discuss the advantages offered by multihop wireless. We then quantify the importance of the energy-efficiency problem by presenting data that shows: (1) the energy density of batteries has shown little improvement in recent years and (2) the radio interface can be a major source of energy drain, particularly in devices with little or no displays (e.g., cell phones and sensors). We also argue that security research is of great importance in this domain due to the ease with which the channel can be tapped and the resource constraints faced by many wireless devices. In Section 1.1, we outline the major contributions of this dissertation.

Having demonstrated the relevance of the problems that we are addressing, we next propose techniques to address energy efficiency and secure symmetric key distribution. In Chapter 3, we propose techniques that reduce wasteful listening while checking for signals to wake up a device's radio. In particular, we propose using carrier sensing to make wake-up signal checking more efficient for both in-band and out-of-band protocols.

In Chapter 4, we use adaptive sleeping and listening to improve the energy efficiency of in-band protocols. This compliments our previous work [27–29] that used adaptive techniques for out-of-band protocols.

Chapter 5 focuses on energy efficient broadcast dissemination in power save networks. In particular, we propose a probabilistic protocol that allows users to reduce energy consumption while maintaining a desired latency and reliability. Additionally, we implemented our protocol in TinyOS [11] to test it on readily available sensor hardware.

In Chapter 6, we address security in sensor networks by proposing a pairwise symmetric key distribution protocol. Unlike previous work, we propose using the underlying channel diversity to address the problem. In doing so, our protocol shows significant improvements in connectivity and resilience to adversaries when compared to other protocols.

7.1 Future Work

In this section, we briefly mention areas of future work based on our thesis. We outline four directions—two in energy efficiency and two in security.

- *Implementation and testing in an application context:* Most of our work is tested in simulation with controlled traffic patterns. To truly quantify and qualify the benefits of our power save protocols, an implementation would be beneficial. As with the implementation of our broadcast protocol in Section 5.3, we anticipate that many design decisions do not manifest themselves in simulation. Additionally, this would provide code that users in need of power save could run.

- *Power save for multichannel and multi-interface protocols:* With the emergence of devices and protocols that use multiple channels and/or are equipped with multiple interfaces [137–139], it may be interesting to determine how power save can be used in this realm. All of our energy efficiency work focuses on either devices with a single channel or two interfaces (one of which is used exclusively for control messages). While such scenarios are of interest since they are widely used, considering the more general case of how to design power save protocols for devices with k channels and m interfaces may be a promising area of future work.

- *Quantifying the tradeoffs of public key exchange versus symmetric key approaches in sensor networks:* As mentioned in Section 2.3, we think that the research community could greatly benefit from a rigorous comparison of using public key exchange on sensors versus the pure symmetric key approaches emerging. While public key computation does have high computation overhead when compared with symmetric key operations, they provide many advantages. Public keys are much more secure in binding a key to an identity and have much less memory and communication overhead. Fully exploring such metrics would help determine under what circumstances not using public key exchanges is more efficient.

- *Exploring other techniques that use diversity for security:* In Chapter 6, we have proposed a method to use channel diversity for security in sensor networks. Traditionally, the focus of wireless diversity is on *performance* [137–139] instead of *security*. A potential area of future work is considering other uses of diversity for security (e.g., transmission power, channel bitrates, multiple interfaces and channels). Looking at applications of such diversity to security is a promising area of future work.

Appendix A

Carrier Sensing Modifications to Handle Synchronization Errors

In this section, we show the correctness of the modifications to CS-ATIM discussed in Section 3.1.1. As mentioned previously, we assume that the node's clocks are always within Δ seconds of each other. Thus, Δ represents the maximum error between the clocks of *any* two nodes in the network. The modifications, shown in Figure A.1, are as follows. Without loss of generality, we assume that a node with the fastest clock in the network begins the current beacon interval at time T_{f0}. Thus, the latest a node's beacon interval can begin is:

$$T_{s0} = T_{f0} + \Delta \tag{A.1}$$

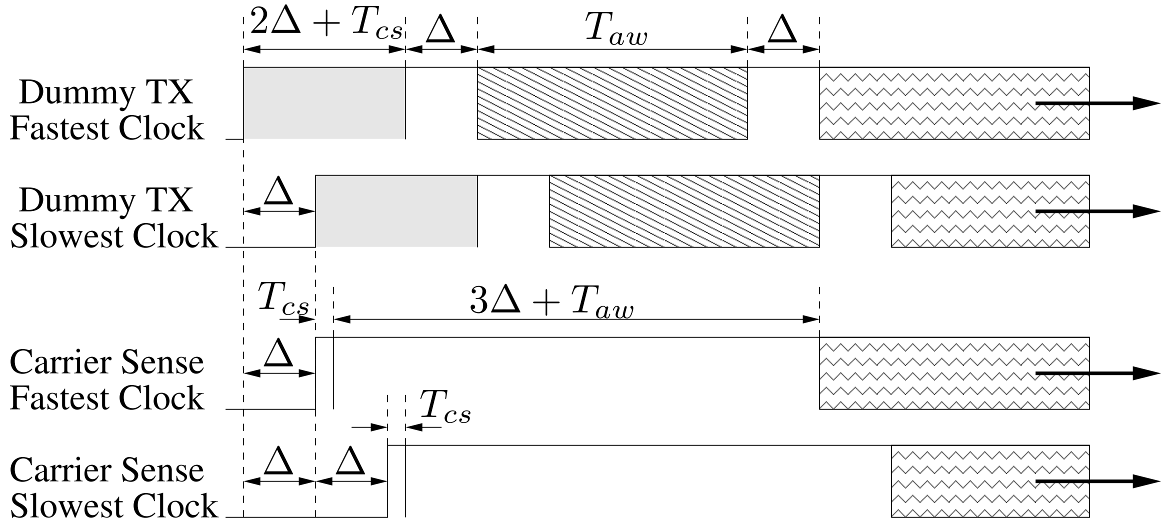

Figure A.1: CS-ATIM time synchronization. The shaded area denotes a dummy packet being sent. The slanted lines represent the ATIM window when ATIM packets can be transmitted. The wavy lines denote when data packets can be transmitted.

To account for Δ, nodes that have no packets to advertise must wake up Δ seconds after the beacon interval is scheduled according to their local clock. Thus, a node with the fastest clock carrier senses the channel from time:

$$T_{f1} = T_{f0} + \Delta \tag{A.2}$$

until:

$$\begin{aligned} T_{f2} &= T_{f1} + T_{cs} \\ &= T_{f0} + \Delta + T_{cs} \end{aligned} \tag{A.3}$$

A node with the slowest clock carrier senses the channel from time:

$$\begin{aligned} T_{s1} &= T_{s0} + \Delta \\ &= T_{f0} + 2\Delta \end{aligned} \tag{A.4}$$

until

$$\begin{aligned} T_{s2} &= T_{s1} + T_{cs} \\ &= T_{f0} + 2\Delta + T_{cs} \end{aligned} \tag{A.5}$$

For a node that has packets to advertise, it begins transmitting the dummy packet when the beacon interval begins according to its local clock and transmits the dummy packet for $2\Delta + T_{cs}$ time. Thus, a node with the fastest clock transmits its dummy packet from time:

$$T_{f3} = T_{f0} \tag{A.6}$$

until:

$$T_{f4} = T_{f0} + 2\Delta + T_{cs} \tag{A.7}$$

Since $T_{f3} < T_{f1} < T_{f2} < T_{f4}$ and $T_{f3} < T_{s1} < T_{s2} = T_{f4}$, nodes with the fastest clock and nodes with the slowest clock are both guaranteed to carrier sense this dummy packet for the specified T_{cs} length of time. A node with packets to advertise with the slowest clock will transmit its dummy packet from time:

$$\begin{aligned} T_{s3} &= T_{s0} \\ &= T_{f0} + \Delta \end{aligned} \tag{A.8}$$

until:

$$\begin{aligned} T_{s4} &= T_{s0} + 2\Delta + T_{cs} \\ &= T_{f0} + 3\Delta + T_{cs} \end{aligned} \tag{A.9}$$

Since $T_{s3} = T_{f1} < T_{f2} < T_{s4}$ and $T_{s3} < T_{s1} < T_{s2} < T_{s4}$, nodes with both the fastest and slowest clocks will carrier sense this dummy packet for the specified time, T_{cs}.

If a node without packets to advertise detects the channel idle at the end of the T_{cs} time, it will return to sleep. However, if the node detects the channel as busy, it will remain on for an additional $3\Delta + T_{aw}$ time, as show in Figure A.1, for reasons explained below. For a node that *does* have a packet to advertise, it will begin sending ATIM packets Δ seconds after it finishes transmitting the dummy packet. During this Δ time gap between the end of the dummy packet and the beginning of the ATIM window, the node *may* receive packets and reply with ACKs, however, it *may not* send any ATIMs or data packets during this time. At the end of the ATIM window T_{aw} seconds later, the node waits another Δ seconds before it starts sending data packets. Again, during the Δ time gap, the node may receive and reply with ACKs, but may not send ATIMs or data packets. For a node with the fastest clock, it is guaranteed to be on from time T_{f4} (the time it finished transmitting the dummy packet) until:

$$\begin{aligned} T_{f5} &= T_{f4} + 2\Delta + T_{aw} \\ &= T_{f0} + 4\Delta + T_{cs} + T_{aw} \end{aligned} \tag{A.10}$$

A node with the fastest clock is allowed to transmit during its ATIM window which starts at time:

$$\begin{aligned} T_{f6} &= T_{f4} + \Delta \\ &= T_{f0} + 3\Delta + T_{cs} \end{aligned} \tag{A.11}$$

and ends at time:

$$\begin{aligned} T_{f7} &= T_{f6} + T_{aw} \\ &= T_{f0} + 3\Delta + T_{cs} + T_{aw} \end{aligned} \tag{A.12}$$

For a node with the slowest clock, it is guaranteed to be on from time T_{s4} until:

$$\begin{aligned} T_{s5} &= T_{s4} + 2\Delta + T_{aw} \\ &= T_{f0} + 5\Delta + T_{cs} + T_{aw} \end{aligned} \tag{A.13}$$

A node with the slowest clock is allowed to transmit during its ATIM window which starts at time:

$$\begin{aligned} T_{s6} &= T_{s4} + \Delta \\ &= T_{f0} + 4\Delta + T_{cs} \end{aligned} \tag{A.14}$$

and ends at time:

$$\begin{aligned} T_{s7} &= T_{s6} + T_{aw} \\ &= T_{f0} + 4\Delta + T_{cs} + T_{aw} \end{aligned} \tag{A.15}$$

Because $T_{f4} < T_{s6} < T_{s7} = T_{f5}$, a node with the fastest clock is guaranteed to be listening during the entire ATIM window of a node with the slowest clock. Similarly, because $T_{s4} = T_{f6} < T_{f7} < T_{s5}$, a node with the slowest clock is guaranteed to be listening after its dummy packet transmission during the entire ATIM window of a node with the fastest clock.

Appendix B

Minimum Energy Routing Proof

Sections B.1, B.2, and B.3 give instances of known NP-complete problems [140, 141]. We use these for the reduction in our proof in Section B.4.

B.1 Steiner Tree Problem (*ST*)

INSTANCE: An undirected graph $G = (V, E)$, an edge cost function $c : E \rightarrow N$, a subset $S \subseteq V$ of required vertices.

SOLUTION: A subtree of G that includes all the vertices in S. This is called a Steiner tree. Note that vertices in $V \setminus S$ may be included in the Steiner Tree and are called *Steiner vertices*.

MEASURE: Sum of the edge weights in the subtree.

B.2 Steiner Tree With Unit Edge Weights Problem (*ST-UE*)

A proof to show that ST is NP-complete is based on a reduction from the exact covering by 3-sets problem [140]. The ST proof [140] answers the following decision problem: given an undirected bipartite graph $G = (V, E)$, a subset of vertices $S \subseteq V$, and an integer B, is there a tree T in G that spans all of the S terminals and has at most B edges?

By design, the ST proof [140] shows that ST is NP-complete even if the cost function is $c : E \rightarrow 1$. Thus, we know that even though *ST-UE* (defined below) is a limited case of *ST*, it is still NP-complete.

INSTANCE: An undirected graph $G = (V, E)$, an edge cost function $c : E \rightarrow 1$, a subset $S \subseteq V$ of required vertices.

SOLUTION: A subtree of G that includes all the vertices in S.

MEASURE: Sum of the edge weights in the subtree.

We can see that the measure in *ST-UE* is equivalent to the following measure:

MEASURE 2: The number of edges in the subtree.

Trivially, minimizing the number of edges in a subtree also minimizes the number of vertices in the subtree since $V_T = E_T + 1$.

B.3 Steiner Tree on Bidirected Graphs (*ST-BG*)

Any instance of *ST* (which, of course, includes *ST-UE*) can be reduced to *ST-BG* by replacing every undirected edge $e_{ij} \in E$ with two directed edges e_{ij} and e_{ji}[1] and giving both of the directed edges the same cost as the original undirected edge. Then, any one node in S, which we denote r, is chosen as the root.[2]

Thus, the *ST-BG* problem (also called the *Steiner arborescence problem* [140]) is defined as follows.[3]

INSTANCE: A bidirected graph $G = (V, E)$, an edge cost function $c : E \rightarrow 1$, a subset $S \subseteq V$ of required vertices, and a root vertex, r.

SOLUTION: A directed subtree of G such that there exists a path from r to every vertex in S.

MEASURE: Sum of the edge weights in the subtree.

The corresponding decision problem is: given an instance of *ST-BG*, is there a solution such that the sum of the edge weights is less than W?

B.4 Minimum Energy Routing for Multilevel Power Save (*MER*)

We now define the MER problem and show that it is NP-complete using a reduction from *ST-BG*. As described in Section 4.2, we only consider the latency induced by the power saving protocol because this delay tends to be larger relative to contention and queuing delay in the networks that we consider. Thus, the l_i term mentioned below is only a function of a node's power save state and *not* a function of the number of flows that it and its neighbors are forwarding.

[1]The notation e_{ij} denotes an edge between i and j in the undirected case and a directed edge from i to j in the directed case.

[2]In the undirected case, declaring a root is unnecessary since every node can reach every other node in the tree. In the directed case, we specify a root to create a structure which ensures that the root can reach every other node in the tree.

[3]We skip the general definition of *ST-BG*, where $c : E \rightarrow N$, and just focus on the version with unit edge weights.

INSTANCE: A bidirected graph $G = (V, E)$, a set of flows F (i.e., a set of source-destination tuples), a maximum end-to-end latency threshold for a path L, and k the number of power save states available to each node. Each power save state has an associated latency, l_i, and energy consumption, g_i (where $1 \leq i \leq k$). For $i < j$, $l_i \leq l_j$ and $g_i \geq g_j$. When a node is in PS state i, its energy consumption is g_i and the latency cost of all its *incoming* edges is l_i.

SOLUTION: A set of power save states for each node such that each flow in F can be routed without L being violated for *any* of the flows.

MEASURE: Sum of the energy consumed by the power save state (i.e., g_i) of each node in the network.

The decision problem that we use for *MER* is: can we assign power save states for an instance of *MER* such that the sum of the energy consumed by the power save state of each node is less than Y?

It is easy to verify that *MER* is in NP. Given a set of PS states for each node, all of the link costs in the network can be fixed (i.e., the appropriate value of l_i for all incoming links to a node). Then, we do shortest path routing on the weighted graph obtained by using latencies as edge weights for each flow in F and verify that the cost of each path is less than L, which can be done in polynomial time. Additionally, we verify that the sum of all the power save states is less than Y which can be done in polynomial time.

For convenience, we consider a special case of *MER* where:

- $k = 2$
- $g_1 = 1$ and $g_2 = 0$
- $l_1 = 1$ and $l_2 = |V|$
- $L = |V| - 1$
- All flows originate from one sender
- A flow *is capable* of satisfying the latency constraint. This can be checked in polynomial time by placing all nodes in their highest energy state and computing a flow's shortest path cost. If this cost is greater than L, then we can immediately decide that the instance of *MER* is unsolvable.

By showing that the above special case of *MER* is NP-complete, we will have proved the general *MER* problem to be NP-complete. We do so with a reduction from *ST-BG*. We show that given any instance of the *ST-BG* problem, it is possible to construct an instance of the *MER* problem such that the instance of *ST-BG* has a total edge weight less than W if and only if the *MER* instance has a total energy consumption less than $W + 1$.

Given an instance of *ST-BG*, we convert it to an instance of *MER* as follows. The graph, G, from *ST-BG* is used as the graph in *MER*. The root, r, from *ST-BG* is the one sender in our special case of *MER* and each vertex in *ST-BG*'s S set corresponds to a receiver in *MER*.

Now, we need to show that an instance of *ST-BG* has a total edge weight less than W if and only if the corresponding *MER* instance has a total energy consumption less than $W + 1$.

If *ST-BG* Has Total Edge Weight $< W$: Then, we select all of the nodes in *ST-BG*'s subtree to remain in PS state 1 while all other nodes are put in PS state 2. Since the cost of each edge in the tree is 1 and there can be at most $|V| - 1$ edges in the tree, then the latency must be less than or equal to L. This is because each of the selected nodes has an incoming latency of $l_1 = 1$ and there can be at most $|V| - 1$ edges in the tree since there are $|V|$ nodes total. Thus, the total latency is at most $L = |V| - 1$. Since there must be at most $W - 1$ edges in the subtree, there can be at most W nodes in PS state 1 and, thus, the sum of the energy consumption in the network is less than $W + 1$.

If *MER* Has Total Energy Consumption $< W + 1$ and the Latency $\leq L = |V| - 1$: Then, all the nodes on every routing path must be in PS state 1 or else the latency would be greater than L (since one node in PS state 2 would make the latency at least $|V| > L$). Thus, each node on the routing paths is using one unit of energy. Therefore, if the total energy consumption is less than $W + 1$, then at most W nodes in the network are using one unit of energy and the source can reach all receivers. Since the source, each receiver, and all intermediate nodes on the paths form a tree with at most W nodes, we have a subtree with at most $W - 1$ edges.

■

Appendix C

PBBF Interfaces and Packet Formats in TinyOS

C.1 Packet Formats

C.1.1 SimplePbbfMsg

```
typedef struct SimplePbbfMsg {
   /* Source can be either the broadcast source or a predefined UART address */
   uint16_t source;
   /* A broadcast sequence number that is unique per source */
   uint8_t seqno;
   /* How many hops the packet has traveled from the broadcast source */
   uint8_t hopCount;
   /* The p value that the node should use */
   uint8_t pVal;
   /* The q value that the node should use */
   uint8_t qVal;
   /* The r value that the node should use */
   uint8_t rVal;
   /* Sequence number of the test run for this packet */
   uint8_t runSeqno;
   /* Timestamp for stats (granularity = 1/921.6 kHz) */
   uint32_t latency;
} __attribute__ ((packed)) SimplePbbfMsg;
/* packed attribute removes field padding on Mica2 architecture */
```

C.1.2 PbbfStatsMsg

```
typedef struct PbbfStatsMsg {
   /* Source ID of the node reporting the stats */
   uint16_t nodeId;
   /* Total amount of data packets sent by SimplePbbfBcast */
   uint16_t totalDataSent;
   /* Total amount of data packets resent according to the r parameter */
   uint16_t totalDataResent;
   /* Total data packets received by SimplePbbfBcast (includes duplicates) */
   uint16_t totalDataRecv;
   /* Total application level packets received (duplicates suppressed) */
   uint16_t totalAppRecv;
   /* Average end-to-end latency from broadcast source to this node */
   uint32_t avgLat;
   /*****
    * Fraction of time the node's radio was not sleeping
    * 0=0%, 255=100%, uniform spacing between
    *****/
   uint8_t fracOnTime;
   /* Sequence number of the test run for which stats are being reported */
   uint8_t runSeqno;
} __attribute__ ((packed)) PbbfStatsMsg;
/* packed attribute removes field padding on Mica2 architecture */
```

C.1.3 UARTMsg

UARTMsg can be of type either SimplePbbfMsg or PbbfStatsMsg.

C.2 Interfaces

C.2.1 PBBF Interface

```
interface PbbfControl {
   /*****
    * Each parameter can be set to one of 11 discrete values corresponding
    * to probability values between 0.0 and 1.0 spaced uniformly.
    *****/
   command void setPLevel(uint8_t);
   command void setQLevel(uint8_t);
   command void setRLevel(uint8_t);
}

interface PbbfNotifier {
   /* Signals PBBF module when a sleep decision needs made */
   event result_t sleepDecisionPoint();
   /* PBBF tells the signaling module whether or not to sleep */
   command void setSleep(bool);
}
```

C.2.2 Broadcast Send Interface

```
interface BcastSender {
   /* Used by the application to send a broadcast packet */
   command result_t send(TOS_MsgPtr);
   event result_t sendDone(TOS_MsgPtr, result_t);
}
```

C.2.3 Stats Handling Interface

```
interface PktStats {
   /* A packet was sent, bool tells whether it was a control packet */
   command void SentPkt(bool);
   /* A packet was received, bool tells whether it was a control packet */
   command void RecvdPkt(bool);
   /*****
    * Collect the stats from the received packet.  uint32_t is the
    * latency of the packet since it was sent by the source.
    *****/
   command void HandleRecvdStats(TOS_MsgPtr, uint32_t);
   /* Signal that the stats have been handled */
   event result_t HandleRecvdStatsDone(TOS_MsgPtr);
}

interface RadioPktStats {
   /* Signal to the stats module when the radio switches on and off */
   event result_t radioPoweredOn();
   event result_t radioPoweredOff();
}

interface ReportStats {
   /*****
    * Used by the stats collection module to transmit collected stats
    * back to the sink.
    *****/
   command result_t ReportStats();
   event result_t ReportStatsDone(result_t);
}

interface PbbfStats {
   /*****
    * Signal to the stats module when PBBF sends a packet twice
    * according to the r parameter.
    *****/
   event result_t didSecondSend(TOS_MsgPtr);
}
```

C.2.4 Control Packet Handling Interface

```
interface NetworkInit {
   /*****
    * Signals a component when a control packet has
    * been received to initialize the current test run
    * for an application.  The input parameters gives a
    * unique sequence number to identify the test run.
    *****/
   event result_t isInitialized(uint8_t);
}

interface PktHandler {
   /*****
    * When a control packet is received, pass it to the
    * control packet handling module.
    *****/
   command result_t handle(TOS_MsgPtr);
   event result_t done(TOS_MsgPtr, result_t);
}
```

References

[1] M. J. Miller, C. Sengul, and I. Gupta, “Exploring the Energy-Latency Trade-off for Broadcasts in Energy-Saving Sensor Networks,” in *IEEE ICDCS 2005*, June 2005.

[2] C. Schurgers, V. Tsiatsis, S. Ganeriwal, and M. Srivastava, “Topology Management for Sensor Networks: Exploiting Latency and Density,” in *ACM MobiHoc 2002*, June 2002.

[3] C. Schurgers, V. Tsiatsis, S. Ganeriwal, and M. Srivastava, “Optimizing Sensor Networks in the Energy-Latency-Density Design Space,” *IEEE Transactions on Mobile Computing*, vol. 1, pp. 70–80, January–March 2002.

[4] B. H. Bloom, “Space/Time Trade-offs in Hash Coding with Allowable Errors,” *Communications of the ACM*, vol. 13, July 1970.

[5] R. C. Merkle, “A Certified Digital Signature,” in *Advances in Cryptology - CRYPTO 1989*, August 1989.

[6] R. Anderson, H. Chan, and A. Perrig, “Key Infection: Smart Trust for Smart Dust,” in *IEEE International Conference on Network Protocols (ICNP) 2004*, October 2004.

[7] IEEE 802.11, *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, 1999.

[8] D. A. Gross, “The Other Wireless Revolution.” `http://tinyurl.com/lsrss` (`http://www.state.gov/e/eb/rls/rm/2005/48757.htm`), June 2005.

[9] D. McGrath, “Report: RFID production to increase 25 fold by 2010.” `http://tinyurl.com/dmjg5` (`http://www.eetimes.com/showArticle.jhtml?articleID=177101437`), January 2006.

[10] K. German, “CNET’s quick guide to Bluetooth Headsets.” `http://tinyurl.com/dslev` (`http://reviews.cnet.com/4520-6454_7-5140288-1.html`), June 2004.

[11] TinyOS Community Forum. `http://webs.cs.berkeley.edu/tos`.

[12] TinyOS Community Forum Stats. http://www.tinyos.net/stats.html.

[13] R. Bruno, M. Conit, and E. Gregori, "Mesh Networks: Commodity Multihop Ad Hoc Networks," *IEEE Communications Magazine*, vol. 43, pp. 123–131, March 2005.

[14] C. Thompson, "Talking in the Dark." http://tinyurl.com/9jfbw (http://www.nytimes.com/2005/09/18/magazine/18idea.html?ex=1284696000&en=e0e25a2b53a52cfa&ei=5090&partner=rssuserland&emc=rss), September 2005.

[15] B. Parno and A. Perrig, "Challenges in Securing Vehicular Networks," in *ACM Hot Topics in Networks (HotNets-IV) 2005*, November 2005.

[16] M. Raya and J.-P. Hubaux, "The Security of Vehicular Ad Hoc Networks," in *ACM SASN 2005*, November 2005.

[17] J. Heidemann, W. Ye, J. Wills, A. Syed, and Y. Li, "Research Challenges and Applications for Underwater Sensor Networking," in *IEEE WCNC 2006*, April 2006.

[18] D. Culler, D. Estrin, and M. Srivastava, "Overview of Sensor Networks," *IEEE Computer*, vol. 37, pp. 41–49, August 2004.

[19] P. Gupta and P. R. Kumar, "The Capacity of Wireless Networks," *IEEE Transactions on Information Theory*, vol. 46, pp. 388–404, March 2000.

[20] T. Starner, "Thick Clients for Personal Wireless Devices," *IEEE Computer*, vol. 35, pp. 133–135, January 2002.

[21] T. Starner, "Powerful Change Part 1: Batteries and Possible Alternatives for the Mobile Market," *IEEE Pervasive Computing*, vol. 2, pp. 86–88, October–December 2003.

[22] T. Starner. Email correspondence, June 2005.

[23] D. G. Sachs, W. Yuan, C. J. Hughes, A. Harris, S. V. Adve, D. L. Jones, R. H. Kravets, and K. Nahrstedt, "GRACE: A Hierarchical Adaptation Framework for Saving Energy," Tech. Rep. UIUCDCS-R-2004-2409, University of Illinois at Urbana-Champaign, February 2004.

[24] N. Jain. Vodafone Symposium Presentation at the University of Illinois at Urbana-Champaign, April 8–10, 2005. The author was at Qualcomm Research at the time of the presentation.

[25] Crossbow Technology Inc. http://www.xbow.com.

[26] J. Polastre, J. Hill, and D. Culler, "Versatile Low Power Media Access for Wireless Sensor Networks," in *ACM SenSys 2004*, November 2004.

[27] M. J. Miller, "Minimizing Energy Consumption in Sensor Networks Using a Wakeup Radio," Master's thesis, University of Illinois at Urbana-Champaign, November 2003.

[28] M. J. Miller and N. H. Vaidya, "A MAC Protocol to Reduce Sensor Network Energy Consumption Using a Wakeup Radio," *IEEE Transactions on Mobile Computing*, vol. 4, pp. 228–242, May/June 2005.

[29] M. J. Miller and N. H. Vaidya, "Minimizing Energy Consumption in Sensor Networks Using a Wakeup Radio," in *IEEE WCNC 2004*, March 2004.

[30] D. V. Hoffman, "Essential Wireless Hacking Tools." `http://tinyurl.com/89wsx` (`http://www.ethicalhacker.net/content/view/16/24/`).

[31] Wi-Foo - The Secrets of Wireless Hacking, "Recon and Attack tools." `http://www.wi-foo.com/index-3.html`.

[32] S. Fluhrer, I. Mantin, and A. Shamir, "Weaknesses in the Key Scheduling Algorithm of RC4," in *Workshop on Selected Areas in Cryptography*, vol. 2259 of *Lecture Notes in Computer Science*, August 2001.

[33] AirSnort. `http://airsnort.shmoo.com/`.

[34] aircrack. `http://freshmeat.net/projects/aircrack/`.

[35] C. E. Jones, K. M. Sivalingam, P. Agrawal, and J. C. Chen, "A Survey of Energy Efficient Network Protocols for Wireless Networks," *ACM Wireless Networks*, July 2001.

[36] V. Raghunathan, C. Schurgers, S. Park, and M. B. Srivastava, "Energy-Aware Wireless Microsensor Networks," *IEEE Signal Processing Magazine*, March 2002.

[37] A. J. Goldsmith and S. B. Wicker, "Design Challenges for Energy-Constrained Ad Hoc Wireless Networks," *IEEE Wireless Communications*, pp. 8–27, August 2002.

[38] A. Ephremides, "Energy Concerns in Wireless Networks," *IEEE Wireless Communications*, August 2002.

[39] J. Elson and K. Römer, "Wireless Sensor Networks: A New Regime for Time Synchronization," in *ACM Hot Topics in Networks (HotNets) 2002*, October 2002.

[40] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi, "The Flooding Time Synchronization Protocol," in *ACM SenSys 2004*, November 2004.

[41] R. Zheng, J. C. Hou, and L. Sha, "Asynchronous Wakeup for Ad Hoc Networks," in *ACM MobiHoc 2003*, June 2003.

[42] Y.-C. Tseng, C.-S. Hsu, and T.-Y. Hsieh, "Power-Saving Protocols for IEEE 802.11-Based Multi-Hop Ad Hoc Networks," in *IEEE Infocom 2002*, June 2002.

[43] O. Dousse, P. Mannersalo, and P. Thiran, "Latency of Wireless Sensor Networks with Uncoordinated Power Saving Mechanisms," in *ACM MobiHoc 2004*, May 2004.

[44] M. J. McGlynn and S. A. Borbash, "Birthday Protocols for Low Energy Deployment and Flexible Neighbor Discovery in Ad Hoc Wireless Networks," in *ACM MobiHoc 2001*, October 2001.

[45] W. Ye, J. Heidemann, and D. Estrin, "An Energy-Efficient MAC Protocol for Wireless Sensor Networks," in *IEEE Infocom 2002*, June 2002.

[46] V. Rajendran, K. Obraczka, and J. J. Garcia-Luna-Aceves, "Energy-Efficient, Collision-Free Medium Access Control for Wireless Sensor Networks," in *ACM SenSys 2003*, November 2003.

[47] M. L. Sichitiu, "Cross-Layer Scheduling for Power Efficiency in Wireless Sensor Networks," in *IEEE Infocom 2004*, March 2004.

[48] M. J. Miller and N. H. Vaidya, "On-Demand TDMA Scheduling for Energy Conservation in Sensor Networks," tech. rep., University of Illinois at Urbana-Champaign, June 2004.

[49] C. Guo, L. C. Zhong, and J. M. Rabaey, "Low Power Distributed MAC for Ad Hoc Sensor Radio Networks," in *IEEE GlobeCom 2001*, November 2001.

[50] E. Shih, P. Bahl, and M. J. Sinclair, "Wake on Wireless: An Event Driven Energy Saving Strategy for Battery Operated Devices," in *ACM MobiCom 2002*, September 2002.

[51] J. M. Rabaey, M. J. Ammer, J. L. da Silva Jr., D. Patel, and S. Roundy, "PicoRadio Supports Ad Hoc Ultra-Low Power Wireless Networking," *IEEE Computer*, July 2000.

[52] J. M. Rabaey, J. Ammer, T. Karalar, S. Li, B. Otis, M. Sheets, and T. Tuan, "PicoRadios for Wireless Sensor Networks — The Next Challenge in Ultra-Low Power Design," in *IEEE International Solid-State Circuits Conference (ISSCC) 2002*, February 2002.

[53] L. C. Zhong, J. Rabaey, C. Guo, and R. Shah, “Data Link Layer Design for Wireless Sensor Networks,” in *IEEE MILCOM 2001*, October 2001.

[54] L. Gu and J. A. Stankovic, “Radio-Triggered Wake-Up Capability for Sensor Networks,” in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) 2004*, May 2004.

[55] C. F. Chiasserini and R. R. Rao, “Combining Paging with Dynamic Power Management,” in *IEEE Infocom 2001*, April 2001.

[56] X. Yang and N. H. Vaidya, “A Wakeup Scheme for Sensor Networks: Achieving balance between energy saving and end-to-end delay,” in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) 2004*, May 2004.

[57] C. S. Raghavendra and S. Singh, “PAMAS – Power Aware Multi-Access protocol with Signalling for Ad Hoc Networks,” *ACM Computer Communications Review*, July 1998.

[58] A. El-Hoiydi and J.-D. Decotignie, “WiseMAC: An Ultra Low Power MAC Protocol for Multi-hop Wireless Sensor Networks,” in *Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS) 2004*, July 2004.

[59] H. Woesner, J.-P. Ebert, M. Schläger, and A. Wolisz, “Power-Saving Mechanisms in Emerging Standards for Wireless LANs: The MAC Level Perspective,” *IEEE Personal Communications*, pp. 40–48, June 1998.

[60] E.-S. Jung and N. H. Vaidya, “An Energy Efficient MAC Protocol for Wireless LANs,” in *IEEE Infocom 2002*, June 2002.

[61] E.-S. Jung and N. H. Vaidya, “Improving IEEE 802.11 Power Saving Mechanism.” in submission, 2004.

[62] J.-M. Choi, Y.-B. Ko, and J.-H. Kim, “Enhanced Power Saving Scheme for IEEE 802.11 DCF based Wireless Networks,” in *IFIP Personal Wireless Communication (PWC) 2003*, September 2003.

[63] T. van Dam and K. Langendoen, “An Adaptive Energy-Efficient MAC Protocol for Wireless Sensor Networks,” in *ACM SenSys 2003*, November 2003.

[64] Y. Li, W. Ye, and J. Heidemann, “Energy and Latency Control in Low Duty Cycle MAC Protocols,” in *IEEE WCNC 2005*, March 2005.

[65] G. Lu, B. Krishnamachari, and C. S. Raghavendra, “An Adaptive Energy-Efficient and Low-Latency MAC for Data Gathering in Wireless Sensor Networks,” in *IEEE IPDPS 2004*, April 2004.

[66] R. Zheng and R. Kravets, "On-demand Power Management for Ad Hoc Networks," in *IEEE Infocom 2003*, April 2003.

[67] C. Sengul and R. Kravets, "TITAN: On-Demand Topology Management in Ad Hoc Networks," *ACM Mobile Computing and Communications Review (MC2R)*, vol. 9, pp. 77–82, January 2005.

[68] C. Hu and J. Hou, "LISP: A Link-Indexed Statistical Traffic Prediction Approach to Improving IEEE 802.11 PSM," in *IEEE International Conference on Distributed Computing Systems (ICDCS) 2004*, March 2004.

[69] O. Chipara, C. Lu, and G.-C. Roman, "Efficient Power Management based on Application Timing Semantics for Wireless Sensor Networks," in *IEEE ICDCS 2005*, June 2005.

[70] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris, "Span: An Energy-Efficient Coordination Algorithm for Topology Maintenance in Ad Hoc Wireless Networks," in *ACM MobiCom 2001*, July 2001.

[71] Y. Xu, J. Heidemann, and D. Estrin, "Geography-informed Energy Conservation for Ad Hoc Routing," in *ACM MobiCom 2001*, July 2001.

[72] Y. Xu, J. Heidemann, and D. Estrin, "Adaptive Energy-Conserving Routing for Multihop Ad Hoc Networks," Tech. Rep. 527, USC/Information Sciences Institute, 2000.

[73] N. Reijers and K. Langendoen, "Efficient Code Distribution in Wireless Sensor Networks," in *ACM WSNA 2003*, September 2003.

[74] T. Stathopoulos, J. Heidemann, and D. Estrin, "A Remote Code Update Mechanism for Wireless Sensor Networks," Tech. Rep. CENS-TR-30, University of California, Los Angeles, Center for Embedded Networked Computing, 2003.

[75] P. Levis, N. Patel, D. Culler, and S. Shenker, "Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks," in *USENIX/ACM Networked System Design and Implementation (NSDI) 2004*, March 2004.

[76] J. W. Hui and D. Culler, "The Dynamic Behavior of a Data Dissemination Protocol for Network Programming at Scale," in *ACM SenSys 2004*, November 2004.

[77] D. B. Johnson and D. A. Maltz, "Dynamic Source Routing in Ad Hoc Wireless Networks," in *Mobile Computing* (T. Imielinski and H. Korth, eds.), ch. 5, pp. 153–181, Kluwer Academic Publishers, 1996.

[78] C. E. Perkins and E. M. Royer, "Ad-Hoc On Demand Distance Vector Routing," in *IEEE WMCSA 1999*, February 1999.

[79] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks," in *ACM MobiCom 2000*, August 2000.

[80] P. Sinha, R. Sivakumar, and V. Bharghavan, "CEDAR: A Core-Extraction Distributed Ad Hoc Routing Algorithm," in *IEEE Infocom 1999*, March 1999.

[81] P. Sinha, R. Sivakumar, and V. Bharghavan, "Enhancing Ad Hoc Routing with Dynamic Virtual Infrastructures," in *IEEE Infocom 2001*, April 2001.

[82] Y. Wang, W. Wang, and X.-Y. Li, "Distributed Low-Cost Backbone Formation for Wireless Ad Hoc Networks," in *ACM MobiHoc 2005*, May 2005.

[83] Z. J. Haas, J. Y. Halpern, and L. Li, "Gossip-Based Ad Hoc Routing," in *IEEE Infocom 2002*, June 2002.

[84] Y. Sasson, D. Carin, and A. Schiper, "Probabilistic Broadcast for Flooding in Wireless Mobile Ad Hoc Networks," in *IEEE WCNC 2003*, March 2003.

[85] R. Friedman, M. Gradinariu, and G. Simon, "Locating Cache Proxies in MANETs," in *ACM MobiHoc 2004*, May 2004.

[86] S.-Y. Ni, Y.-C. Tseng, Y.-S. Chen, and J.-P. Sheu, "The Broadcast Storm Problem in a Mobile Ad Hoc Network," in *ACM MobiCom 1999*, August 1999.

[87] Kerberos: The Network Authentication Protocol. `http://web.mit.edu/kerberos/`.

[88] A. Perrig, R. Szewczyk, J. D. Tygar, V. Wen, and D. E. Culler, "SPINS: Security Protocols for Sensor Networks," *Wireless Networks (WINET)*, vol. 8, September 2002.

[89] L. Eschenauer and V. D. Gligor, "A Key-Management Scheme for Distributed Sensor Networks," in *ACM Computer and Communications Security (CCS) 2002*, November 2002.

[90] H. Chan, A. Perrig, and D. Song, "Random Key Predistribution Schemes for Sensor Networks," in *IEEE Security and Privacy Symposium 2003*, May 2003.

[91] S. Zhu, S. Xu, S. Setia, and S. Jajodia, "Establishing Pairwise Keys for Secure Communication in Ad Hoc Networks: A Probabilistic Approach," in *IEEE International Conference on Network Protocols (ICNP) 2003*, November 2003.

[92] J. Newsome, E. Shi, D. Song, and A. Perrig, "The Sybil Attack in Sensor Networks: Analysis & Defenses," in *ACM ISPN 2004*, April 2004.

[93] W. Du, J. Deng, Y. S. Han, and P. K. Varshney, "A Pairwise Key Pre-distribution Scheme for Wireless Sensor Networks," in *ACM Computer and Communications Security (CCS) 2003*, October 2003.

[94] R. Blom, "Non-Public Key Distribution," in *Advances in Cryptology - CRYPTO 1982*, August 1982.

[95] D. Liu and P. Ning, "Establishing Pairwise Keys in Distributed Sensor Networks," in *ACM Computer and Communications Security (CCS) 2003*, October 2003.

[96] C. Blundo, A. D. Santis, A. Herzberg, S. Kutten, U. Vaccaro, and M. Yung, "Perfectly-Secure Key Distribution for Dynamic Conferences," in *Advances in Cryptology - CRYPTO 1992*, August 1992.

[97] S. Zhu, S. Setia, and S. Jajodia, "LEAP: Efficient Security Mechanisms for Large-Scale Distributed Sensor Networks," in *ACM Computer and Communications Security (CCS) 2003*, October 2003.

[98] Diffie-Hellman key exchange. `http://en.wikipedia.org/wiki/Diffie-Hellman`.

[99] Elliptic Curve Diffie-Hellman. `http://en.wikipedia.org/wiki/Elliptic_Curve_Diffie-Hellman`.

[100] D. J. Malan, M. Welsh, and M. D. Smith, "A Public-Key Infrastructure for Key Distribution in TinyOS Based on Elliptic Curve Cryptography," in *IEEE SECON 2004*, October 2004.

[101] V. Gupta, M. Millard, S. Fung, and Y. Zhu, "Sizzle: A Standards-based end-to-end Security Architecture for the Embedded Internet," in *IEEE PerCom 2005*, March 2005.

[102] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid, "Recommendation for Key Management – Part 1: General." National Institute of Standards and Technology (NIST) Special Publication 800-57, August 2005.

[103] `ns-2` – The Network Simulator. `http://www.isi.edu/nsnam/ns`.

[104] MICA2 Mote Datasheet. `http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/6020-0042-05_A_MICA2.pdf`.

[105] TR1000 Hybrid Transceiver Datasheet. `http://www.rfm.com/products/data/tr1000.pdf`.

[106] R. Szewczyk, J. Polastre, A. Mainwaring, and D. Culler, "Lessons from a Sensor Network Expedition," in *The European Workshop on Wireless Sensor Networks (EWSN) 2004*, January 2004.

[107] M. J. Miller and N. H. Vaidya, “Improving Power Save Protocols Using Carrier Sensing and Busy-Tones for Dynamic Advertisement Windows,” tech. rep., University of Illinois at Urbana-Champaign, December 2004.

[108] E.-S. Jung and N. H. Vaidya, “A Power Saving MAC Protocol for Wireless Networks,” tech. rep., University of Illinois at Urbana-Champaign, 2002.

[109] X. Yang and N. Vaidya, “On Physical Carrier Sensing in Wireless Ad Hoc Networks,” in *IEEE Infocom 2005*, March 2005.

[110] T. H. Cormen, C. E. Leiserson, , R. L. Rivest, and C. Stein, *Introduction to Algorithms.* The MIT Press, 2 ed., 2001.

[111] N. H. Vaidya, “A Case for Two-Level Distributed Recovery Schemes,” in *ACM SIGMETRICS 1995*, May 1995.

[112] R. Draves, J. Padhye, and B. Zill, “Comparison of Routing Metrics for Static Multi-Hop Wireless Networks,” in *ACM SIGCOMM 2004*, August–September 2004.

[113] Link-state routing protocol. `http://en.wikipedia.org/wiki/Link-state_routing_protocol`.

[114] P. Jacquet, P. Mühlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot, “Optimized Link State Routing Protocol,” in *IEEE International Multi Topic Conference (INMIC) 2001*, December 2001.

[115] D. Eppstein, “Finding the k Shortest Paths,” *SIAM Journal on Computing*, vol. 28, no. 2, pp. 652–673, 1998.

[116] S. Doshi and T. X. Brown, “Minimum Energy Routing Schemes for a Wireless Ad Hoc Network,” in *IEEE Infocom 2002*, June 2002.

[117] S. Singh, M. Woo, and C. S. Raghavendra, “Power-Aware Routing in Mobile Ad Hoc Networks,” in *ACM MobiCom 1998*, October 1998.

[118] J.-H. Chang and L. Tassiulas, “Energy Conserving Routing in Wireless Ad-hoc Networks,” in *IEEE Infocom 2000*, March 2000.

[119] B. Liang and Z. J. Haas, “Virtual Backbone Generation and Maintenance in Ad Hoc Network Mobility Management,” in *IEEE Infocom 2000*, March 2000.

[120] P.-J. Wan, K. M. Alzoubi, and O. Frieder, "Distributed Construction of Connected Dominating Set in Wireless Ad Hoc Networks," in *IEEE Infocom 2002*, June 2002.

[121] G. R. Grimmet and A. M. Stacey, "Critical probabilities for site and bond percolation models," *Annals of Probability*, vol. 26, no. 4, pp. 1788–1812, 1998.

[122] J. Polastre, R. Szewczyk, and D. Culler, "Telos: Enabling Ultra-Low Power Wireless Research," in *ACM/IEEE IPSN/SPOTS*, April 2005.

[123] Chipcon CC1000 Datasheet. `http://www.chipcon.com/files/CC1000_Data_Sheet_2_1.pdf`.

[124] A. Perrig, R. Canetti, J. D. Tygar, and D. Song, "Efficient and Secure Source Authentication for Multicast," in *ISOC NDSS 2001*, February 2001.

[125] A. D. Wood and J. A. Stankovic, "Denial of Service in Sensor Networks," *IEEE Computer*, vol. 35, October 2002.

[126] A. Broder and M. Mitzenmacher, "Network Applications of Bloom Filters: A Survey," *Internet Mathematics*, vol. 1, no. 4, pp. 485–509, 2004.

[127] G. Tsudik, "Message Authentication with One-Way Hash Functions," in *IEEE Infocom 1992*, May 1992.

[128] M. J. Miller and N. H. Vaidya, "Leveraging Channel Diversity for Key Establishment in Wireless Sensor Networks." extended version of *Infocom 2006* paper, 2005.

[129] W. Du, R. Wang, and P. Ning, "An Efficient Scheme for Authenticating Public Keys in Sensor Networks," in *ACM MobiHoc 2005*, May 2005.

[130] T. S. Rappaport, *Wireless Communications: Principles and Practice*. Prentice Hall PTR, 2 ed., 2002.

[131] A. Perrig, R. Canetti, J. D. Tygar, and D. Song, "Efficient Authentication and Signing of Multicast Streams over Lossy Channels," in *IEEE Security and Privacy Symposium 2000*, May 2000.

[132] Y.-C. Hu, A. Perrig, and M. Sirbu, "SPV: Secure Path Vector Routing for Securing BGP," in *ACM SIGCOMM 2004*, August–September 2004.

[133] Binomial distribution. `http://en.wikipedia.org/wiki/Binomial_distribution`.

[134] A. Cerpa, J. L. Wong, M. Potkonjak, and D. Estrin, "Temporal Properties of Low Power Wireless Links: Modeling and Implications on Multi-Hop Routing," in *ACM MobiHoc 2005*, May 2005.

[135] S. Zhu, S. Setia, S. Jajodia, and P. Ning, "An Interleaved Hop-by-Hop Authentication Scheme for Filtering Injected False Data in Sensor Networks," in *IEEE Security and Privacy Symposium 2004*, May 2004.

[136] F. Ye, H. Luo, S. Lu, and L. Zhang, "Statistical En-route Filtering of Injected False Data in Sensor Networks," in *IEEE Infocom 2004*, March 2004.

[137] J. So and N. Vaidya, "Multi-Channel MAC for Ad Hoc Networks: Handling Multi-Channel Hidden Terminals Using A Single Transceiver," in *ACM MobiHoc 2004*, May 2004.

[138] P. Kyasanur and N. H. Vaidya, "Routing and Interface Assignment in Multi-Channel Multi-Interface Wireless Networks," in *IEEE WCNC 2005*, March 2005.

[139] P. Bahl, R. Chandra, and J. Dunagan, "SSCH: Slotted Seeded Channel Hopping for Capacity Improvement in IEEE 802.11 Ad-Hoc Wireless Networks," in *ACM MobiCom 2004*, September 2004.

[140] F. K. Hwang, D. S. Richards, and P. Winter, *The Steiner Tree Problem.* Elsevier Science Publishers, 1992.

[141] P. Crescenzi and V. Kann, "A compendium of NP optimization problems." `http://www.nada.kth.se/~viggo/problemlist/compendium.html`.

Author's Biography

Matthew J. Miller was born near Columbus, Ohio in 1978. He later moved to Amarillo, Texas for three years and then to Anderson, South Carolina. Here he graduated from Westside High School in 1996 and then attended Clemson University. Matthew received the BS degree, summa cum laude, from Clemson in May 2001 with a major in Computer Engineering and minors in Computer Science and Mathematics.

In August 2001, Matthew began his graduate studies in the Computer Science department at the University of Illinois at Urbana-Champaign (UIUC). He was a 2001 recipient of the National Science Foundation Fellowship and the ASEE National Defense Science and Engineering Graduate Fellowship. Matthew completed the MS degree at UIUC in December 2003 under the supervision of Prof. Nitin H. Vaidya. His MS thesis was entitled *Minimizing Energy Consumption in Sensor Networks Using a Wakeup Radio.*

Matthew married his wife, Leigh Ann, in June 2002. In their free time, they enjoy ultimate frisbee, running, working out, reading, and board games. Additionally, they have been actively involved in churches and Christian organizations.