CROSS-LAYER DESIGNS FOR ENERGY-SAVING SENSOR AND AD HOC NETWORKS

BY

MATTHEW JEFFERSON MILLER

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

PROPOSAL

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, 2005

Urbana, Illinois

CROSS-LAYER DESIGNS FOR
ENERGY-SAVING SENSOR AND AD HOC NETWORKS

Matthew Jefferson Miller, Ph.D.
Department of Computer Science
University of Illinois at Urbana-Champaign, 2005
Nitin H. Vaidya, Adviser

Energy consumed by the the radio interface of resource-limited devices, such as sensors, is a major source of the overall energy usage of the device. By placing the radio interface in a sleep state where no communication is possible, the radio reduces its power level by several orders of magnitude when compared to keeping the radio in a listen state. Thus, it is desirable to place the radio interface in the sleep state as much as possible without greatly hindering the performance required by applications.

Most previous work on energy-saving protocols is primarily isolated to one layer and, thus, consider neither the benefits of cross-layer interaction within this domain nor the effects of energy-saving on traditional wireless network primitives such as information propagation via the broadcast medium. In this work, we explore protocols that use cross-layer designs for energy saving and some cross-layer tradeoffs that arise as a result of reducing energy consumption.

Table of Contents

Chapter 1

Introduction

As many new applications emerge as the result of wireless networking technology and research, it is important to consider design decisions for these devices. In particular, some of these designs may differ substantially from the protocols already in place for wireline networks. For example, while the strict layering of the network stack is generally regarded as a favorable design decision for wireline networks, wireless networks may require more flexibility in interactions between layers of the network stack.

While there are many possible interactions between layers to consider, our work focuses on cross-layer design for saving energy in wireless networks. 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, which 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 only increased by 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-type performance 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 also requires research in the areas of architecture, operating systems, and application design [9]. Though our design techniques are applicable to multihop ad hoc networks in general, we note that our work is particularly beneficial for devices with no display (e.g., sensors) or very small displays (e.g., cell phones). As shown in

Table 1.1: Improvement of various laptop components between 1990 and 2001 [6–8].

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 ×

Table 1.2: Fraction of energy used by device components for different hardware and traffic [10].

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

Table 1.2, the wireless interface energy consumption of such devices can account for over 60% of the device's overall energy usage.

Wireless interfaces typically 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 [11], 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.

Motivated by the large reduction in energy consumption that is possible from entering the sleep state, we focus on *power save protocols*. While there has been a large amount of research in this area [2, 12–29], most protocols operate in isolation at one layer (usually around the link layer) without considering benefits that can be gained by allowing interaction between layers. In Chapter 2, we discuss these protocols in detail and discuss the cross-layer designs used by some of these protocols.

Another aspect of cross-layer design for energy-efficient wireless networks is investigating the effects of energy-saving on traditional wireless network primitives. In particular, we look at information propagation via the broadcast medium of wireless networks to see the tradeoffs in application performance that result from energy-saving protocols.

Table 1.3: Characteristics of a Mica2 Mote radio [11].

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

1.1 Proposal Outline

In Chapter 2, we look at related work related to our thesis. In Section 2.1, we discuss power save protocols and broadly classify them into four categories: *synchronous* (Section 2.1.1), *asynchronous* (Section 2.1.2), *out-of-band* (Section 2.1.3), and cross-layer designs (Section 2.1.4). In Section 2.2, we give an overview of work related to the broadcast applications which are affected by energy-saving protocols: efficiently propagating data via multihop broadcasts (Section 2.2.1) and symmetric key distribution (Section 2.2.2).

In Chapter 3, we propose cross-layer protocol designs which take advantage of interaction between the link layer and physical layer. 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 available at the physical layer. We demonstrate how this cross-layer technique can be used to augment both synchronous and out-of-band protocols. We then take advantage of carrier sensing to dynamically adjust the listening time for wake-up signal detection.

Chapter 4 explores a novel cross-layer design with interaction between the link layer and routing layer. We consider using multiple levels of power save protocols at the link layer. Each level presents a different tradeoff in terms of energy and latency (i.e., levels with an increased energy consumption have a lower latency). We explore a routing layer protocol design that discovers routes which allow the nodes to consume as little energy as possible while maintaining a user-specified latency.

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 between energy, latency, and reliability. By simulation, we characterize this tradeoff.

In Chapter 6, we develop a protocol for symmetric key distribution that uses one-hop broadcasts. The protocol distributes keys in a manner that is resilient to attackers and provides good connectivity while requiring only a small amount of memory. After describing this protocol, we characterize the cross-layer effects that arise from using a power save protocol during the key distribution by considering the tradeoff between security and energy consumption. Chapter 7 summarizes the proposal.

Chapter 2

Related Work

In this chapter, we discuss related work for saving power in ad hoc and sensor networks. Our survey includes both cross-layer and single-layer protocols. Additionally, we review related work to the applications discussed in Chapter 4 and Chapter 6.

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?* We categorize the many approaches to this problem into four sections. Note that this categorization is not intended to be mutually exclusive; some works could easily fit into more than one category. We try to categorize the works with the most similarity.

Synchronous Protocols (Section 2.1.1): 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 [30] 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 [31, 32] is where two communicating nodes wake up T seconds after the last packet reception and T is adjusted dynamically based on traffic patterns.

Asynchronous Protocols (Section 2.1.2): Nodes wake up independently according to their own schedule and try to discover other nodes that are currently awake. When the wake-ups of two nodes overlap, they can communicate. An example [20, 21] is choosing deterministic schedules to guarantee overlap within a bounded latency. Another example [18, 33] is when nodes wake up non-deterministically such that overlap is within a bounded time with high probability.

Out-of-Band Protocols (Section 2.1.3): A node's data radio sleeps until an out-of-band channel alerts them to wake up. An example [22, 34] is a low-power radio idly listening on a separate, wake-up channel. Another example [1, 2] 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.

Cross-Layer Protocols (Section 2.1.4): We separately categorize approaches which use a cross-layer design in their power save approach.

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 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 [31, 32] for an example of this combination of techniques).

We note that the focus of this thesis is on power save protocols to reduce idle listening energy. There is a vast area of research 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 Synchronous Protocols

Many of the protocols in this section assume 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] even claim precision on the order of *a microsecond*. For the purposes of our work, we assume that some such external mechanism is available.

We begin by describing 802.11 PSM [30]. In Chapter 3, we augment 802.11 PSM with a cross-layer design to reduce its energy consumption. 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

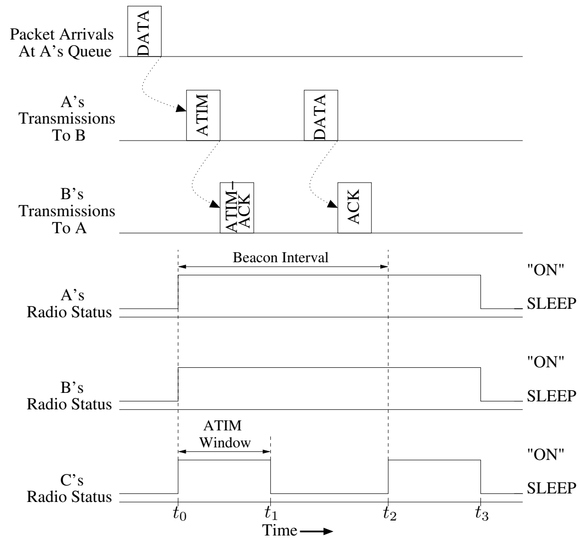

Figure 2.1: IEEE 802.11 IBSS power save mode [30].

nodes will remain on after the ATIM window and attempt 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 [30]. 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.

In [41], 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 cross-layer design in Chapter 3 that dynamically adjusts the ATIM window.

Other works have also proposed dynamic ATIM window adjustment. DPSM [13] 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 [14] is similar to our work in that the ATIM window ends when the channel is idle for a specified amount of time. However, IPSM only works in 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 [15], 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 only uses static ATIM window sizes whereas our techniques allows dynamic adjustment of the window.

S-MAC [42] 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 wakeups. A neighborhood 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 nodes waking up in its vicinity will sense the transmission.

T-MAC [25] 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 [43, 44], modifications are made to S-MAC to reduce the multihop delay of packet forwarding. Also in [43], a global scheduling algorithm is developed for S-MAC to converge to one sleep schedule in the network. The carrier sensing techniques in could be used to complement the S-MAC protocols.

In [31,32], synchronous wake-ups are added to an out-of-band protocol. Nodes engaged in communication schedule times in the future to wake up based on past traffic patterns.

TRAMA [19] uses TDMA to schedule queued packets. The TDMA scheduling is performed 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.

2.1.2 Asynchronous Protocols

An example of an asynchronous protocol [21] is nodes choosing their awake times such that they are guaranteed to overlap with each neighbor's awake time within a bounded time period. In this work, three protocols are proposed which 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 a lot of energy. The second approach requires the nodes to only stay awake for a long active interval once every T beacon intervals. During the other $T-1$ intervals, the node will only wakeup for 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 only stay awake for an advertisement window. They cite a proof that guarantees this scheme will allow unsynchronized hosts to overlap. 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 [20], a deterministic protocol for neighbor discovery is presented. Using combinatoric theory, 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 is reduced and all neighbors should be able to contact each other within Y slots to synchronize their communication.

In [33], a non-deterministic approach is used for neighbor discovery. Nodes wake up probabilistically in each slot and can only communicate with other nodes that are also on in that slot. The basic idea is a nondeterministic protocol where a node is awake for a 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 the time interval.

The protocol in [18] 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 have received the packet with high probability.

The idea of preamble sampling has been used with B-MAC [16]. 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 [17] 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, there are some key differences, as discussed in Section 3.1.5.

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	$O(100)$ bps	50 kbps
Range (m)	10	10
Transition Energy, sleep→idle	—	$1\,\mu\text{s} \times 1\,\text{mW}$
Transition Energy, idle→sleep	—	$1\,\mu\text{s} \times 0\,\text{mW}$

2.1.3 Out-of-Band Protocols

Examples of out-of-band protocols include PicoRadio [22, 45–47] which uses a low-power hardware device is proposed to serve as a wake-up channel with a low idle listening cost. A MAC protocol has been designed which allows nodes to wakeup 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 wakeup channel. The nodes then communicate over the high powered data channel of the receiver. This design uses a CDMA scheme which 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 wakeup signal, which increases the hardware complexity. Table 2.1 shows the target specifications for the PicoRadio hardware.[1]

Similar to PicoRadio, in [48], a hardware design for a wake-up radio is presented using circuit simulations. A wakeup channel is also used in [34]. 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 channel. This will cause the high powered radio to turn on so data communication can begin. However, this protocol is designed for systems with centralized access points or proxies.

In [49], paging interfaces are used so a base station can wake up certain nodes when it has data to send. Here a base station uses RF ID tags to wakeup devices which 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. The idea that is a device will remain in a 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

[1] These values were obtained in an email correspondence with Brian Otis, the lead graduate student for the hardware design.

sleep state. The non-target devices in the paged sleep state will then start the sleep cycle again once they determine 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.

STEM and STEM-BT [1,2] are also out-of-band wake-up protocols. In Chapter 3, we demonstrate how our cross-layer techniques can be applied to these protocol. Thus, we defer a detailed description of these protocols to Section 3.2.1.

The PAMAS protocol [50] adapts basic mechanisms of IEEE 802.11 [30] 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.

2.1.4 Cross-Layer Protocols

Protocols in this section use cross-layer techniques in that the power save protocol makes sleep decisions based on a past history of the multihop traffic in the network.

In [27], 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 which reduces latency and preserves throughput better than only using 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. TITAN [51] extends the work from [27]. 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.

Other work uses TDMA techniques schedule "flows" of data packets [26,52] where periodic flows attempt 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 [26] and [52], 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.

LISP [28] is an extension to 802.11 PSM where nodes attempt 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 overhead ATIM-ACKs and previous ATIM/ATIM-ACK handshakes.

In [29], ESSAT is designed to handle 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.

2.2 Broadcast Applications

In this section we look at two applications that are affected by the use of energy-saving protocols. We investigate these cross-layer effects that are induced by the power save protocols. In this section, we cover work related to these two applications. The first is efficiently propagating information by broadcasting over multiple hops (Section 2.2.1), which is the focus of Chapter 5. The second application, covered in Chapter 6, is broadcast for symmetric key distribution in sensor networks (Section 2.2.2).

2.2.1 Efficient Information Propagation via Multihop Broadcast

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 [53], the authors demonstrate a software architecture to allow the application of such updates. In other works [54–56], 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 [57, 58] and querying for sensor data [59].

One popular method for reducing the overhead of broadcast is to form a backbone in the network where only certain nodes forward data [60–62], which can reduce overhead. Another method, which is most similar to our work in Chapter 5, is probabilistic broadcast [63–65], 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 there is typically a high level of redundancy in a broadcast flood [66]. In our protocol, we attempt to use this redundancy to reduce the *energy* consumed by the broadcast.

2.2.2 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 terms of 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 [67], the SPINS security architecture for sensor networks is presented. It includes a key establishment protocol that requires two sensors to get a pairwise key from a trusted server with which both of the sensors share a secret key. A disadvantage of this approach is that the server may become a bottleneck in large networks. The LEAP architecture [68] provides a method of establishing pairwise keys provided sensors cannot be compromised during a short initialization phase after deployment and the sensor hardware can be trusted to completely erase keying material after initialization.

Eschenauer and Gligor [69] were among the first to consider key predistribution for sensor networks. In their work, referred to as the basic scheme, sensors are loaded with randomly chosen keys out of a master key pool prior to 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. [70] 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 the form of diverse path selection. Other schemes propose that keys be distributed deterministically based on a sensor's ID [71, 72].

Du et al. [73] adapt a key predistribution scheme originally proposed by Blom [74] for sensor networks by using finite fields to generate multiple key spaces that can be randomly deployed to sensors. Liu et al. [75] extend a key distribution method proposed by Blundo et al. [76] that uses polynomial based distribution methods.

The work that is most similar to ours is that of Anderson et al. [5]. The protocol is based on the assumption that the number of adversary devices in the network at the time of key establishment is very 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 Chapter 6, 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 utilize 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 Chapter 6 for more detail 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 [5], unlike our work, is not to make it difficult for a nearby attacker to compromise a link nor to operate in hostile environments where there may be a large number of adversaries.

Chapter 3

Cross-Layer Protocol Design at the Link Layer and Physical Layer

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 synchronous protocol (Section 3.1.2) as well as out-of-band protocols (Section 3.2.5 and Section 3.2.6).

Next, we observe that physical layer carrier sensing can also be used to dynamically adjust the time that nodes remain on when packets are being transmitted. This technique is applicable to any power save protocol that has groups of nodes wake up at certain times and remain on as long as they are involved in data communication. We demonstrate how this technique can be applied to a synchronous protocol (Section 3.1.3).

3.1 Using Carrier Sensing for Dynamic Advertisement Windows

In this section, we further discuss our proposed techniques to leverage physical layer carrier sensing for energy efficiency and demonstrate their application to a synchronous power save protocol. Specifically, we look at techniques to improve the IBSS *Power Save Mode* (PSM) in IEEE 802.11 [30]. IBSS (Independent

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

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.

The major contributions of this work are:

- We show how carrier sensing can be used to determine if it is necessary to listen for traffic advertisements. This allows us to avoid listening for long periods when no packets will be advertised.
- We dynamically re-size the ATIM window based on the number of advertisements to be sent in the current window. While we have explored dynamic adjustment of the ATIM window previously [13, 77], this is the first work of which we are aware that achieves this in a multi-hop environment using a single channel.

3.1.1 Protocol Design Discussion

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 [13, 28] 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 will not be able to 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 there is more data to advertise.

In Section 3.1.2, we use a short carrier sensing period preceding the ATIM window where nodes can indicate whether or not 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 3.1.3, 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. 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.

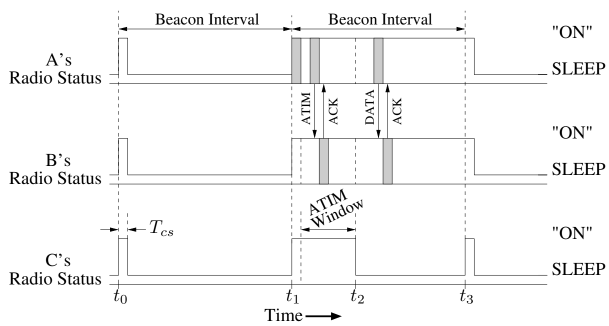

Figure 3.1: CS-ATIM protocol.

3.1.2 Technique 1: Carrier Sensing Preceding the ATIM Window

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 there is traffic 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) which 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 only wake up for T_{cs} at the beginning of every interval when there are no packets to be advertised in their neighborhood. When there are packets 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, there are no packets 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 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 [30], 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, it is clear that nodes can use significantly less energy than 802.11 PSM listening at the beginning of each beacon interval when there are no packets to be advertised. When there are packets to be advertised, CS-ATIM only uses slightly more energy than 802.11 PSM because of the short carrier sensing period. In terms of 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, there is a slightly larger delay in CS-ATIM 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, creates the risk that nodes may erroneously carrier sense energy that is due to interference in the frequency band rather than the transmission of a dummy packet. 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 3.1.5, the effects of false positives on CS-ATIM are tested.

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 there are no nodes with 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 there is no data to be sent.

3.1.3 Technique 2: Dynamic ATIM Window Adjustment

The CS-ATIM protocol 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 there is a small number of packets 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 which need to be transmitted and then the ATIM window should end right after the last ATIM-ACK is received.[2] This is what past work attempts to achieve either through heuristics [13] or dynamically extending the window when packets are received [14,77]. 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 [78]. 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 there are several cases in which 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.[3] 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, there are *two* carrier sensing periods that follow the beginning of the beacon interval:

- **CS**$_1$: As in CS-ATIM, DCS-ATIM begins with a carrier sensing period of length T_{cs} during which

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

[3] Currently, we do not test these situations in our simulations.

time nodes use the protocol from Section 3.1.2 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, $\mathbf{CS}_2$, (again 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 adjustment scheme described below. For example, a node may wish to 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_{data}). In the IEEE 802.11 specification [30] for direct-sequence spread spectrum (DSSS), the default CW_{data} 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 $CW_{aw} < CW_{data}$ 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_{data}.

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

$$\begin{aligned} T_{idle} = {} & DIFS + T_{slot} \cdot CW_{aw} + prop_{max} \\ & + T_{atim} + SIFS + prop_{max} + T_{ack} \\ & + DIFS + T_{slot} \cdot CW_{aw} + prop_{max} \end{aligned} \tag{3.1}$$

where $DIFS$ and $SIFS$ are the DCF and Short Interframe Space as specified by IEEE 802.11 [30], respectively. The values T_{atim} and T_{ack} are the time durations required to send an ATIM and ATIM-ACK, respectively.[4] The maximum propagation delay between two nodes is denoted as $prop_{max}$. T_{idle} is designed

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

to be 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 starving data packets, an upper limit is set on the size that the dynamic ATIM window can reach. Currently, this upper bound is 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 will end the ATIM phase and wait 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 beginning of the ATIM window. It is postponed until this time to avoid sending potentially long data packets while other neighbors are still trying to transmit ATIMs.

An example of DCS-ATIM compared to 802.11 PSM is given in Figure 3.2. First, there is the additional carrier sensing at the beginning of DCS-ATIM. 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 only uses 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 802.11 PSM if a data packet arrives at the node towards the end of 802.11 PSM's static ATIM window.

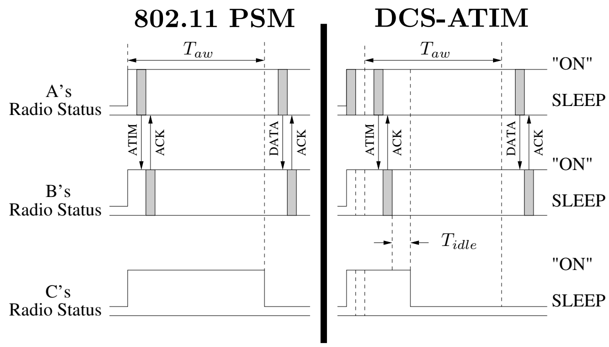

Figure 3.2: 802.11 PSM vs. DCS-ATIM.

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 have to 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. In this case, the node will have to wait until the next ATIM window to advertise the packet.

3.1.4 Comparison with Preamble Sampling

We note that the technique described in Section 3.1.2 is similar to the preamble sampling used in protocols such as B-MAC [16] and WiseMAC [17]. However, there are some key differences between our proposed carrier sensing technique and these preamble sampling protocols. The advantage of preamble sampling is that it works in completely unsynchronized environments.[5] However, because the nodes may transmit their preamble at any time (which serves as the wake-up signal), there is an increased chance 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 only serves as a binary indication of whether or not the channel is busy, interference among wake-up signals is tolerable, as discussed in Section 3.1.2.

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

[5] WiseMAC [17] 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.

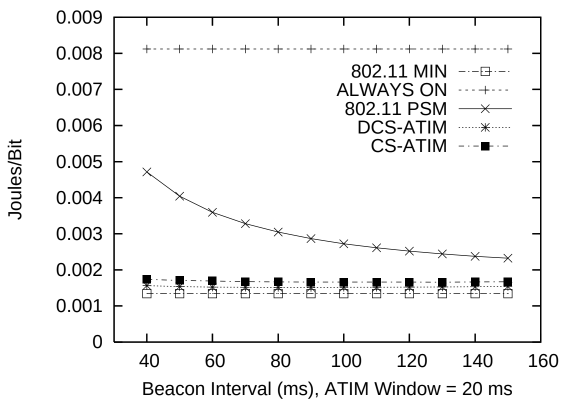

Figure 3.3: Energy vs. beacon interval.

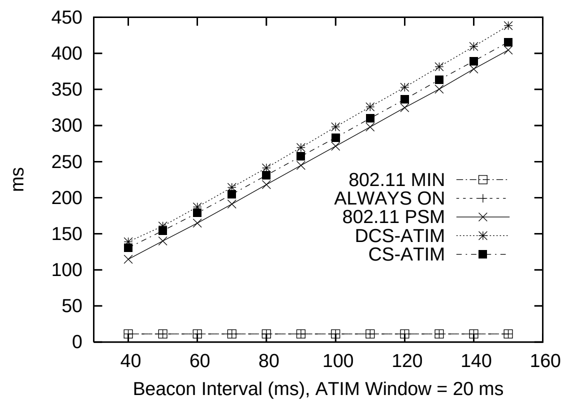

Figure 3.4: Latency vs. beacon interval.

carrier sense the channel (e.g., typically on the order of tens of microseconds) for both unicast and broadcast packets. Thus, the well-known overhead problem associated with broadcast storms [66] is exacerbated by the preamble sampling protocols whereas our protocol does not add any extra overhead to broadcast packets when compared to unicast packets.

Finally, when there are several senders transmitting to a receiver (e.g., in sensor networks when aggregation is performed or in ad hoc routing when a node's route replies attract multiple senders), then preamble sampling significantly increases latency beyond our proposed protocol. This is because only one sender can transmit a preamble for a receiver per beacon interval. By contrast, multiple senders can transmit data packets per beacon interval in our protocol.

3.1.5 Brief Presentation of Simulation Results

To test our protocols, we simulated them by modifying the MAC and physical layers of *ns-2* [79]. More extensive results and discussion is provided in [80]. We place 50 nodes uniformly at random in a 1000 m × 1000 m area. There are five flows sending 512 byte data packets at a rate of 1 kbps per flow. By default, the following values are used: the beacon interval length is 100 ms, T_{aw} is 20 ms, T_{cs} is 1 ms, and T_{idle} is set to 3.19 ms according to Equation 3.1.

From Figure 3.3 and Figure 3.4, we see that 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 while increasing the latency by only about 8 ms to 15 ms.

3.2 Using Carrier Sensing to Reduce the Listening Time for Out-of-Band protocols

In this section, we demonstrate how physical layer carrier sensing can be applied to an out-of-band power save protocol. An advantage of this technique is, 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, there are tradeoffs in 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 [1, 2] and STEM-BT [2] (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 and Discussion

In STEM [1, 2], 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 [2] 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, there are two sub-protocols. One is the *transmitting* sub-protocol, which is performed when a node has data to send and attempts to wake up the 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

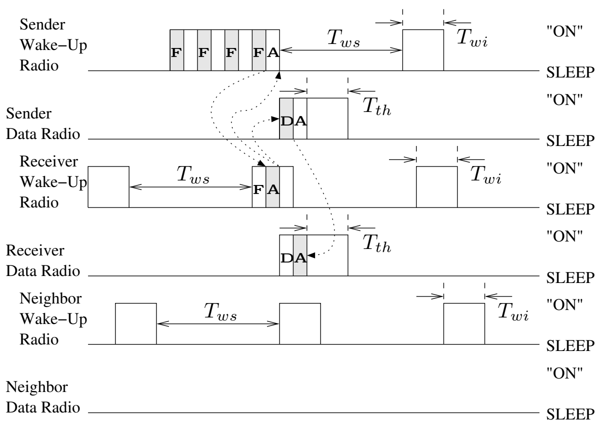

Figure 3.5: STEM protocol [1, 2].

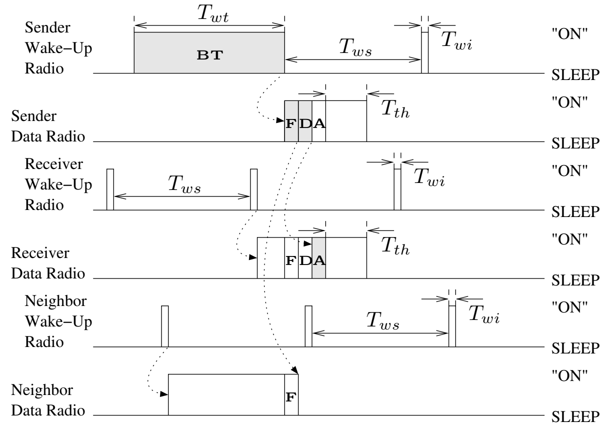

Figure 3.6: STEM-BT protocol [2].

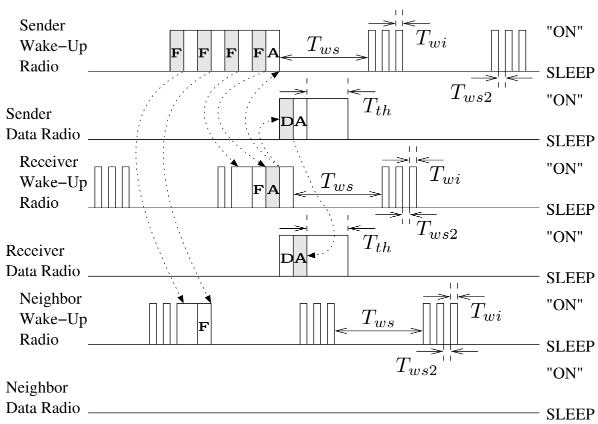

Figure 3.7: Proposed STEM-H protocol.

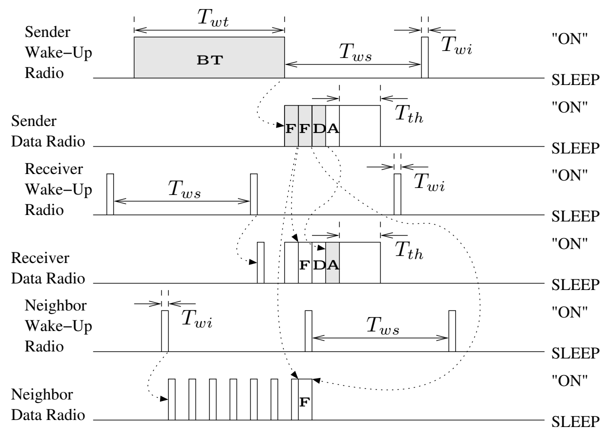

Figure 3.8: Proposed STEM-BT2 protocol.

sent and they need to wake up their data radio. In STEM and STEM-H, the wake-up radio must be capable of sending and receiving data packets. By contrast, in STEM-BT and STEM-BT2, the wake-up radio only needs to be able to send and detect a busy tone (i.e., making a binary decision whether the channel is busy or not). Figures 3.5, 3.6, 3.7, and 3.8 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 shaded area, a node is sending; otherwise, a node is receiving.

3.2.2 STEM [1, 2] (Figure 3.5)

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 brevity, we omit a discussion of how wake-up channel collisions are handled. 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 [1].

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 [2] (Figure 3.6)

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 only has to 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 observations. First, the wake-up process of STEM is relatively inexpensive (in terms of energy) for all nodes other than the sender when compared to 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.[6] 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 only transmits for $T_{wt}/2$ time on average. By contrast, the wake-up procedure for STEM-BT is relatively expensive compared to 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) there are a large number of neighbor nodes in the vicinity of the sender[7] 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 to 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 [11]). All nodes in the network must spend $\frac{T_{wi}}{T_{ws}+T_{wi}}$ fraction of the time idly listening even if there is no network traffic. 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 3.2.7, T_{wi} is about 80 times larger for STEM than for STEM-BT for our experimental 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.

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

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

3.2.5 Proposed Protocol: STEM-H (Figure 3.7)

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 (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 only idly listen 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 the length of T_{wt}, which guarantees sufficient overlap with STEM-H's monitoring protocol.

Monitoring Protocol For STEM-H's monitoring state, nodes only wake up 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 3.7, 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 3.2.7, STEM-H does no worse than STEM, in terms of 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: STEM-BT2 (Figure 3.8)

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.[8] 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 3.2.7, 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

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

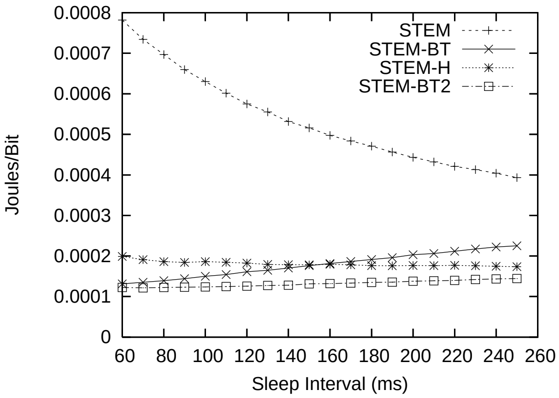

Figure 3.9: Energy consumption of the protocols.

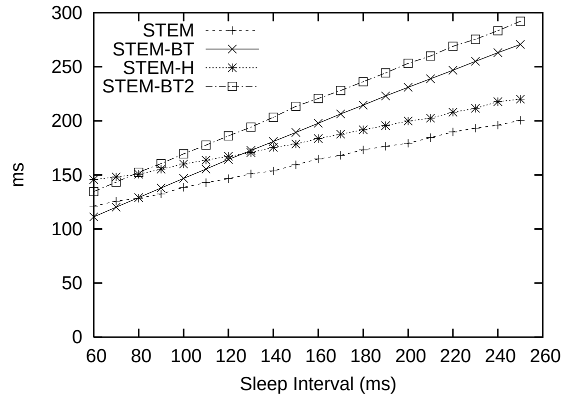

Figure 3.10: Latency of the protocols.

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 Brief Presentation of 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* [79]. Additionally, we did analysis on the expected energy consumed by each of the four protocols, which is presented in [81]. By default, we use $T_{ws} = 100\,\text{ms}$ and, for STEM-BT, STEM-H, and STEM-BT2, we set $T_{wi} = 1\,\text{ms}$ [82]. From Figure 3.9 and Figure 3.10, we see that STEM-H greatly reduces the energy consumption of STEM with a rather small increase in latency. However, STEM-BT2's gains over STEM-BT are rather disappointing, especially considering that they come at the cost of increased latency. Though, in Figure 3.11, we show that STEM-BT and STEM-BT are are more susceptible to false positives than STEM and STEM-H and, thus, maybe be less desirable in environments with high levels of interference (e.g., a factory).

3.3 Summary

In this chapter, we have proposed and demonstrated two methods of cross-layer design where the link layer takes advantage of carrier sensing at the physical layer to reduce the energy consumption of power save protocols. The first technique, discussed in Section 3.1.2 and Section 3.2.1, is to use carrier sensing to significantly reduce the energy spent listening for wake-up signals. We have demonstrated how this technique can be applied to both a synchronous and an out-of-band power save protocol (IEEE 802.11 PSM [30] and

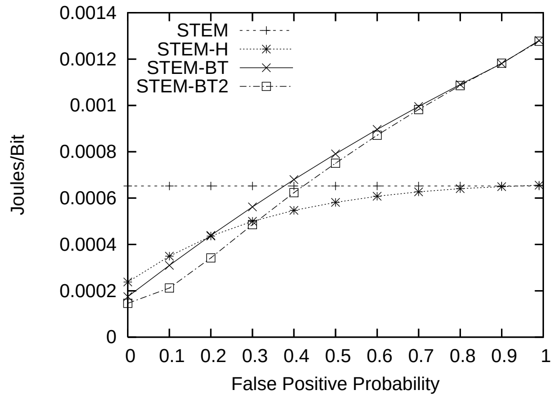

Figure 3.11: Effects of false positives on protocols.

STEM [1, 2], respectively). This technique is particularly beneficial when traffic is light and, most of the time, no packets need to be advertised.

The second technique, described in Section 3.1.3, allows nodes to dynamically extend their advertisement window as long as advertisements continue to be sent and their neighbors remain on. When no packets have been carrier sensed for a sufficiently long time, a node can either return to sleep or wait for the data phase to begin. This technique helps maintain small advertisement windows even when there are few packets to send, while still allowing larger advertisement windows when traffic is heavier.

Chapter 4

Cross-Layer Protocol Design at the Link Layer and Routing Layer

Many previous energy-efficient routing protocols [83–85] focused mainly on using routes with nodes that have more remaining battery life to avoid exhausting nodes as much as possible. However, only recently have routing protocols emerged that are explicitly designed for nodes using power save [27, 51]. Like this latter class of protocols we propose a routing protocol with a cross-layer design that considers the underlying power save protocols when making decisions.

In particular, we design a cross-layer 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 [27, 51] where only two levels of power save are assumed (i.e., (1) not using any power save and (2) using 802.11 PSM). By using the knowledge of multiple levels of power save protocols at the routing layer, we believe this will allow applications (e.g., sensor reports) to achieve an acceptable latency while reducing the network-wise energy consumption. In this chapter, we describe a routing protocol with such a cross-layer design. In Section 4.1, we give an overview of the protocol we plan to simulate. In Section 4.2, we discuss future work we plan to do for this part of our thesis.

4.1 Multi-Level Power Save Routing

In this chapter, we discuss an approach for cross-layer enhancements for power save at the routing and MAC layer. 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 **A** → **B** → **C**. Using network layer information, we can design power save protocols to consider the whole route. In pure MAC-based approaches, the power save protocol would run independently at links $\mathbf{A} \rightarrow \mathbf{B}$ and $\mathbf{B} \rightarrow \mathbf{C}$.

We propose viewing the problem as a multilevel design problem. The idea of using multilevel design to achieve acceptable tradeoffs is prevalent in computer science (see [86] 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.

It is important to note that a number of different power save strategies could be employed at each of the levels. An example of a protocol which uses this technique is found in [27]. In this paper, there are two levels with nodes on the active route remaining in an "always on" state while all other nodes enter 802.11 PSM. The set of nodes on active routes is determined by which nodes have received route-reply (RREP) messages or forwarded data recently. When a node has done neither of these activities recently, it returns to the inactive state. However, the work in [27] does not generalize to the notion of multilevel wake-up, instead concentrating on one specific case of this idea. Our proposition is to investigate the range of protocols which can apply this multilevel technique.

4.1.1 Link Layer Multilevel Power Save Protocols

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. The one we plan to test initially is 802.11 PSM, which is described in detail in Chapter 2.

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 only wake up 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. Thus, to calculate the beacon interval for level PS_i, we have:

$$BI_i = 2^{i-1} \times BI_{base} \quad \text{, when } i > 0 \tag{4.1}$$

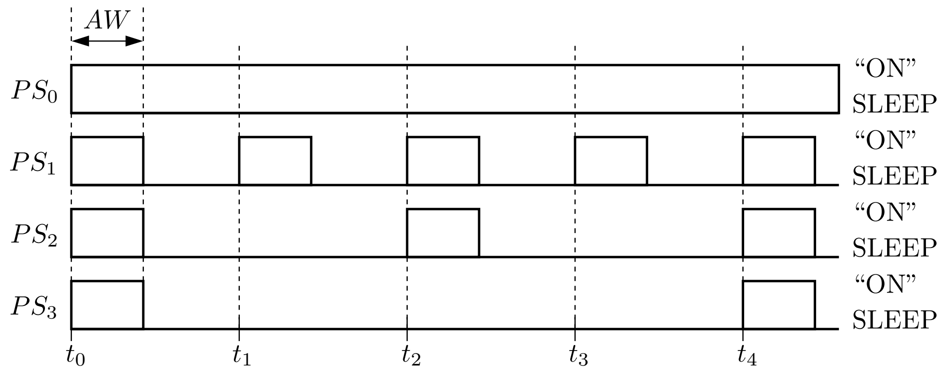

Figure 4.1: Multilevel power save with 802.11 PSM [30].

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., $BI_1 = BI_{base}$).

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 one of these reference point beacon intervals that are spaced BI_{k-1} time units apart. 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. This guarantees the given 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.

Figure 4.1 illustrates the multilevel link layer protocol with 802.11 PSM and $k = 4$. In this figure, AW corresponds to the ATIM window size and we only show 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 reference points in this example are at t_0 and t_4.

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 [1,2] and B-MAC [16]. Nodes using a longer sleeping time between listening periods would save more energy, but require a longer latency to be awakened by neighbors. Additionally, it may be possible to use different protocols on different levels so long as the protocols do not conflict with each other.

4.1.2 Multilevel Power Save Routing Protocol

In this section, we propose an on-demand protocol that can be used in conjunction with the multilevel power save protocols described in Section 4.1.1. We choose to use an on-demand protocol because this allows a source node to compare the path metrics for multiple paths to the destination at the time of route discovery. With a proactive protocol, keeping track of multiple paths to each destination could be prohibitively expensive in terms of storage when traffic is relatively light in the network. In this section, we discuss five aspects of the routing protocol: (1) route discovery, (2) the link cost metric, (3) energy load balancing, (4) maintaining soft timers for flows, and (5) tracking the power save levels of neighbors.

Route Discovery The first aspect of an on-demand routing protocol is the route discovery. Obviously power save affects a broadcast flooding route discovery procedure since the route request will take longer to propagate throughout the network. In Chapter 5, we look more generally at the problem of broadcast dissemination in power save networks and propose a protocol that allows an energy-latency tradeoff. We believe the ideas from Chapter 5 can be used for route discovery.

Link Cost Metric Our goal in choosing a link cost metric is to select paths that have an acceptable power save-induced delay while reducing the number of nodes using high power states as much as possible. From a latency perspective, the application specifies a value, L, which is the maximum end-to-end latency increase *from power save* that it is willing to accept. We note that we only address the latency increase from power save protocols in this work. In particular, increased latency from sources such as queuing delay, contention, and processing/aggregation delay are not addressed in this metric. We believe that the vast amounts of Quality-of-Service (QoS) protocols that have been proposed to address these delays could be used to compliment our protocol.

The L metric could be specified in terms of worst-case latency, average-case latency, or some other statistical definition. We assume the node j is using the k_j-th power level. Thus, PS_{k_j} to denotes its power save level and BI_{k_j} is the length of its beacon interval. Thus, for a path of n nodes, the worst-case latency, our protocol considers the route for use if:

$$BI_{k_1} + BI_{k_2} + \cdots + BI_{k_n} < L \tag{4.2}$$

and if the average-case latency is used, we seek routes such that:

$$\frac{1}{2}\left(BI_{k_1} + BI_{k_2} + \cdots + BI_{k_n}\right) < L \tag{4.3}$$

assuming that the transmission time of a given packet follows a uniform distribution with respect to the sleep schedule of its upstream neighbor.

As mentioned early, we use a source routing on-demand protocol (e.g., DSR [57]). In the route request packet ($RREQ$), each node adds its current power save state to the $RREQ$ before forwarding it. We disable any form of caching since this exacerbates the problem of stale information about the current power save levels on a route. When the destination of the $RREQ$ receives the packet, it sends a route reply ($RREP$) if either:

1. It is the first $RREQ$ it has received for a given source ID and sequence number pair.

2. It is the first $RREQ$ for the source ID and sequence number pair with a power save latency less than L.

3. Let l_{max} be the largest current latency for the route discovery that is less than L and $hc_{l_{max}}$ be the hop count for that path. Let l_{cur} be the latency in the current $RREQ$ and $hc_{l_{cur}}$ be the hop count for the path in the current $RREQ$. Then, the destination will reply with a $RREP$ if (1) $l_{cur} = l_{max}$ and $hc_{l_{cur}} < hc_{l_{max}}$ or (2) $l_{cur} < l_{max}$.

When the source of the $RREQ$ receives these $RREP$'s, it chooses the path with the highest latency that is less than L. If two paths are tied in this metric, then the protocol prefers routes with the lowest hop count.[1] In every data packet a node sends out on a chosen path, it piggybacks two values. The first is the PS_{max}, the highest latency power save level that is acceptable for nodes on the route. The second value is a timeout value that intermediate nodes on the route can use to set their soft timers, as discussed below. If no paths are found with a power save latency less than L for some specified amount of time, then the PS_{max} value is sent along the path with the lowest hop count. If a node that receives a data packet with a PS_{max} value that is greater than its current power save, then it remains in its current power save state state since it is shared with other flows that require a smaller latency. Nodes with a higher power save level transition to PS_{max} to reduce the latency of the path to an acceptable level for the source node. The source node chooses PS_{max} such that $n \times BI_{PS_{max}} < L$ for the worst-case latency or $n \times \frac{1}{2}BI_{PS_{max}} < L$ for the average case latency.

Energy Load Balancing As in previous work [83–85], 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

[1]We note that other metrics such expected number of transmissions or packet loss [87] could be used instead.

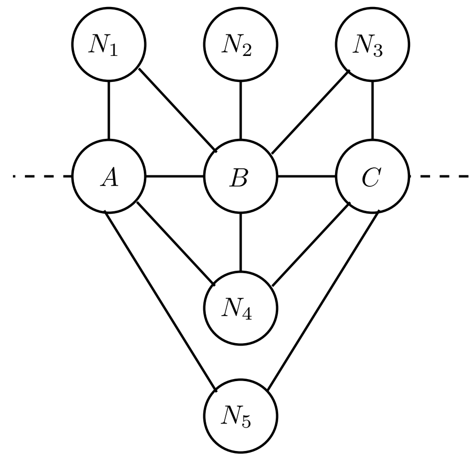

Figure 4.2: Patching a route in multilevel power save.

nodes periodically attempt to "patch" their place on the route with another node with a power level less than or equal to it that can be reached by both its upstream and downstream neighbors on the route. A node could attempt this procedure when its residual energy falls below a specified level or when its recent energy consumption *rate* exceeds a certain level.

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 the 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 4.2. Here, we assume 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. 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.

If a node is part of multiple, disjoint routes, it can still attempt this patch procedure incrementally by applying it to the path which requires the highest power save 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 attempt 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.

Soft Timers for Flows 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 interarrival time for the packets on a flow is highly application dependent, in the spirit of the cross-layer nature of our work, we propose letting the application specify this timeout value and piggybacking it on data packets sent by the flow, as mentioned above. 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 PS_{max} values being piggybacked by each flow.

Tracking Neighbors' Current Power Save States The final aspect of our routing protocol that we address is how nodes maintain which power save level their neighbors are currently using. This is necessary so that a node can wake up its neighbors with the lowest latency possible. A node can piggyback information about its current power save state in each packet it sends and neighbors can cache this information. In the situation where a node attempts to contact a neighbor at the neighbor's assumed power (e.g., the neighbor is assumed to be in the power save state with BI_i, but in reality has transitioned to BI_j, where $BI_j > BI_i$), then the node can fall back to using the BI of the power save state with the lowest possible energy (i.e., BI_{k-1}). If it is still unable to contact the neighbor, the link can be considered to have failed. We note that this is similar to the scheme used in [27] for two-level power save routing.

4.2 Future Work

As future work, we plan to simulate and test the routing and link layer protocol we have described in this chapter. Of course, the protocols may always be subject to modification as issues arise in the simulation. Additionally, we plan to integrate the power save techniques described in Section 3.1 with the multilevel power save protocol from this section to demonstrate a cross-layer approach that spans the physical, routing, and link layers. Other work that we may do, time permitting, includes adapting the broadcast techniques from Chapter 5 to the route discovery process and simulating other multilevel power save protocols besides 802.11 PSM.

Chapter 5

The Effects of Energy-Saving on Multihop Broadcasts

Note: The research in this chapter is joint work done with Cigdem Sengul and Indranil Gupta. This proposal and thesis only focuses on the part of the work on which I made significant contributions in terms of ideas, simulations, and testing.

In Chapter 3 and Chapter 4, we have proposed cross-layer design techniques for energy-saving protocols. In this chapter and Chapter 6, we investigate how energy-saving can affect upper layers. Cross-layer effects are a reality in any network which uses power save protocols. Thus, it is important to characterize the tradeoffs associated with these cross-layer effects.

In this chapter, we focus on energy-saving effects on the propagation of information via multihop broadcast. 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 network administrator. 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, which conserves much less energy than the first option, but has a high latency that may be unacceptable to some applications. In our work, we propose a lightweight mechanism to augment existing power save protocols. The utility of this mechanism is twofold: (1) it allows us to investigate the tradeoffs possible in the energy-latency-reliability domain and (2) it gives network administrators adjustable knobs to obtain a wider range of performance possibilities than are achievable with the two options currently offered by power save protocols. To this end, we propose PBBF, Probability-Based Broadcast Forwarding.

5.1 Probability-Based Broadcast Forwarding

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. There is some mechanism by which all of a node's neighbors will be awake at the same time to receive a broadcast.

While we focus on a synchronous protocol in this paper, 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 in 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.

For concreteness in this work, we use IEEE 802.11 PSM [30] as the base protocol to demonstrate PBBF. While 802.11 PSM may not be directly applicable to sensor networks, it has the most complete specification of any of the power save protocols. 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 [42], are very 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 terms of energy 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 despite the fact that not all neighbors may be awake to receive the broadcast. The second parameter, q, represents the probability that a node remains on after the active time when it normally would sleep.

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 than *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).

```
Sleep-Decision-Handler()
  /* Called at the end of active time */
  /* If stayOn is true, remain on; otherwise sleep*/
stayOn ← false

if DataToSend = true or DataToRecv = true
  then
        stayOn ← true
  else if Uniform-Rand(0, 1) < q
          then stayOn ← true


Receive-Broadcast(pkt)
  /* Called when broadcast packet pkt is received */
if Uniform-Rand(0, 1) < p
  then Send(pkt)
  else Enqueue(nextPktQueue, pkt)
```

Figure 5.1: Pseudo-code for PBBF.

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

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

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 $q < 1$.

In the subsequent sections, we investigate these interactions more thoroughly.

5.2 Brief Presentation of Simulation Results

To explore PBBF, we simulated it in two different ways. First, we used a grid topology with ideal MAC and physical layers. 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 with a code distribution application in *ns-2* with uniformly random topologies. This allows us to explore how PBBF can be used with an application in a more realistic setting. In both sets of simulations, there is one source which periodically transmits broadcasts. In this proposal, we only present a brief overview of the results from the ideal simulator.

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

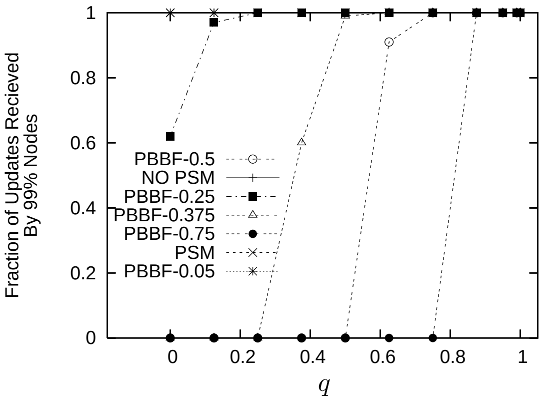

Figure 5.2: Threshold behavior for 99% reliability.

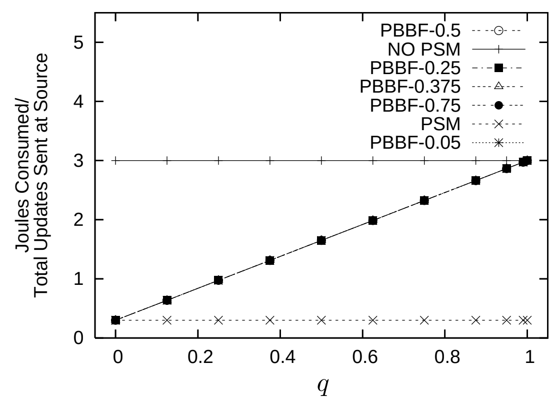

Figure 5.3: Average energy consumption.

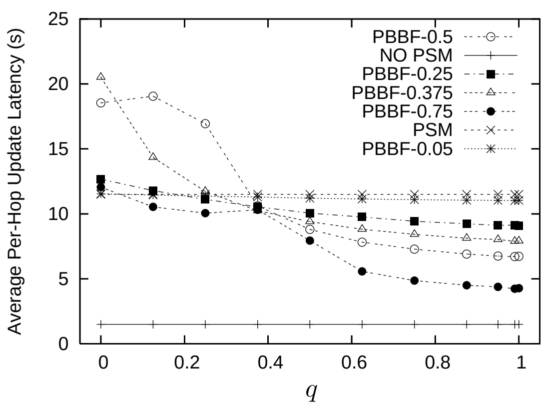

Figure 5.4: Average per-hop update latency.

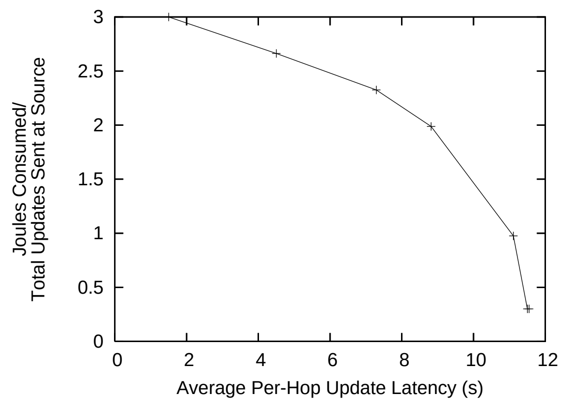

Figure 5.5: Energy-latency tradeoff for 99% reliability.

as is seen with connectivity in percolation theory. In Figure 5.3, we see the relationship between energy 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 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.

5.3 Summary

In this chapter, we have looked at some of the cross-layer effects of using an energy-saving protocol in the context of multihop data dissemination via broadcast. We have designed a lightweight protocol to augment many types of existing power save protocols that allows a tradeoff in terms of energy, latency, and reliability for multihop broadcasts. This allows network administrators more control over the cross-layer effects from energy saving.

Chapter 6

The Effects of Energy-Saving on Symmetric Key Distribution

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 to protocols for other types of networks:

- *Sensors are resource constrained:* The sensors are generally assumed to be small devices that are barely noticeable in most environments. 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 [11] 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 [5, 69, 70, 73, 75].

- *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:* Future 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:* It is envisaged that sensor networks may, for example, be tossed out of an airplane to monitor an area. In such a case, it is impossible to know in advance which specific 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 small 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 may be able to populate the network with a significant number of its own devices.

- *Planned incremental additions maybe 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 paper, we focus on distributing pairwise keys between 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 [88]. Additionally, we are interesting in the cross-layer effects that arise from using the protocol as a result of energy-saving. Specifically, we investigate the tradeoffs that occur in terms of energy and security for the protocol.

The design space for pairwise key distribution lies between two extremes. On one end, each sensor could be loaded with $N-1$ keys prior to 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 only uses 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 only stores 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 novel, distributed protocol that requires only one key for each one-hop neighbor to be stored in memory after a short initialization phase 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 uniformly at random in a desired area. In our model, the network is relatively dense (e.g., more than ten one-hop neighbors per sensor) so there are multiple neighbors which a sensor can overhear.

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. As part of future work, discussed in Section 6.5, we plan to implement the protocol do determine the effects of a more realistic physical layer. We assume that the radio can communicate on multiple, non-interfering channels, but can only listen to or transmit on a single channel at any given time. For example, devices such as Mica Mote sensors [11] and any 802.11 compliant hardware [30] 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 [5, 69, 70, 73, 75], we consider denial-of-service to be beyond the scope of this paper. See [89] for a discussion of possible solutions to address such attacks in sensor networks.

The attacker has two means by which it attempts to learn link keys. The first is by compromising legitimate sensors and learning all the keying material that is stored on the device. In this case, obviously all communication with the compromised sensor becomes insecure. However, learning the keying 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 [68] in which an attacker can only compromise sensors *after* some initialization time.

The second method by which an attacker may attempt to learn link keys is by listening to the plaintext keys that are exchanged during the initialization phase as discussed in Section 6.2. Since all the information needed to reconstruct 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 [5], we assume that the hardware that an attacker deploys is similar to the legitimate sensor hardware in the network. In particular, 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. However, we *do* consider the effects of an attacker adding more colluding hardware which essentially allows it to listen to multiple channels simultaneously.

6.1.3 Bloom Filters [3]

In our protocol, we use Bloom filters [3] 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., [90]).

Each sensor is preloaded with the same h one-way hash functions [91], $H^1_{BF}, H^2_{BF}, \ldots, H^h_{BF}$.[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

[1] A one-way function, H, is one that 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 $H(y) = H(x)$).

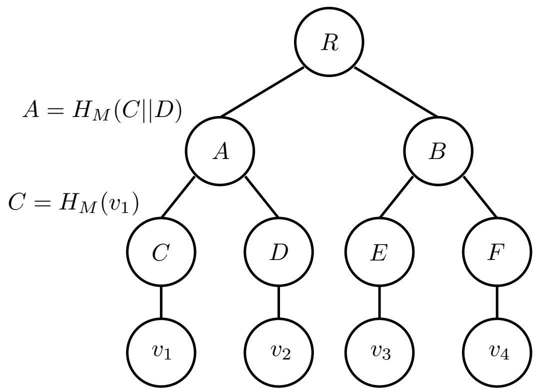

Figure 6.1: An example Merkle tree [4] for four objects: v_1, v_2, v_3, v_4. The leaf nodes are generated by doing a one-way hash, H_M, on the corresponding object. The interior nodes are generated by doing a one-way hash, H_M, on a concatenation of the children of a node.

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. In [92], we investigate what values of h and s are appropriate for our scheme to avoid false positives with high probability. However, our scheme is designed to be robust against occasional false positives as described in Section 6.2.5.

6.1.4 Merkle Trees [4]

Another cryptographic primitive used in our protocol is Merkle trees [4]. We use Merkle trees to provide authentication that the set of keys being broadcast by a sensor was generated by a trusted source prior to 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 prior to 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.

In Figure 6.1, we give an example of how a Merkle tree is generated 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 [91]. 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.

Each sensor that wishes to later verify the authenticity of objects is 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

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)

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.

Prior to deployment, sensors are loaded with a unique, non-overlapping set of α keys by a trusted source. Unlike previous work [69,70], this set of keys is known only to 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 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 uniformly at random 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 attempt 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 its 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 currently 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 of its neighbors. For the discovery phase, each sensor hashes all of 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 performed. 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 only needs to store its link keys, α preloaded keys, and Merkle nodes for authentication. In [92], we discuss how sensors can be incrementally added after the initial deployment of the network. Now, we describe the protocol in detail.

6.2.2 Predeployment Phase

In this section, we describe how keys for sensors in the initial deployment are loaded onto each device. In [92], 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 some larger key pool as has been done in previous work [69, 70] (i.e., a sensor's set of keys do not overlap with another sensor's set of keys). Once the key sets have been 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 generates 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), which we denote as BF. Thus, the Merkle leaf for that sensor is $H_M(ID||H_M(BF))$.[2]

[2] We note that if there is some knowledge of a sensor's post-deployment location *prior* to deployment, then the techniques from [93] can be used to reduce the amount of communication required to verify that a sensor's leaf is legitimate.

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 the sensors are deployed, 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 its neighbor if the key exists in the Bloom filter associated with the neighbor's ID. This scheme is vulnerable to two types of 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 attempt to rebroadcast legitimate packets while remaining undetected.

The second type of attack is for the malicious devices to generate arbitrary keys which hash to all 1's in a legitimate sensor's Bloom filter. In [92], 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 generate 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.

6.2.3 Initialization Phase

The goal of the initialization phase is 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 how much contention there is for the channel. We

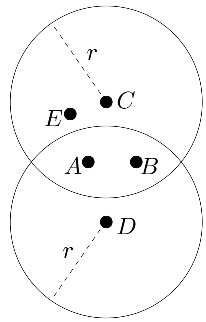

Figure 6.2: One-hop neighbors A and B can both overhear broadcasts from C and D. However, C cannot overhear broadcasts from D and vice versa. E is a one-hop neighbor of A, B, and C.

assume that broadcasts are sent using CSMA/CA to reduce collisions.

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 one-hop 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 will elaborate on how this set of shared keys is 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, there is a $(1/c)^2$ probability that A and B are both listening to the channel on which C broadcasts. Given that A and B are listening to the same channel on which C broadcasts, there is a $1-(1/c)$ probability that E is *not* listening to that channel. When this occurs, A and B will have a shared key that can be used to secure the link against eavesdropping by E.

The spatial diversity comes from the fact that two neighbors, say A and B, may overhear broadcasts from a unique set 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 are all listening to the channel on which C broadcasts a key, E may not correctly receive a packet that A and B do receive correctly. In this case, A and B learn a key that E does not know. Packet loss can occur for a couple of reasons. First, it may be lost due to noise and interference degrading the received signal.

If two sensors do happen to end up in close proximity to each other, our protocol still provides some security from diversity. There are two cases to consider. The first is when two sensors are close enough to have the same set of one-hop neighbors, but still are spaced far enough apart to have uncorrelated packet loss. In this case, the sensors may still receive a different set of keys since they will be switching to different channels *and* experiencing different packet losses. The second case is when the sensors are within the coherence distance of each other. In this case, the diversity in key sets still exists due to the fact that 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, in order to determine a link key, sensors must discover which keys are known by its neighbors. To reduce communication overheard, we use Bloom filters [3] to advertise key sets in a compact manner.

In our protocol, a sensor creates a Bloom filter for advertising its keys by including each of the α keys that it transmitted as well as a subset of the keys it overheard from other sensors' broadcasts. We denote the set of keys that a sensor u overheard from other neighbors as K_u. Node u will then choose a subset of size γ to advertise in its filter (i.e., γ is some predetermined constant that, with high probability, is less than the expected value of $|K_u|$). Since the sensor knows the source of each of its overheard keys, there is an upper limit, λ, placed on how many keys in this advertised subset come from any one neighbor. This is done to avoid 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 a Bloom filter of size s. Thus, to place key k in the filter, a sensor will set the bits $H^1_{BF}(k), H^2_{BF}(k), \ldots, H^h_{BF}(k)$ to 1. A sensor will also store the h values associated with *all* 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 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 an advertisement from sensor v. Node u checks to see which of its known keys are included in v's filter. For each key that 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 [70].

Now, u attempts to determine a unique subset of keys that it shares with each neighbor and creates a link keys 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 attempt to create 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 attempt 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 there are a few options. First, it can choose not to use the link. In a dense network, not being able to use a small number of links may be acceptable. Alternatively, the sensors can request that the sensors do another initialization procedure at a later time to attempt to add the links,[3] 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 [70] if there is enough trust among its neighbors to do so.[4]

6.2.5 Key Establishment Phase

The final phase of the protocol is the key establishment phase. At this point, each sensor should know 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 6.3, 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, will challenge its neighbor, v, with the key k_{uv} that it generated in the previous phase. Specifically, u will send a random nonce, RN, encrypted

[3] In the event that another link key initialization procedure occurs, sensors that established link keys previously do not attempt to re-establish a new link key.

[4] Multipath reinforcement 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 keys if A and B both already share η with each of these m neighbors. See [70] for more details.

by key k_{uv} to v along with a Bloom filter containing the hashes for 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, $\eta \ll \alpha + \gamma$.

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},[5] 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 there was a false positive 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 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 [70]).

6.3 Brief Presentation of Simulation Results

In addition to simulation, we have also done some analysis on our protocol [92]. In this section, for brevity, we only give a summary of our simulation results. We simulated our protocol with *ns-2* [79]. 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 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.

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, 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.

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

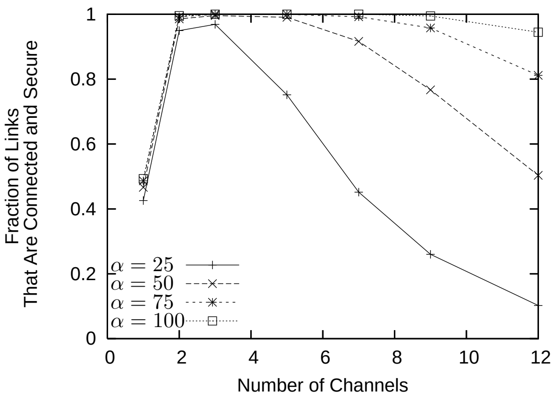

Figure 6.3: Secure connectivity of legitimate sensors vs. the number of channels.

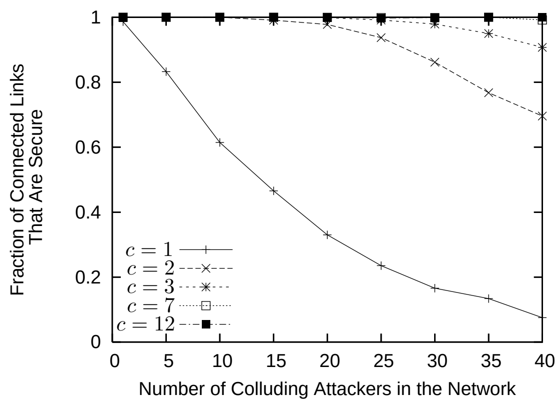

Figure 6.4: Fraction of links between legitimate sensors that are secure vs. the number of colluding attacker sensors.

We set 30% of the sensors, chosen uniformly at random, to be under the control of a malicious entity. In our experiments, 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 between 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 superset of keys known by *all* the colluding malicious sensors.

In Figure 6.3, we show that by adding just *one* extra channel (i.e., $c = 2$), a much larger fraction of the links in the network are secure from the colluding attacker. We note that these benefits obtained by using multiple channel diversity are not possible in the Anderson's work [5], which only uses one channel to broadcast plaintext keys.

At this point, it is evident that setting $c = 2$ provides the significant gain in link security (compared to $c = 1$) while maintaining very high network connectivity (compared to larger values of c). Thus, one may wonder if there is any utility in setting $c > 2$. To answer this question, we refer the reader to Figure 6.4, 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. For brevity, we omit a graph showing the connectivity of the protocol in these settings, but 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 6.4 provide much higher connectivity than is seen in the results for some other key predistribution schemes (e.g., [69, 70]).

Figure 6.4 shows that having more channels allows the connected links to be more secure against the colluding malicious devices. In particular, for $c = 2$, when there are about 15 malicious devices in the

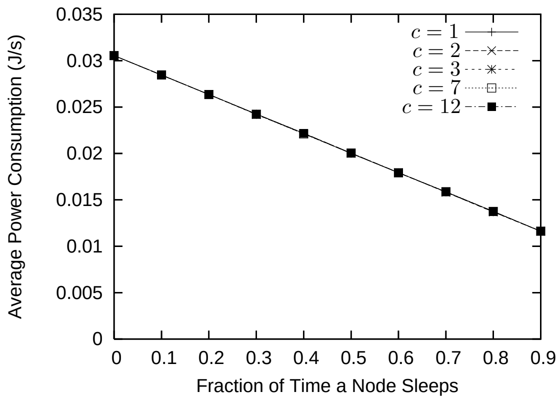

Figure 6.5: Average power consumption vs. the fraction of time legitimate sensors spend sleeping.

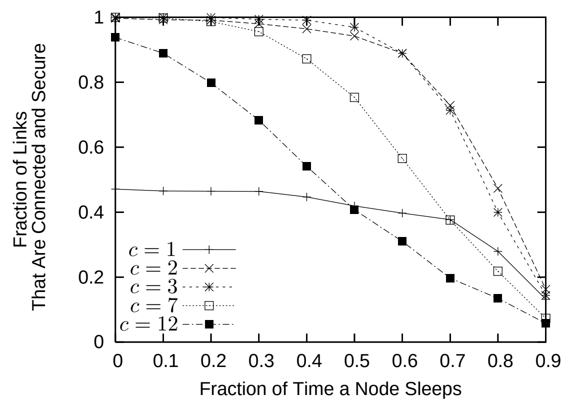

Figure 6.6: Secure connectivity of legitimate sensors vs. the fraction of time legitimate sensors spend sleeping.

network (i.e., 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 remain 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 [5], only up to about ***3%*** of the sensors were malicious.

To test the effect on energy-saving on our protocol, at each fixed interval where a sensor switches channels, it also decides to sleep during the remainder of the interval with probability p. By increasing p, we are able to 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 6.5, 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 6.6, we show how the security of the legitimate links decreases as the time spent sleeping increases. Finally, by combining the data from Figure 6.5 and Figure 6.6, we generate Figure 6.7 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.

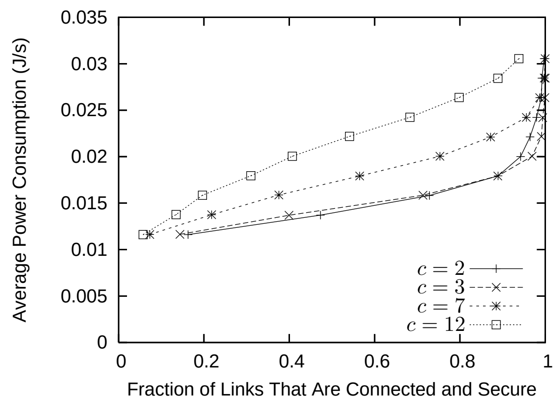

Figure 6.7: Average power consumption vs. the secure connectivity of legitimate sensors.

6.4 Discussion

In this section, we discuss our protocol in relation to the key predistribution method (e.g., [69, 70, 73, 75]) as well as the approach of Anderson et al. [5]. In some scenarios, these methods may be preferable to our protocol. However, we believe there are properties which make our protocol desirable in many environments.

6.4.1 Comparison with Predistribution Schemes

We begin by listing some comparative advantages of our scheme:

- *Network Connectivity:* As shown in Section 6.3, 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 [94].

- *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.

The list of comparative disadvantages of our scheme include:

- *Multihop Key Sharing:* In some applications (e.g., [95, 96]), 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, only establishes keys between neighboring sensors. As mentioned in Section 6.5, adapting our protocol to establish keys with sensors multiple hops away is an area we will pursue 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 require of $O(\lg N)$ overhead associated with Merkle trees (e.g., [93]) 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.

6.4.2 Comparison with Anderson et al. [5]

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 [11] already have this capability. However, by in large, we feel that our protocol offers significant comparative advantages:

- *Greatly Increased Security:* Essentially, any adversary that compromises our protocol would also be able to compromise Anderson's protocol [5]. However, there are many cases where 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 6.3. 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.

6.5 Summary and Future Work

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 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.

Via analysis and simulation, we show that our protocol performs very well in terms of network connectivity and resilience to colluding malicious devices compared to 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.

The first area of future work we plan to pursue is implementing the protocol on sensor hardware. Next, we plan to augment the protocol to allow key establishment among sensors that are multiple hops away from each other, as discussed in Section 6.4.1. Finally, time permitting, we would like to more thoroughly investigate the schemes an attacker can use do compromise our authentication scheme described in Section 6.2.2. In particular, it is interesting to consider the optimal strategy an attacker can use to rebroadcast a legitimate sensor's keys and authentication data without being detected.

Chapter 7

Summary of Proposal

In this proposal, we have considered the problem of saving energy on the radio interface of wireless devices. In Chapter 1, we quantify the importance of this 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).

Having demonstrated the relevance of the problem we are addressing, we next propose *cross-layer designs* to improve energy-saving protocols. By utilizing cross-layer designs, we take advantage of the interactions between layers to improve performance. In Chapter 3, we propose cross-layer techniques that allow the link layer and physical layer to interact. In particular, we propose using the carrier sensing capability provided by the physical layer to reduce the time the link layer spends listening for wake-up signals.

In Chapter 4, we propose using multiple levels of power save protocols, each of which provides a different tradeoff in terms of energy and latency (i.e., a lower latency is possible at the cost of increased energy consumption). Within this environment, we propose a routing protocol that can be used to take advantage of the multiple levels of power save protocols available at the link layer to allow flows to find routes that consume as little energy as possible while providing an acceptable end-to-end latency. Section 4.2 discusses the proposed future work we plan for this part of the thesis.

We then extend the scope of the thesis to investigate the performance tradeoffs that arise for applications as a result of the cross-layer effects of using energy-saving protocols. Such exploration is important because these tradeoffs will arise in networks that use power save protocols. In this thesis, we characterize the tradeoffs that are possible for a couple of applications which are relevant to multihop wireless networks. In Chapter 5, we focus on the application of propagating data via multihop broadcast. This is widely used in many wireless applications, such as discovering routes [57, 58], propagating code updates [54–56], and querying for sensor data [59]. We present a lightweight protocol that can be added to many existing power save protocols to give network administrators a wider range of possible tradeoffs between energy, latency, and reliability. Additionally, we characterize the energy-latency tradeoffs that are possible with the protocol.

In Chapter 6, we consider the effects energy-saving may have on a key distribution protocol. This application is important because it allows the resource-constrained wireless devices to create a secret, symmetric key for each of its neighbors. This key can then be used for encryption and authentication while using orders of magnitude less computation time than public-key techniques. We propose a broadcast-oriented key distribution protocol and then explore the tradeoffs that arise, in terms of security and energy, when energy-saving protocols are used. Section 6.5 discusses the proposed future work we plan for this part of the thesis.

References

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

[2] 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.

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

[4] R. C. Merkle, "A Certified Digital Signature," in *Advances in Cryptology - CRYPTO 1989*, August 1989.

[5] 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.

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

[7] 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.

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

[9] 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.

[10] 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.

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

[12] 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.

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

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

[15] 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.

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

[17] 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 (ALGO-SENSORS) 2004*, July 2004.

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

[19] 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.

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

[21] 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.

[22] 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.

[23] Y. Xu, J. Heidemann, and D. Estrin, "Geography-informed Energy Conservation for Ad Hoc Routing," in *ACM MobiCom 2001*, July 2001.

[24] A. Cerpa and D. Estrin, "ASCENT: Adaptive Self-Configuring sEnsor Networks Topologies," in *IEEE Infocom 2002*, June 2002.

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

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

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

[28] 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.

[29] 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.

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

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

[32] 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.

[33] 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.

[34] 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.

[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] 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.

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

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

[44] 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.

[45] 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.

[46] 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.

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

[48] 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.

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

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

[51] 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.

[52] 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.

[53] N. Reijers and K. Langendoen, "Efficient Code Distribution in Wireless Sensor Networks," in *ACM WSNA 2003*, September 2003.

[54] 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.

[55] 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.

[56] 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.

[57] 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.

[58] C. E. Perkins and E. M. Royer, "Ad-Hoc On Demand Distance Vector Routing," in *IEEE WMCSA 1999*, February 1999.

[59] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks," in *ACM MobiCom 2000*, August 2000.

[60] P. Sinha, R. Sivakumar, and V. Bharghavan, "CEDAR: A Core-Extraction Distributed Ad Hoc Routing Algorithm," in *IEEE Infocom 1999*, March 1999.

[61] P. Sinha, R. Sivakumar, and V. Bharghavan, "Enhancing Ad Hoc Routing with Dynamic Virtual Infrastructures," in *IEEE Infocom 2001*, April 2001.

[62] Y. Wang, W. Wang, and X.-Y. Li, "Distributed Low-Cost Backbone Formation for Wireless Ad Hoc Networks," in *ACM MobiHoc 2005*, May 2005.

[63] Z. J. Haas, J. Y. Halpern, and L. Li, "Gossip-Based Ad Hoc Routing," in *IEEE Infocom 2002*, June 2002.

[64] Y. Sasson, D. Carin, and A. Schiper, "Probabilistic Broadcast for Flooding in Wireless Mobile Ad Hoc Networks," in *IEEE WCNC 2003*, March 2003.

[65] R. Friedman, M. Gradinariu, and G. Simon, "Locating Cache Proxies in MANETs," in *ACM MobiHoc 2004*, May 2004.

[66] 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.

[67] 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.

[68] 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.

[69] L. Eschenauer and V. D. Gligor, "A Key-Management Scheme for Distributed Sensor Networks," in *ACM Computer and Communications Security (CCS) 2002*, November 2002.

[70] H. Chan, A. Perrig, and D. Song, "Random Key Predistribution Schemes for Sensor Networks," in *IEEE Security and Privacy Symposium 2003*, May 2003.

[71] 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.

[72] J. Newsome, E. Shi, D. Song, and A. Perrig, "The Sybil Attack in Sensor Networks: Analysis & Defenses," in *ACM ISPN 2004*, April 2004.

[73] 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.

[74] R. Blom, "Non-Public Key Distribution," in *Advances in Cryptology - CRYPTO 1982*, August 1982.

[75] D. Liu and P. Ning, "Establishing Pairwise Keys in Distributed Sensor Networks," in *ACM Computer and Communications Security (CCS) 2003*, October 2003.

[76] 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.

[77] 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.

[78] E.-S. Jung and N. H. Vaidya, "A Power Saving MAC Protocol for Wireless Networks," tech. rep., University of Illinois at Urbana-Champaign, 2002.

[79] `ns-2` – The Network Simulator. `http://www.isi.edu/nsnam/ns`.

[80] M. J. Miller and N. H. Vaidya, "Improving Power Save Protocols Using Carrier Sensing for Dynamic Advertisement Windows," in *IEEE MASS 2005*, November 2005.

[81] M. J. Miller and N. H. Vaidya, "Using Carrier Sensing to Improve the Energy Consumption of Sensor Network Wake-Up Protocols," tech. rep., University of Illinois at Urbana-Champaign, August 2004. extended version of *Trans. on Sensor Networks* submission.

[82] 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.

[83] S. Doshi and T. X. Brown, "Minimum Energy Routing Schemes for a Wireless Ad Hoc Network," in *IEEE Infocom 2002*, June 2002.

[84] S. Singh, M. Woo, and C. S. Raghavendra, "Power-Aware Routing in Mobile Ad Hoc Networks," in *ACM MobiCom 1998*, October 1998.

[85] J.-H. Chang and L. Tassiulas, "Energy Conserving Routing in Wireless Ad-hoc Networks," in *IEEE Infocom 2000*, March 2000.

[86] N. H. Vaidya, "A Case for Two-Level Distributed Recovery Schemes," in *ACM SIGMETRICS 1995*, May 1995.

[87] R. Draves, J. Padhye, and B. Zill, "Comparison of Routing Metrics for Static Multi-Hop Wireless Networks," in *ACM SIGCOMM 2004*, August–September 2004.

[88] A. Perrig, R. Canetti, J. D. Tygar, and D. Song, "Efficient and Secure Source Authentication for Multicast," in *ISOC NDSS 2001*, February 2001.

[89] A. D. Wood and J. A. Stankovic, "Denial of Service in Sensor Networks," *IEEE Computer*, vol. 35, October 2002.

[90] A. Broder and M. Mitzenmacher, "Network Applications of Bloom Filters: A Survey," *Internet Mathematics*, vol. 1, no. 4, pp. 485–509, 2004.

[91] G. Tsudik, "Message Authentication with One-Way Hash Functions," in *IEEE Infocom 1992*, May 1992.

[92] M. J. Miller and N. H. Vaidya, "Leveraging Channel Diversity for Key Establishment in Wireless Sensor Networks." extended version of *Infocom 2006* paper, 2005.

[93] W. Du, R. Wang, and P. Ning, "An Efficient Scheme for Authenticating Public Keys in Sensor Networks," in *ACM MobiHoc 2005*, May 2005.

[94] 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.

[95] 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.

[96] 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.