2012

Cluster based jamming and countermeasures for wireless sensor network MAC protocols

David Trammell
Louisiana State University and Agricultural and Mechanical College

CLUSTER BASED JAMMING AND COUNTERMEASURES FOR WIRELESS SENSOR NETWORK MAC PROTOCOLS

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Department of Computer Science

by
David Trammell
B.S., Nicholls State University, 2002
May, 2012

Acknowledgments

It is a pleasure to thank my advisor Dr. Kannan and my committee Dr. Busch, Dr. Jianhua Chen and Dr. Zhang. Without their expertise and guidance, this would not have been possible.

This work was motivated by discussions about security in MAC protocols for wireless sensor networks brought to my attention by Dr. Kannan.

I'd also like to thank my original advisor Dr. Kraft and several other members of the Computer Science faculty and staff—Dr. Iyengar, Dr. Douglas, Dr. Tyler—for their encouragement. I must also thank the faculty in general for their excellence which is a great source of inspiration.

I'd like to thank Bruce Lin and Carl Fink for their friendship throughout the graduate program, and last but not least, I'd like to thank my parents for their constant love and support.

Table of Contents

List of Figures

Abstract

A wireless sensor network (WSN) is a collection of wireless nodes, usually with limited computing resources and available energy. The medium access control layer (MAC layer) directly guides the radio hardware and manages access to the radio spectrum in controlled way. A top priority for a WSN MAC protocol is to conserve energy, however tailoring the algorithm for this purpose can create or expose a number of security vulnerabilities. In particular, a regular duty cycle makes a node vulnerable to periodic jamming attacks. This vulnerability limits the use of use of a WSN in applications requiring high levels of security.

We present a new WSN MAC protocol, RSMAC (Random Sleep MAC) that is designed to provide resistance to periodic jamming attacks while maintaining elements that are essential to WSN functionality. CPU, memory and especially radio usage are kept to a minimum to conserve energy while maintaining an acceptable level of network performance so that applications can be run transparently on top of the secure MAC layer. We use a coordinated yet pseudo-random duty cycle that is loosely synchronized across the entire network via a distributed algorithm. This thwarts an attacker's ability to predict when nodes will be awake and likewise thwarts energy efficient intelligent jamming attacks by reducing their effectiveness and energy-efficiency to that of non-intelligent attacks. Implementing the random duty cycle requires additional energy usage, but also offers an opportunity to reduce asymmetric energy use and eliminate energy use lost to explicit neighbor discovery.

We perform testing of RSMAC against non-secure protocols in a novel simulator that we designed to make prototyping new WSN algorithms efficient, informative and consistent. First we perform tests of the existing SMAC protocol to demonstrate the relevance of the novel simulation for estimating energy usage, data transmission rates, MAC timing and other

relevant macro characteristics of wireless sensor networks. Second, we use the simulation to perform detailed testing of RSMAC that demonstrates its performance characteristics with different configurations and its effectiveness in confounding intelligent jammers.

Chapter 1
Introduction

1.1 Overview of Wireless Sensor Networks

A wireless sensor network (WSN) is a collection of wireless nodes that work together to collect a variety of sensory data and report it to a base station by transmitting data, node to node, through the network. The nodes that make up a WSN generally have limited computing resources and limited energy reserves. This presents a particular set of programming challenges that must be met in order to make efficient use of WSN hardware.

WSN node design has been characterized by a few common traits. Nodes have been made at smaller and smaller sizes as developing technology allows. Ideally, nodes will eventually be inexpensive enough to be deployed for one application and forgotten when energy reserves are drained or the hardware otherwise fails. This would allow a wider range of practical applications. Hence WSN nodes must be designed for long life. The life span is currently limited primarily by battery power, which is extended by efficient communication algorithms. CPU efficiency is also important, but the wireless transmitter consumes 2 to 3 times as much energy per second as a typical WSN CPU.

With the primary goals of miniaturization, low cost, disposability and a low duty cycle, a WSN node is expected to have limited computational resources (CPU, memory, power consumption). Indeed, the less demanding WSN algorithms are, the less costly it is to miniaturize WSN node hardware with existing technology. Consequently, developing efficient algorithms for WSN has an immediate effect on the speed at which WSN nodes can be miniaturized. While efficiency is prized, minimum standards of performance, functionality, and security must still be met by a WSN according to the demands of the application.

1.1.1 Applications

It is not hard to envision a wide variety of WSN applications based on the capabilities of existing and theoretical hardware [18, 9]. A WSN may be deployed to monitor and report stimuli in almost any environment imaginable if the hardware and software are advanced enough.

- Generalized applications in environmental observation are numerous. A WSN could be placed among important resources to monitor for fire, flood, weather phenomenon, a variety of pollutants, sound or anything relevant to the resources supported by the WSN.
- WSNs may be used in scientific research to take regular data samples in difficult to reach areas and route the information for convenient access. As an example, this has been done in California redwood trees with the purpose of determining climate conditions at a variety of elevations simultaneously. Using a WSN, scientists were able to take the samples simultaneously and repeatedly with minimal intrusion into the eco system [42]. In a similar example, scientists monitored the habitat of seabirds on a difficult to reach island off the coast of Maine [28].
- Military and security applications include real-time surveillance and other intelligence. A zone covered by a properly equipped WSN could track movement of an entity, determine attributes of the entity (size, speed, direction), monitor for chemical agents, or determine an optimal path through an area according to chosen parameters (maximum aerial cover). Network security is clearly important in military applications.
- If nodes become small enough, future applications may include analogous data collecting scenarios in biological systems (such as human or animal bodies). Important data

could be collected regarding nutrition and disease, new avenues of research may be opened and restorative functions might be performed [29].

The needs of the sensing application determine the design of WSN hardware and software. Real-time monitoring may require high or variable bandwidth and low multi-hop latency while highly active, dense networks may require consistent performance under heavy traffic loads at the expense of optimal latency and bandwidth. We note that the definition of high bandwidth and heavy traffic can be quite low compared to typical wired and wireless networking.

1.1.2 Architecture

A wireless sensor network is made of a collection of wireless nodes. Nodes may be deployed manually in optimal locations, or deployed randomly to cover a given area. A WSN may be stationary, or individual nodes may be mobile [40]. Nodes gather information using a variety of sensors, and the nodes cooperate to route sensory information to aggregation nodes or to edge nodes (called base stations) designed to communicate outside of the network [20].

A base station is a node that provides an access point connecting the WSN to other computer systems. It may be a particular node connected to an outside computer by a wire, or it might be any node within wireless range of a non-network wireless device authorized to communicate with the WSN. A base-station may be permanent or semi-permanent for real-time monitoring applications, or may only be connected periodically when data is collected from the network.

A WSN is designed with the expectation that multiple nodes will cooperate to perform many functions via distributed algorithms, and functions are optimized for a multi-hop topology. Routing protocols must be able to cope with unreliable nodes and use techniques such as distributing traffic over redundant paths for energy fairness. The routing protocol

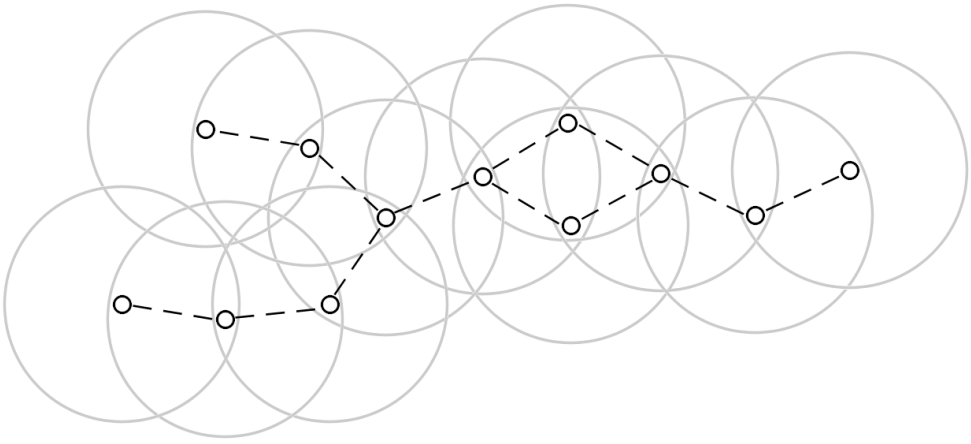

FIGURE 1.1. Wireless Sensor Network: A WSN is a multi-hop network of wireless nodes where nodes collaborate to collect, analyze, and report data. Network connectivity is based on each node's location and wireless range.

LEACH provides an example where cluster management (which uses extra energy) is rotated among nodes to maintain energy fairness [18].

The MAC layer is designed to detect new nodes and form connections as they become available. It is responsible for physical addressing and maintaining and measuring link quality. Nodes may drop out of the network temporarily and be restored (for example, if the battery can be changed or if communication is otherwise impaired), and new nodes may be added at any time. The MAC layer is optimized for multi-hop communication despite being directly responsible for only link level communication. We discuss the MAC layer in detail below [5].

1.1.3 WSN Node Hardware

An individual node (or mote) is a self-contained computer system with a general purpose CPU, memory, flash ROM (for program code and non-volatile storage), battery power, and a variety of I/O interfaces. Typically, the I/O devices include the wireless radio, a series of LEDs, and a communication port that can be connected to a variety of sensor boards or to a

communication board (allowing that node to function as a wired base station). As a specific example, the Mica Z mote has an 8 bit RISC CPU that executes a maximum of 16 MIPS at the highest clock speed, 128 KB program flash memory and a 2.4 GHz radio. The radio is controlled via hardware compliant with IEEE 802.15.4 physical layer specifications.

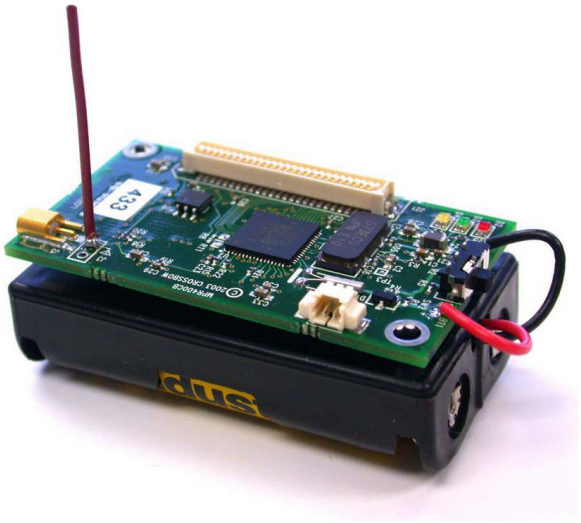

FIGURE 1.2. MICA 2 Mote: A Crossbow MICA 2 WSN Mote. The MICA 2 transmits on 333, 433 and 866 MHz frequency bands with a symbol rate of 38400 (19.2 Kbps in with Manchester encoding). The AA batteries indicate scale.

1.1.4 WSN Node Software

The operating system that runs on a WSN node should be carefully designed to run as efficiently as possible and to solve the particular problems inherent in a WSN. It may also be custom tailored to the needs of a particular application. At each layer, energy consumption is a primary concern [20]. The typical layers of the OSI model may be blurred or greatly reduced in order to conserve memory and energy [5].

Additionally, due to the constrained resources of individual nodes, many algorithms must function in a distributed manner [40, 4]. For example, at the application level, database systems have been designed that can function on a WSN. A database is a natural fit for the

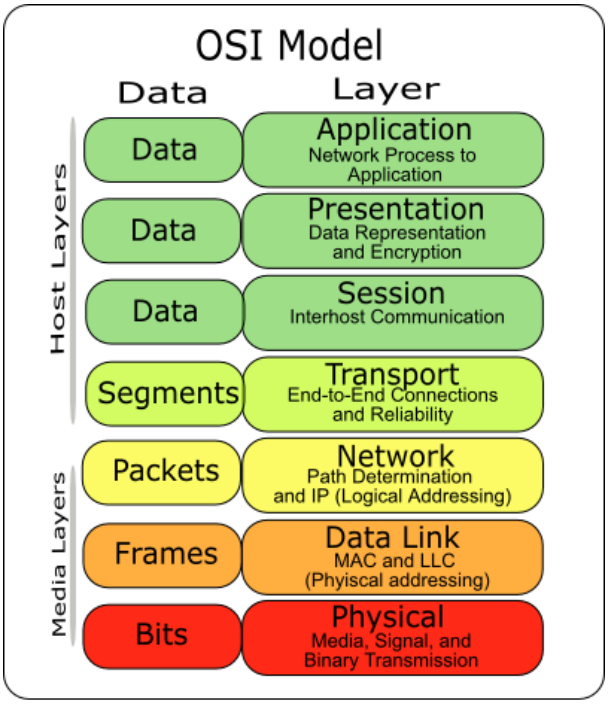

FIGURE 1.3. WSN OSI: The entire network performs as an application and media layers may be blurred together for efficiency.

function of a WSN (to collect data), but a very poor fit from the perspective of an individual node due to lack of computing resources and storage.

1.1.5 Tiny OS and Nested C

Tiny OS is an open source operating system designed to run on a variety of WSN hardware platforms [19]. Tiny OS is programmed using Nested C, which is statically linked (as opposed to dynamic linking typically used in ordinary dialects of C, C++ and other imperative languages) [5].

Static linking makes Tiny OS code more difficult to read and write, but has the benefit of allowing the compiler to perform optimizations across link boundaries. A series of procedure calls spanning several linked modules may be compiled as a single instruction at compile time due to static linking. These optimizations reduce the size of the compiled binary, and reduce memory usage during execution by limiting procedure calls which increase

the size of the call stack. Tiny OS does not currently support multiple threads due to memory constraints.

A Tiny OS application is written in Nested C by creating at least one new file with application code and another wiring the application to lower level Tiny OS structures. The lower level structures exist as a collection of statically linked source code files that can be directly edited by the programmer. At compile time, the OS and application are compiled together as a single executable binary targeting the particular node hardware that the application is intended to be run on. Again, this is in contrast to traditional operating systems which are already compiled so that new code must be compiled separately and dynamically linked to it.

1.2 Overview of MAC Layer Design

The medium access control (MAC) layer in a computer system is a sub-layer of the link layer and is responsible for interfacing with the physical layer, which in turns directly manages access to the network substrate (the radio spectrum) via the radio. The MAC layer manages the point to point communication from one node to another while leaving the problem of the explicit multi-hop path to the higher network layer. Although the MAC layer does not determine a path from one node to another across multiple hops, the MAC layer manages timing among multi-hops in order to prevent two-hop interference and optimize attributes that are important to the particular sensing application.

When a node receives a packet, the link layer (which includes the MAC layer) may take responsibility for ensuring that it is not a duplicate and sending a receipt acknowledgement (ACK) or request for retransmission to the sender. When sending a packet the MAC layer must ensure that the medium is available, send the packet multiple times if necessary, and report the status to the higher layers. It must provide a physical address. Commonly used

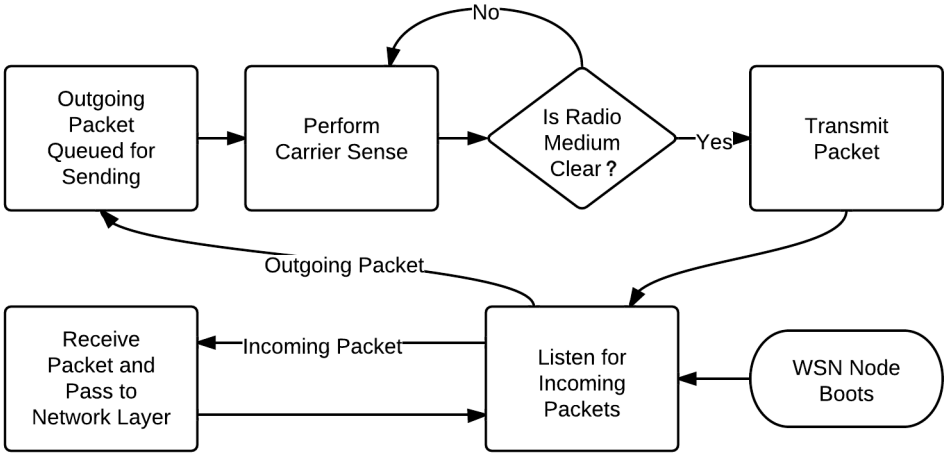

FIGURE 1.4. MAC Layer Communication: This flowchart shows a very simple MAC layer, CSMA, that avoids transmitting when another node is already using the medium. This protocol makes no effort at achieving fairness among equal peers, no effort at energy conservation, and has no special designs to aid with multi-hop communication.

MAC designs can trace their roots to MACA, which uses request to send, clear to send and acknowledgment packets to solve the hidden terminal problem [24]. This was refined [7] and eventually became the basis for the widely used IEEE 802.11 protocol [31].

Several optimizations are necessary for the WSN MAC layer due to the need to consume as little energy and memory as possible and for the specialized needs of multi-hop wireless networks [14, 45].

1.2.1 WSN MAC Energy Optimization

The primary energy consumer in a WSN is usually the wireless radio[20, 5], hence a WSN MAC protocol is designed to function while spending a lot of time with the radio powered down (sleeping). The node wakes up periodically to check the channel for communications from other nodes. The variety of MAC protocols that coordinate a sleep-wake cycle (or duty cycle) in WSNs can be categorized as synchronous, loosely synchronous and asynchronous

[35]. Since data packets are relatively rare events in most WSN applications, energy usage is dominated by listening in most protocols.

The ratio between energy used in transmitting (Tx) and receiving (Rx) changes depending on the encoding method of the radio transmitter. For example, the cc1000 on the MICA 2 mote uses 30 millijoules per second to receive and 81 mj/s to transmit at maximum power (1:2.7), while the cc2420 on the newer MICA Z motes use about 52.2 mj/s to receive and 59.1 mj/s to transmit at full power (nearly 1:1). Depending on the Tx:Rx ratio, the MAC protocol can be biased toward transmitting or receiving to manage coordination with minimal energy use.

Energy use generally takes precedence over throughput and packet latency, and a sleep interval necessarily increases multi-hop latency (message delivery time across multiple hops). However, optimizations can be performed to mitigate the problem if minimal latency is a high priority. This is the case for sensor networks engaged in real-time monitoring.

1.2.2 WSN MAC Multi-Hop Design

The multi-hop nature of a WSN stands in contrast to most wireless devices which communicate directly with a base station that is connected to a wired network. In a single-hop network, communication is simplified and the base station (which being wired is generally not energy constrained and can listen continuously) can take on management functions to facilitate communication. A base station may also serve as a communication hub for a two hop network in which every node is a client of the base station and clients communication through the base station. Such a network may be wireless and optimized for peer to peer transfer as is the case with Bluetooth [16]. In a wireless multi-hop network of equal peers, nodes must get along without specially equipped nodes to facilitate communication.

1.3 Motivation

We focus on two particular areas of interest to WSN research. The first is the nature of the research environment itself. The second is the problem of security against intelligent mote-based jammers. We summarize these problems as follows.

1.3.1 Testing WSN MAC Protocols

Designing and testing a new MAC protocol on WSN hardware can be a very difficult proposition due to several physical factors that negatively effect research efforts. The difficulties present a barrier to entry and slow the speed with which new designs can be created and tested. They also discourage thorough testing of variables and configurations. The main problems are the inconsistency of variables in physical testing spaces, expense of testing hardware, and difficulty of accessing physical spaces to alter network configurations and obtain detailed debugging information.

When testing on physical WSN motes in a physical space, the nature of the space has a significant effect on radio transmissions. Radio chip manufactures report generic transmission ranges for outdoor and indoor use with the indoor range being much lower. Whether inside or out, radio transmissions can be blocked and reflected by physical obstructions and interference may come from unpredictable sources and at unpredictable times and strengths. A MAC layer certainly must be able to cope with unreliable transmissions and medium conditions, however during initial testing, the medium should be carefully controlled to ensure the accuracy and repeatability of results. Doing this with actual motes means controlling the physical space, a very expensive and limiting proposition. Testing is also limited by the available hardware which can be expensive and difficult to obtain.

Testing WSN algorithms with physical motes also presents a challenge for testing various configurations and obtaining debugging information. Flashing a mote's program memory

and physically moving motes to a test environment is a time consuming activity that increase with the size of the network being tested. This discourages thorough testing of program variables and makes debugging very difficult. Debugging information may be recorded on the nodes themselves, but due to memory limitations on the WSN, the information may not be thorough and could affect results. A completed WSN application would ideally record little or no debugging information. Finally, physically obtaining debug information is no easier than flashing the motes and setting up an experiment.

1.3.2 WSN MAC Security

Security in WSN MAC protocols is an ongoing concern. Efficient, strong encryption usually relies on expensive hardware, and encryption done in software is relatively inefficient (it consumes a much higher amount of CPU time and thus energy) [32]. Consequently, encryption is generally avoided but would be necessary for applications requiring security. Failure to encrypt packets leaves a network open to a class of detailed-knowledge MAC layer attacks. Malicious nodes can read timing information directly from packet headers and create modified packets that may place the network into an energy wasting state.

WSN nodes are also vulnerable to jamming attacks [43]. An opponent interested in disabling the WSN for a short period can ceaselessly fill the useful portion of the radio spectrum preventing any communication from occurring. This constant jamming could be accomplished with a few powerful devices, but such devices could easily be triangulated and removed. Constant jamming could also be accomplished by distributing jamming nodes among the legitimate network, but constant use of the radio would result in a very short life span (a few days) for the jamming nodes disabling the legitimate network only temporarily. Nevertheless, even short term loss of function in a critical WSN could be catastrophic for

some applications. Reliably circumventing a determined constant jammer on a single channel remains an open problem.

An opponent interested in long term disabling of a WSN has other options [26]. Intelligent jamming nodes can be distributed throughout a legitimate network that takes advantage of weaknesses in the WSN MAC layer to jam the network while remaining as energy efficient as the legitimate nodes. Jamming nodes can record the local network traffic and identify the length of the WSN duty cycle via cluster analysis of the packet interarrival times. With this knowledge, the jammer can then jam the network by transmitting only during the brief control interval of the duty cycle, thus preventing any useful traffic from occurring without spending much more energy than the network itself.

1.4 Problem Definition

If a WSN cannot communicate securely it cannot be relied upon for critical applications. We consider the problem of designing a MAC protocol for wireless sensor networks that meets the traditional goals of a WSN–low energy consumption and memory usage with bandwidth comparable to other algorithms–while being secure against intelligent jammers. We expect that networks engaged real-time monitoring for security will be the most likely targets for such attacks, hence we focus on the loosely synchronous class of protocol which can be most effectively optimized simultaneously for low-latency and high bandwidth.

We also consider the problem of initial design and testing of new MAC protocols in a consistent and easily accessible simulation environment intended for prototyping and exploring detailed tradeoffs of slight design changes prior to further testing on physical mote hardware. The problems are defined as follows:

- Each node in a network must follow rules for accessing the communication medium in an ordered manner that optimizes attributes desirable to the network. These rules

are referred to as a medium access control (MAC) protocol and they govern direct communication between nodes.

- Initial testing of a new MAC protocol would ideally occur in a carefully controlled environment that can be easily accessed and duplicated.
- Desirable network attributes in a WSN include data throughput, transmission latency, transmission reliability, but energy and resource usage are of prime importance to extend node lifetime and decrease per node cost.
- Since the network functions as a unit, symmetric energy and resource use is important. If an inconveniently located node is overactive and runs out of energy, a large portion of the network could become unreachable and thus fully or partially compromise the WSN application.
- A malicious wireless node may transmit on the medium in disregard of the MAC protocol with the intention of censoring legitimate communication and otherwise lowering the quality of network attributes. A malicious node may also be interested in conserving energy and working with limited resources, hence we should design MAC protocols that resist energy-efficient jamming attacks.

1.5 Research Contribution

We design and test a novel simulation that is tailored to the needs of wireless sensor networks. The simulation provides a realistic and consistent research platform that serves as an effective virtual test space for new WSN MAC layer designs. We also design and test a novel WSN MAC protocol that mitigates the effect of intelligent jamming attacks on legitimate nodes by changing the regular schedule to a pseudo-randomized schedule that makes transmission times unpredictable for a malicious node. We simultaneously use a feature of the modified protocol to achieve global network synchronization eliminating a source of asymmetric energy

use. We also note that a pseudo-random schedule eliminates the need for an explicit periodic neighbor discovery period while improving performance and lowering energy consumption, and we measure the effectiveness of random neighbor discovery under different circumstances.

1.6 Dissertation Organization

The remaining chapters are organized as follows. In chapter 2 we review existing literature related to the problem of medium access control and jamming in wireless networks and sensor networks. In chapter 3 we discuss the details of our novel simulation. In chapter 4 we discuss the details of our novel protocol RSMAC in comparison to SMAC. In chapter 5, we present experimental results comparing our simulation to SMAC to previous research and comparing RSMAC to SMAC in simulation. In chapter 6 we conclude and summarize.

Chapter 2
Background

2.1 Initial Wireless MAC

In this chapter we review literature relevant to the development of WSN MAC protocols. This begins with early wireless protocols not intended for WSNs and concludes with the latest WSN MAC research.

The earliest wireless network protocol used a simple scheme called carrier sense multiple access (CSMA) for medium access control. A radio channel was monitored by several nodes, and any node that wished to communicate would listen for a period to determine if the channel was in use before transmitting. This works with few problems only if each node can hear every other node. CSMA is simply listen for silence before transmitting. A collision between transmissions can occur if two nodes happen to choose the same time for transmission. This occurs very frequently when lots of traffic is present. Protocols that use CSMA are called contention protocols because nodes compete for the medium instead of pre allocating time slots. With no special precautions taken, one node may win contention more frequently than other nodes.

When nodes are spread out so that some nodes cannot hear all transmissions, a situation called the hidden terminal may arise (Figure 2.1, page 16). If Node A senses that the channel is empty and tries to transmit to Node B but Node B is already receiving a transmission from Node C, then Node A will interrupt the transmission from Node C (because Node A cannot directly determine if Node C is transmitting).

The similar exposed terminal problem occurs when Node A overhears Node B transmitting to Node C and chooses not to transmit for fear it will interrupt the transmission

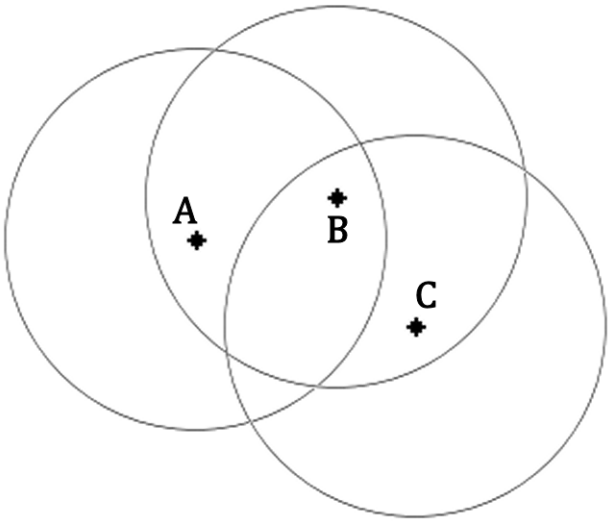

FIGURE 2.1. Hidden Terminal: Node A may interrupt a communication from Node C to Node B because it cannot hear that Node C is transmitting.

(but in fact the target of the transmission is outside the range of Node A and hence no interruption would occur).

2.1.1 MACA

Phil Karn proposed a solution to these problems in 1990 called MACA (Multiple Access with Collision Avoidance) [24]. This solution is still widely used–with enhancements–in many protocols today. A node transmits a request to send (RTS) packet and then waits for a clear to send (CTS) packet before transmitting. Since both the transmitter and receiver send these control (CTRL) packets back and forth, other nodes can avoid the hidden terminal problem, while exposed terminals must remain silent to allow the ACK to be heard by the sender (making the exposed terminal a feature instead of a problem). Note that an RTS collision can still occur, but the cost is lower since the RTS is intentionally short. MACA is commonly referred to as CSMA-CA today.

The RTS/CTS exchange of MACA was inspired by Apple LocalTalk which was a wired protocol in which several devices shared a single wired connection (as opposed to

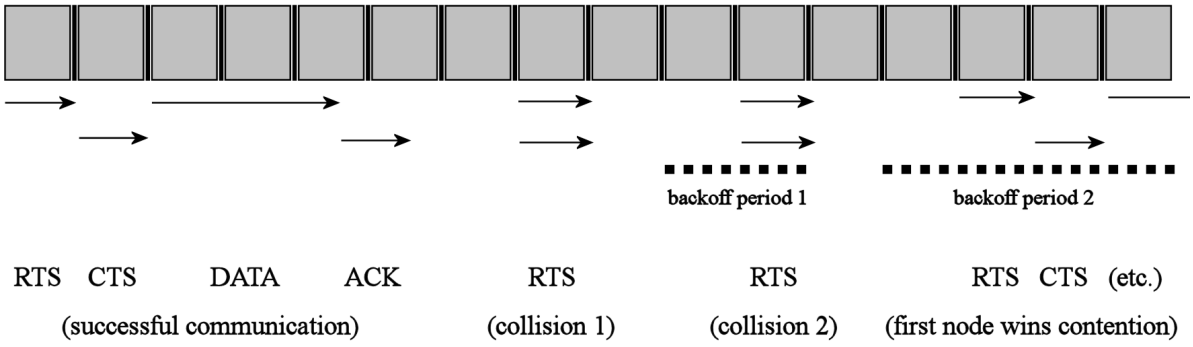

FIGURE 2.2. CSMA-CA with Backoff: The first node wins contention and transmits data. Then a collision occurs followed by random exponential backoff. After an RTS collision with no follow up CTS, both nodes randomly choose from the next two time slots. After a second failure, the nodes choose from four slots. The increasingly large backoff period allows the network to handle heavy traffic from several colliding transmitters (called multiple access). CSMA-CA was inspired by MACA and a variant was used in the widely adopted IEEE 802.11 protocol.

separate point to point connections). In LocalTalk the CTS/RTS dialog was used so that the individual nodes could prepare to receive data (standard CSMA-CA would require that they are constantly ready to receive data). Since all nodes shared the same wired connection, there was no hidden terminal problem to solve. Phil Karn recognized that the RTS/CTS exchange could be used in wireless networks to solve the hidden terminal problem.

MACA is a contention based protocol because any node may still transmit after carrier sense. Collisions may still occur when RTS packets are sent simultaneously, but the cost is low due to the brevity of the packets. If traffic is high, congestion results in a very high rate of RTS collision and low fairness among peers. This problem is addressed with exponential back-off. When an RTS is sent, if no CTS is heard, the RTS is repeated after some random amount of time within a window that increases exponentially after each failure. This prevents two colliding nodes from coordinating their back-off periods. The backoff window increases exponentially with each failure because repeated failures suggest that too many nodes are

trying to win contention (hence a larger window will eventually allow one of them to win the medium). The window is instantly reduced to the smallest size again after a single successful RTS/CTS exchange, although later research determined that a slower reduction is more effective [7].

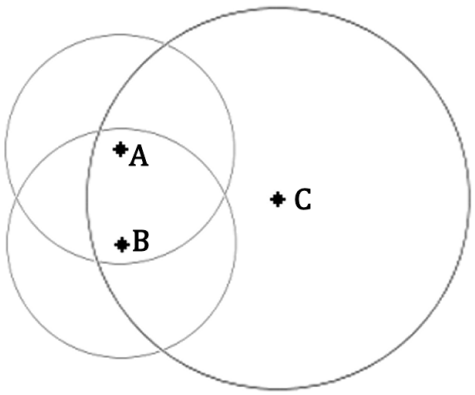

FIGURE 2.3. Effect of Differing Transmission Ranges: Node C will not hear the RTS/CTS exchange of A and B and cannot avoid collisions. CSMA and CSMA-CA do not work effectively without equal transmission range.

Phil Karn also proposed a power control scheme that would attempt to maximize geographical channel usage (akin to several people conversing quietly in a room instead of one pair shouting at each other and inhibiting all other conversation). Essentially, a node would increase the strength of its CTS packets if it determined that its CTS or data packets were not reaching the target (which is suggested if the CTS or data packets are interrupted). In fact, this was not well thought out since the RTS/CTS exchange for solving the hidden terminal problem relies on the assumption that all nodes are transmitting at equal strength [7]. In figure 2.3 (page 18) we can see that nodes A and B would fail to inform node C of their transmissions. Such power control could be used to save energy for DATA packets if the RTS/CTS exchange occurs with a fixed transmission area in mind, but it is not effective for reducing transmission area per node.

2.1.2 IEEE 802.11

Bharghavan et al expanded upon MACA and early work on IEEE 802.11 creating a protocol called MACAW. They added a short DS packet before each DATA packet to indicate a successful RTS/CTS exchange and ongoing data transmission, and ACK packets at the receiver also inform of an ongoing data transmission while confirming the delivery of each DATA packet for the sender [7]. IEEE 802.11 ended up supporting a similar exchange and has become widely used in consumer and business applications [8, 31].

2.2 Energy Efficient MAC Designs

IEEE 802.11 was designed for fixed wireless access points to communicate with a collection of mobile devices and was not designed to be energy efficient. Singh and Raghavendra proposed PAMAS for ad hoc radio networks to conserve power. It is based on MACA but uses a separate channel for control packets and nodes power down while waiting for other nodes to send data [39]. However, it only avoids overhearing (hearing a packet intended for another destination) and collisions (simultaneous transmissions corrupting each other), while idle listening (listening while no transmission are present) is the primary source of energy waste.

Bluetooth was designed for peer to peer communication among mobile devices in a small area and was designed with energy efficiency in mind among other goals (including world-wide compliance with the varied laws and regulations governing the radio spectrum around the world). [16]. However, Bluetooth was not originally designed for the ultra-low power consumption required by WSNs and is not scaleable (being designed for groups of eight nodes rather than multi-hop networks).

2.3 WSN MAC Designs

WSN specific MAC design focuses primarily on conserving energy ahead of other considerations such as bandwidth (rate of data transmission), latency (length of wait between desire

to transmit and successfully completed transmission) and fairness (ensuring that each node has equal access to the channel). WSN MAC designs can be categorized as follows based on the manner in which nodes coordinate sleeping to save energy ([35].

2.3.1 Asynchronous

The BMAC protocol, developed at Berkeley [33], uses a common and brief but uncoordinated check-interval to conserve energy. A node that wishes to transmit precedes every transmission with an ultra long preamble that is long enough to allow every node to wake before the preamble ends and data transmission begins. A preamble is normally a short signal used to allow the receiving node to tune the radio before receiving actual data. A standard preamble might consist of a few bytes of alternating 1's and 0's followed by two 1's to signal the start of the actual message content. In BMAC, the preamble is extended to cover a very long period that matches the pre-chosen network-wide check interval. This allows nodes to spend most their time with the radio disabled to conserve power (recall that merely listening for an incoming transmission can take just as much energy as transmitting).

For example, with a check interval of 80 milliseconds using a cc1000 radio (a common radio on WSN nodes such as the MICA2), BMAC would send a 192 byte preamble prior to any transmission. Any node that is not transmitting can wake from sleep briefly every 80 ms and sample the channel for the preamble. Since WSN energy usage is dominated by listening on the radio, the low rate of listening extends the network lifetime greatly.

The primary benefit of BMAC (over other energy efficient WSN protocols) is simplicity in programming (and thus limited memory usage), but there are several drawbacks. BMAC uses CSMA rather than CSMA-CA. The RTS/CTS handshake is avoided due to the length of the preamble which would be required before both the RTS and the CTS. This means that BMAC suffers from the hidden terminal problem and the cost of a collision is

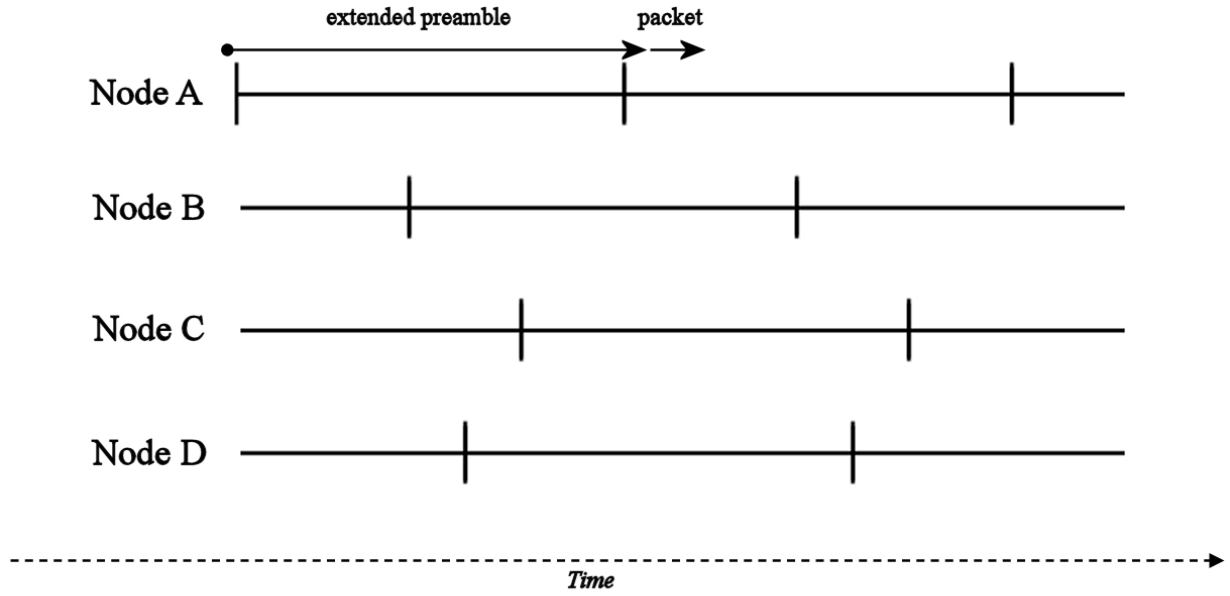

FIGURE 2.4. BMAC Protocol: The vertical lines represent the periodic wake and listen for each node. Each node sleeps when not transmitting. The sleep period is a predefined length with a brief wake period at the ends. If a node wants to transmit, it wakes up and uses an extended preamble that is slightly longer than the sleep period ensuring that every node wakes up in time to hear the subsequent packet. If two nodes want to transmit, the second node waits for the first to finish. The basic protocol is very simple to implement and takes little program memory. However, it does not support collision avoidance because the extended preamble makes the latency and energy cost of an RTS/CTS handshake too great. Instead, mitigating collision is left to higher layers.

very high due to the length of the preamble. This makes BMAC a poor choice for dense networks, or networks which may have bursts of high activity at times.

2.3.2 Synchronous

Opposite to asynchronous protocols, we have synchronous protocols which are characterized by tightly coordinated schedules [18, 34, 35, 36]. Rather than using carrier sense for multiple access (CSMA), time division is used so that each node has an independent and dedicated channel within a large repeating frame. This is referred to as TDMA (time division multiple access). Time is divided into a series of frames consisting of several time slots that serve as listen intervals dedicated for usage by specific nodes. Each node in a local group is assigned a dedicated time slot for communication so the RTS/CTS exchange is not needed to avoid collisions. A node can simply wake up at the allotted time to transmit and listen for transmissions. Neighboring nodes don't communicate directly with each other, but with a cluster head which then passes the communication on to other the destination. The cluster head listens constantly requiring rotation if this paradigm is used in energy constrained peer-to-peer networks.

The chief difficulty in using TDMA in large multi-hop network is inter-cluster communication. There is also the problem of maintaining tight synchronization to maximize efficiency. Global synchronization can be achieved with special hardware, but that is often not available to WSN nodes. For example, a GPS receiver on each node could be used to synchronize with GPS time, or a high power radio transmitter may transmit on a secondary radio channel dedicated to maintaining time synchronization. Without global synchronization hardware, nodes must use a distributed time synchronization algorithm where the clock of one node is used as a reference for the rest of the network.

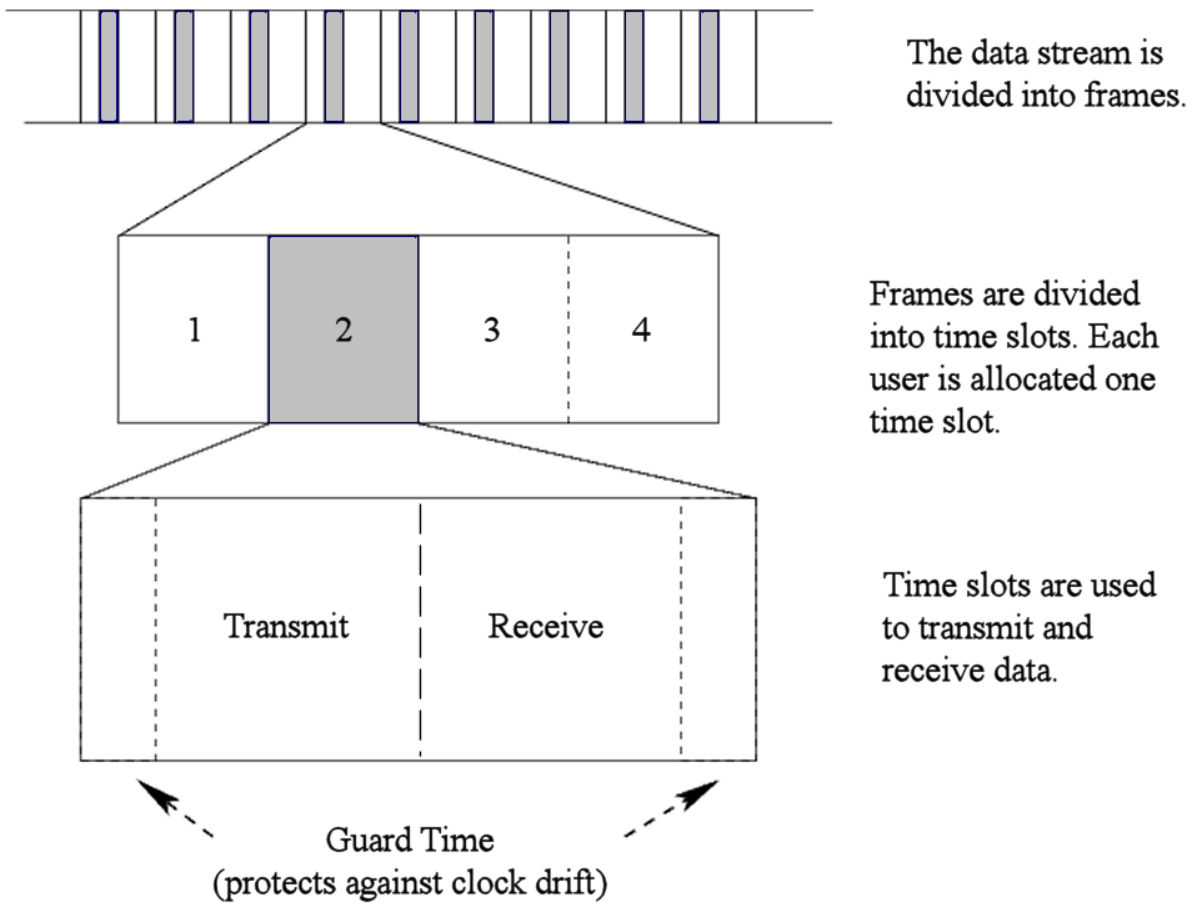

FIGURE 2.5. TDMA Protocol: In traditional TDMA, each node has a dedicated slot for communicating with the cluster head. TDMA was originally designed for one-hop communication through a cluster head rather than multi-hop networks. Nodes do not communicate directly, but send messages through the base station. This completely eliminates contention. Adapting TDMA for multi-hop networks creates a problem of scale in which communication between clusters must be managed despite different clusters following different schedules and cluster heads being too far from one another to communicate directly. Alternatively, distributed TDMA does without cluster heads but comes with performance costs. Packets can only travel one hop per frame, bandwidth is fixed by the slot size, and transmit requests aimed at a single node suffer from contention. Nodes sleep during time slots assigned to other nodes providing energy efficiency.

TDMA slot assignment my be handled in a distributed fashion instead of via dedicated or rotating cluster heads. In this case, nodes must communicate directly with each other and time slots are not fully independent. Nodes can use a small portion of the slot to accept small, fixed-sized, listen requests and the larger portion for transmitting. A node responds to a listen request may waking during the request-sender's transmission period. Using this method, packets may travel only one hop per TDMA frame, which tends to be long compared with the check interval in asynchronous protocols and the frame length of loosely synchronous protocols.

2.3.3 Loosely Synchronous

A loosely synchronous protocol is synchronized but rather than partition the frame into slots dedicated to individual nodes, the listen interval is shared by all nodes. A portion of the interval is used for synchronization, a portion is used for CSMA-CA contention, and the rest is used for sleeping or data transmission (when data exceeds the contention window). SMAC is the prototypical example [45]. Loose synchronization is robust against clock drift because the contention window is much larger than clock drift rates. Periodic sync packets are sent to maintain the loose schedule synchronization within the bounds of clock drift rates.

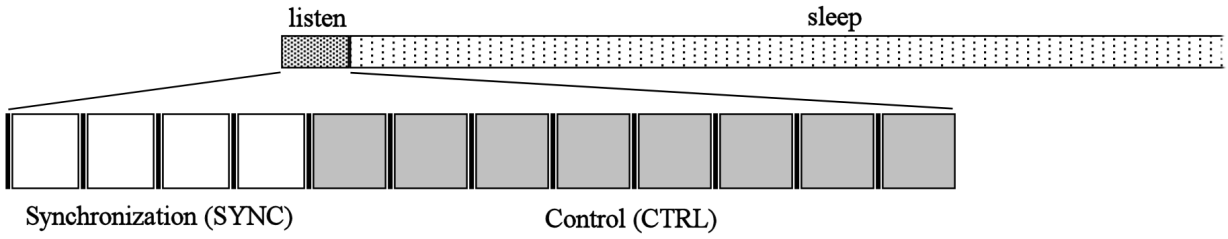

FIGURE 2.6. SMAC Frame: Nodes use CSMA-CA during the listen interval.

SMAC forms local synchronization groups so that nodes on the border of two or more groups have to maintain two schedules. This is a considerable problem because border nodes will use more energy in proportion with the number of secondary schedules maintained. If a node is a border node in the MAC layer grouping and also a routing bottleneck, it may run out of energy much sooner than the rest of the network. If a depleted node was part of an exclusive path, a portion of the network would be cut off until the node or batteries are manually replaced (a task that may be slow or impractical). Since multiple node share a contention window, SMAC can transmit data several hops per from. Adaptive listening extends the range further and the frame size is typically less than in TDMA. In combination, this allows SMAC to achieve much lower multi-hop latency. Low multi-hop latency is a vital trait for real-time monitoring applications, and such applications are the most likely targets of jamming. This is our primary reason for focusing on the security of loosely synchronous class of protocol in this research.

2.4 Wireless Jamming

Jamming in radio communication occurs when a malicious user floods the channel with a transmission intended to scramble the information content of the legitimate transmission or otherwise nullify the transmission capability of legitimate devices. Someone that wishes to censor a WSN may find that a collection of jamming motes is the best option for accomplishing this. Consider the potential ways of censoring a WSN. We could use one or a few very powerful transmitters to jam the network area, but such a limited number of obvious jammers could be easily found and removed. Legitimate WSN nodes may not be easily destroyed for the same reason that we may not easily be able to change the batteries.

The problem of jamming wireless networks was considered by Xu et al. [43]. They developed and tested four types of jammer against wireless networks.

2.4.1 Basic Jammers

The Constant Jammer simply sends an unceasing stream of random bits. If the victim node uses carrier sense for contention, it will never send a packet because the channel will never register as idle. If the victim transmits without using contention, the transmission will be corrupted by the jammer. In ideal circumstances, the Constant Jammer attains a 100 percent censorship rate.

The Deceptive Jammer somehow obtains a legitimate packet and then sends that packet repeatedly. This is similar to the constant jammer in some ways. If a collision occurs, the legitimate transmission will be corrupted. However, if the legitimate node listens to the jammer's stolen packet, it may cause the node to change states because the packet may appear legitimate. For example, if a node hears an apparently legitimate RTS packet, it will attempt to send a CTS and then wait for DATA. In this way, a Deceptive Jammer can cause a victim node to use energy more quickly than a Constant Jammer, while still attaining very high censorship.

The Random Jammer sends randomly generated bits or stolen packets like the previous two jammers. However the signal or packets are sent less than 100 percent of the time to save energy and extend the life of the jamming mote. With less than perfect yet still very high censorship rate, the WSN may still be fully censored for practical purposes. The censorship rate needed to cripple a network depends on the application.

The Reactive Jammer listens to the channel and begins transmitting as soon as a packet is heard with the goal of corrupting that packet. This naturally produces a 100 percent censorship rate in ideal circumstances. However, since the listen to transmit energy ratio ranges from 3:1 to slightly worse than 1:1, the Reactive Jammer may not be any more energy efficient than a Constant Jammer.

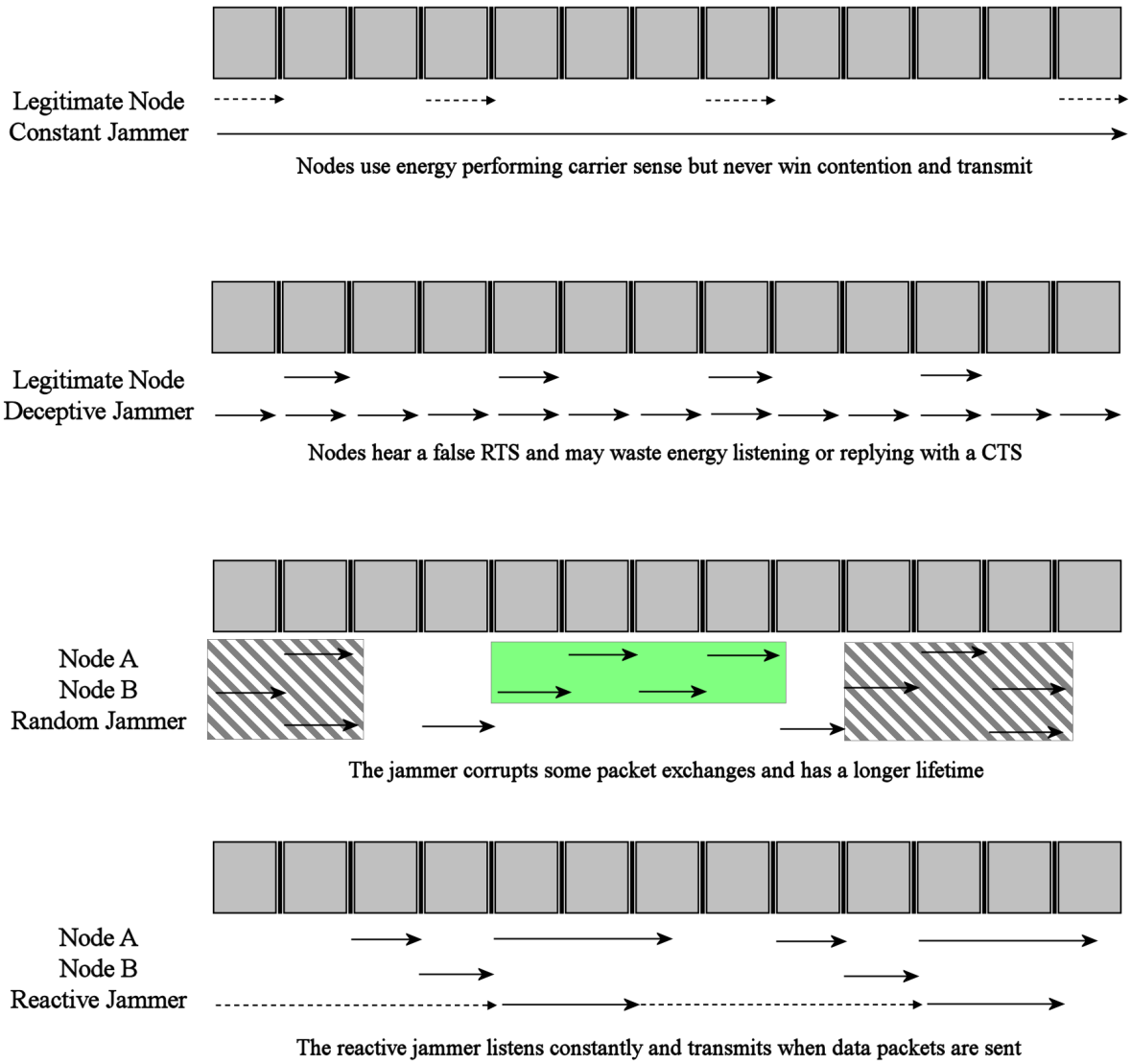

FIGURE 2.7. Wireless Jammers: Xu et al. proposed four jammers that can disrupt general wireless networks. The random jammer conserves energy at the expense of censorship rate, while the other three constantly use energy and would expire within a few days if implemented on a typical WSN mote. Long term censorship of a WSN requires a more careful approach.

The important thing to note about these jamming strategies is that their energy usage is very high compared to WSN energy usage. For example, a Constant Jammer implemented on a MICA2 mote would use two AA batteries in less than four and a half days (81 millijoules of energy per second from 30.8 kilojoules of energy reserves). With a legitimate network life time of 1 to 2 years, that amounts to a censorship of no more than 1 percent of the lifetime of the network.

2.4.2 Intelligent Jammers

Law et al. considered the problem of intelligent wireless sensor jammers with extended lifetimes to match the WSN nodes. First they developed attacks that rely on detailed knowledge gained from listening to control packets of legitimate nodes. If the jammer knows the packet format, it can read the sleep/wake cycle of the victim and synchronize based on that information. From there it's a simple matter for the jammer to become just as energy efficient as the legitimate nodes while maintaining a nearly perfect censorship rate. This suggests a need for packet encryption to maintain security in important networks [25].

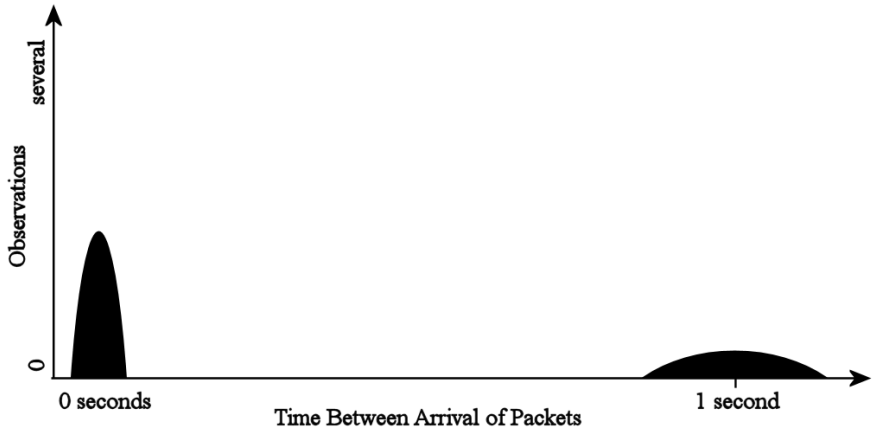

FIGURE 2.8. Cluster Based Jamming: Observations of arrival time between SMAC packets show a clear pattern that can reveal the SMAC frame length to an observant jammer, even if packets are encrypted.

However, even without detailed knowledge, Law et al. found that the temporal characteristics of the WSN reveal important information about the sleep/wake schedule of the nodes [26]. By measuring the time between the sending of two adjacent packets for a long period of time and using cluster analysis on this data, the jammer can estimate the sleep/wake cycle in loosely synchronized protocols, or determine the slot size in TDMA protocols and jam efficiently with minor adjustments. For BMAC derived low-power listening (LPL) protocols, the jammer only needs to learn the minimum sleep interval by measuring the length of the extended preamble. With that information the jammer can be as efficient as a legitimate node. Cluster analysis is not necessary.

2.5 Coping with Jammers

Law et al. suggest basic counter measures to their own jamming technique. By shortening the sleep time in SMAC, the packet interarrival times become close enough that cluster analysis may not work well. However, shortening interarrival times enough to accomplish this is self defeating because node lifetime is greatly reduced.

Routing algorithms have been proposed that identify jammed nodes (or nodes that otherwise have a low rate of success in transmitting) and route traffic around those nodes [38]. However, this will not be effective if a jammer is censoring an exclusive path (or if jamming nodes are widely present among the legitimate nodes).

Jamming may be overcome by increasing transmission strength on a jammed channel [21]. This is a better fit for TDMA protocols because asymmetric links reduces the performance of CSMA-CA based protocols (figure 2.3, page 18), however changing signal strength may affect the slot assignment in TDMA. Power control is the ideal method to mitigate constant jamming on a single channel at the MAC layer, but each type of radio hardware has a maximum transmission strength that will not always be enough to overcome jamming.

Rowe et al. designed and tested a TDMA based WSN with global synchronization hardware that overcame cluster based jamming analysis (reducing its effectiveness and energy efficiency to that of a random jammer) by varying the slot size according to a pseudo-random pattern [32]. This protocol relies on a secondary radio receiver and an AM pulse transmitted on a wired grid to maintain tight synchronization characteristic of TDMA protocols. Additionally, because it is TDMA, it does not meet the goals of low latency and high bandwidth that we seek for a real-time monitoring application.

Chapter 3
Original Simulation Environment

3.1 Testing a MAC Protocol in Simulation

To test new MAC protocols, we have developed a simulation environment tailored to the unique properties of Wireless Sensor Networks that takes into account relevant details of the underlying hardware and also presents a programming environment reminiscent of programming an actual WSN mote. This allows for a more realistic and revealing design process and easier porting of simulation code to an actual mote compared with popular event-based network simulations such as NS-2.

We contend that simulating a design prior to hardware testing has a number of advantages that may result in faster overall development times and more robust design. We summarize the reasons for using a simulation for design and testing below.

- The difficulty of handling physical motes discourages thorough testing of a protocol. WSN testbed motes must be physically deployed to and collected from a testing environment and have their individual program memories reflashed for any program changes. Batteries must be checked or replaced periodically and motes must be monitored for hardware failure and replaced (a relatively frequent occurrence).
- A physical testing environment is generally subject to a great deal of inconsistency and limitations. In the case of outdoor testing, controlling the environment is nearly impossible. For small indoor environments it may be possible at great expense. In the absence of control, experiments will be subject to unknown sources of interference and network topologies cannot be precisely controlled and may be limited by available space. As an example, [45] performed latency testing with 11 nodes placed in a straight

line 1 meter apart and set to transmit at the lowest power. This was an attempt at creating a controlled test of a linear 10 hop network which actually failed unbeknownst to the authors (connectivity was not linear). We cover this example in some detail in chapter 5.

- There are limitations of debugging physical motes for many of the same reasons it is difficult to deploy physical motes. In addition to time and resources consumed collecting debug data from individual motes or setting up external spectrum monitors, attempting to store any data on the motes undergoing testing can affect the result of the experiment due to the limited resources of the motes.

A simulation gives us full control over the virtual radio medium allowing us to consistently control and repeat experiments to an extent that is not possible with a physical medium. We can simulate the radio medium with and without interference, quickly control physical node parameters and modify program code without needing a physical interface to every node in the network. A simulation also provides observational benefits and error reporting that are not possible in a physical network.

There is a lot of precedent for testing new MAC protocols in existing or custom simulators [40, 34, 6, 17, 41, 12, 36] and even jamming [43, 26]. We feel confident as well as justified working in a simulation because medium access control protocols are designed to operate under conditions of varying physical link quality and reliability. The MAC layer experiences link quality and availability variance simply as corrupted packets and missing responses in challenge-response protocols. We feel that more harm than good can potentially be done to initial test results by uncontrolled physical variables. A protocol should eventually be tested in a physical environment, but only after obtaining as much information about the variables as can be obtained in simulation.

3.2 Summary of Simulation Design

Our simulation environment was designed to simulate the timing and energy consumption of radio transmission and interaction among WSN nodes communicating within a virtual radio medium. Our intention is to simulate nodes with a resolution and accuracy sufficient to test the timing characteristics and energy consumption of MAC layer protocols in wireless sensor networks while providing a programming environment reminiscent of programming an actual WSN node for easy portability.

The simulation is generalized enough to simulate stationary or mobile nodes, wired or wireless communications, single or multi-hop networks, and background radio interference. Currently the simulation is run in 2D space, but could be expanded to simulate communication in 3D space. Similarly, the medium object is currently focused on simulating a radio medium, but it could be expanded in a straightforward manner to include physical information for simulating sensing events, physical obstructions to radio communication and node mobility, and anything else of interest.

3.2.1 Simulation Core

The simulation core consists of an object that represents the communication medium and manages interaction between nodes in the medium. After the medium is instantiated, individual nodes are instantiated within the medium and the simulation is set to run for some specific amount of time (in minutes with no particular limit).

In the simplest case, the main simulation loop runs at a resolution of one tick per radio bit of the simulated radio hardware. We can enable multiple transmitter types with different bit rates by setting the tick rate equal to the least common multiple (LCM) of the bit rate of all radios present, and running each node once every N ticks where N is the simulation tick rate divided by the bit rate of a node's radio. One call to run represents

the time taken to transmit one bit on a node's radio. Most calls to run result in very little activity because a WSN node spends most of its time waiting for timers to expire.

3.2.2 Simulated Nodes

When a node's run method is called, it may perform general CPU operations, or interact with the medium by either transmitting or listening on the radio. In the simulation, transmitting amounts to a node simply updating its own transmitter state. This does not require a call to the medium object. It's analogous to using an actual transmitter which is controlled by calling simple functions and waiting for the transmission to complete. Listening is implemented as a query to the medium object and the medium object then determines what the listening node hears based on the transmitter state of each other node on the medium and the radio interaction model. Listening is used to perform carrier sense (determining if a signal is present on the medium) and if carrier sense detects a signal, the node may attempt to decode the signal into a packet.

3.3 Simulated Nodes in Detail

Tiny OS is arguably the most common platform for programming actual WSN nodes. Tiny OS uses nested C (a language developed for embedded systems) which requires that modules be statically linked (or "wired") to one another. This is quite different from standard imperative programming with dynamic linking. While it complicates the code, it allows the compiler to optimize program code across link boundaries converting a series of function calls into a single line of code and other consolidation techniques that save CPU time and memory space.

In our simulation, the physical layer is represented by the basic Node class. This handles basic radio communication and hardware for the node including the clock and simulated clock drift. This class is extended to simulate a variety of different MAC layers. Layers above

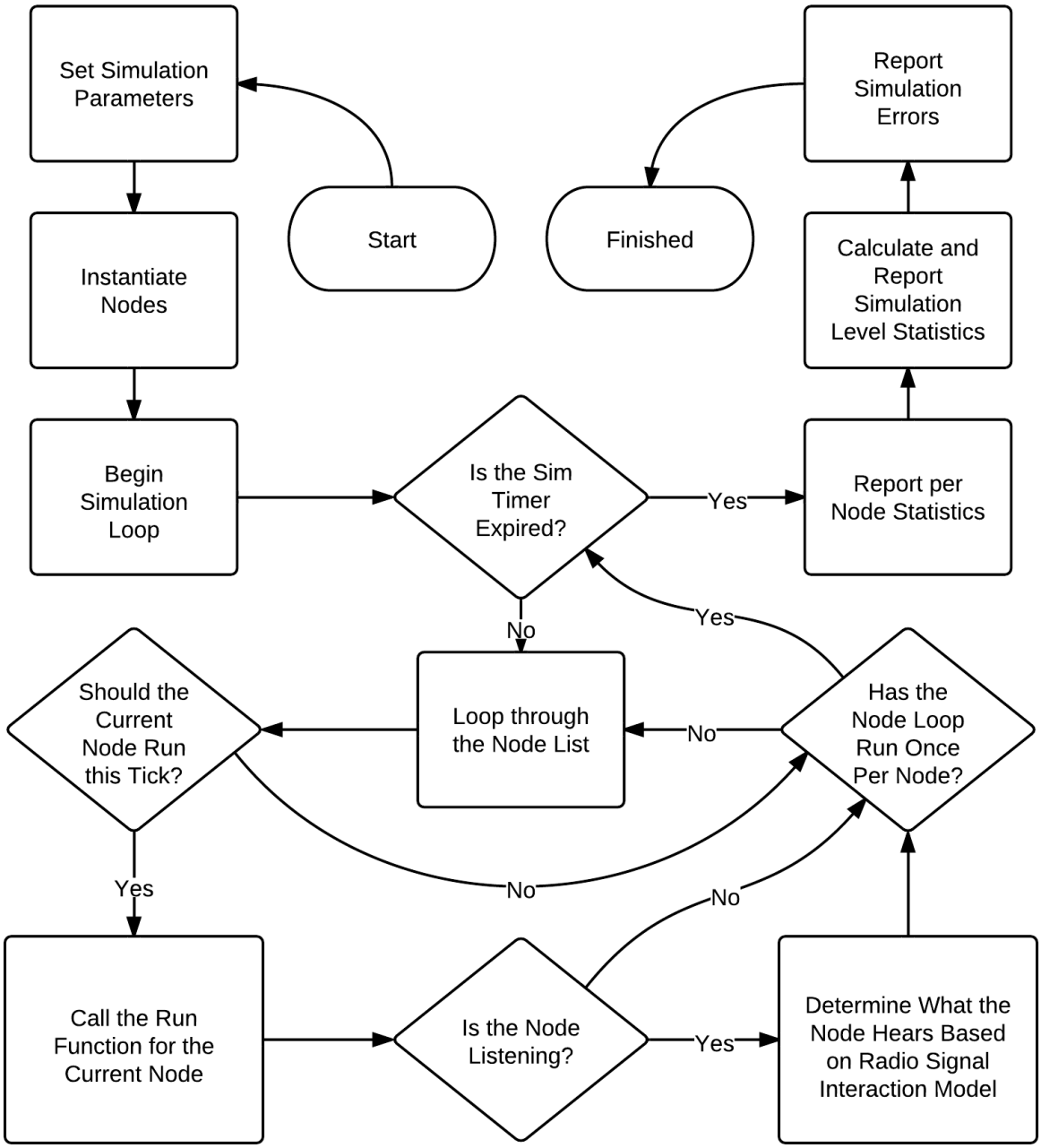

FIGURE 3.1. Simulation Core: The *sim tick rate* is the LCM of the bit rate (bps) of each participating radio. If one radio is present, *sim tick rate* equals bit rate. Each node runs once every N ticks where N is *sim tick rate* divided by node bit rate. Hence, one call to run represents the time it takes the node to transmit one bit. Most calls to run consist of waiting for timers to expire.

the MAC layer are instantiated as separate objects and manually linked to one another in a manner reminiscent of static linking in nested C. Currently we use a single layer above the MAC layer to handle routing and application level tasks such as generation of test traffic. However this could be split into multiple layers that more accurately reflect the OSI model (although considering that WSN programs frequently compress the OSI model, this may not be desirable).

3.3.1 Radio Interaction Model

When a node listens to the radio medium, the simulation determines what it hears (if anything), based on the radio interaction model. In a naive one-hop model, noise and distance are not taken into account so the listener hears the transmission of every node equally well. If one node transmits, a bit is received and if two or more transmit simultaneously, the received bit may be marked as an error or be randomly set to a 0 or 1 with a probability based on the number of interacting signals and their contents. As simple as such a model is, it is useful for testing one hop timing in which only a small amount of signal overlap destroys a transmitted packet with very high probability. We performed initial MAC prototyping with such a model.

The radio interaction model for a radio type can be made as complex as we need. A simple multi-hop model may define a communication range with an immediate cut-off for nodes beyond a certain distance. Interference may be estimated by a stochastic model based on measurements for a particular encoding scheme. A moderately complex multi-hop model would take into account distance and transmission power to determine signal strength at the receiver and the strength of interference for overlapping transmissions. A highly complex model would attempt to simulate transmissions precisely, taking into account antennae type and other physical factors. We currently use a simple distance model and stochastic model

for interference that is suitable for testing the timing of MAC protocols and energy usage based on typical transmission power settings. We discuss this in more detail in chapter 5.

3.3.2 Estimating Energy Consumption of the Radio

Energy consumption in a WSN is dominated by radio transmission and listening by a factor of 1000 to 1 or worse. Consequently we focus on recording radio usage and accurately estimating its energy cost based on the amount of time spent in various radio states, and the power level of transmissions. The energy usage varies with radio type so the radio interaction model includes an energy model for the radio indicating how much energy is required for different radio states and transmission power levels.

3.3.3 Simulating the CPU and Other Hardware

A single simulation tick of a node represents a differing amount of real-world time depending on the bits per second of the radio. As an example, the cc1000 radio of the MICA2 mote transmits at a rate of 19200 bits per second (half the symbol rate due to the use of Manchester encoding). Meanwhile the CPU, the ATmega128L, runs at 8 MHz corresponding to 416 cycles per bit. Most MAC layer processing amounts to a complex series of branch statements with a few variable updates that take significantly less CPU time than is available per tick. For basic CPU processing we record a fixed amount of energy use per second based on the CPU state (active or sleeping).

Recurring CPU intensive calculations are very infrequent on a WSN. The amount of energy use estimated is ultimately based on the amount of time spent with the CPU awake. So we can estimate CPU time for an algorithm as a percentage of real-time and cause the node to wait the correct number of ticks after a complex calculation before continuing its activity and using the result. By leaving the node awake for this artificial wait period, energy use will automatically be recorded in the proper amount.

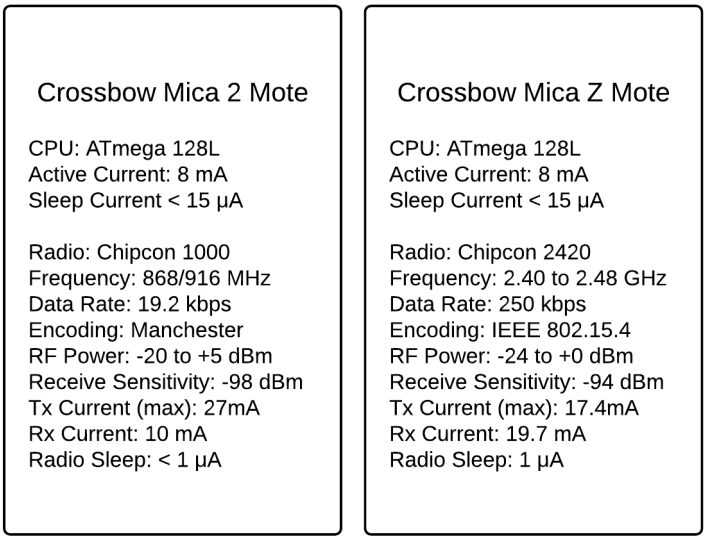

FIGURE 3.2. Node Hardware Comparison: These similar Crossbow Nodes have very different radio hardware. The Mica Z has a more even balance between transmission and reception energy, a higher bit rate and a lower energy draw per bit transmitted (despite higher current draw per second).

We note that since the Node is responsible for recording its own CPU energy usage, the general CPU estimate plus specific additions for complex calculations could be removed in favor of a different model without modifying the simulation core. We also note that on actual WSN test beds, energy use is generally calculated in the same manner that we calculate it in simulation, because performing physical measurements of energy use on a multi-node testbed is generally not practical [45].

3.3.4 Node Verbosity, Statistic Reports and Error Reporting

When the simulation medium and participating nodes are instantiated, each node has a verbosity rating set (which can be increased in mid simulation if an error occurs). As with a verbosity in command line programs, the verbosity setting allows the node to report details of activity in greater or lesser detail. Verbosity is set by flags with different layers having their own sets of flags. The flags are combined into a single integer flag variable for simplicity and adjusting parameters. The node object implements this via a stamp function which writes

the simulation time, node ID and a custom message if the verbosity is high enough. The simulation also records a full log of all node activity with verbosity stamps explicit with each message for later filtering and analysis as needed.

Each node also gathers a variety of statistics which can be recorded using a simple statistics recording function. Statistics recording demonstrates another key advantage of prototyping in simulation. Rather than optimize the functions for recording statistics that are not meant to be a part of the physical program code, we can program it in a straightforward manner and simply not account for any CPU usage in the energy model.

Finally, when the node encounters a fundamental error, it reports the error to the simulation medium object. At the end of the simulation, the medium object asks each node to generate a statistics report and finishes with some aggregated statistics and finally the number of errors allowing quick identification of any major flaws that occurred during a simulation run.

3.4 Potential Improvements

We are aware of several possibilities for upgrading the simulation in the future to increase its utility. Features such as mobile nodes are implicitly supported by the simulation already. While we have not implemented any mobile nodes, a node could update its own location quite easily and the radio model will automatically cause the connectivity between nodes to change based on the changing internode distance. A more complex model would require a node to maintain a velocity parameter and query the medium when updating velocity and location. This would allow the simulation object to implement physical obstructions that may prevent or alter movement. We present two additional ideas below.

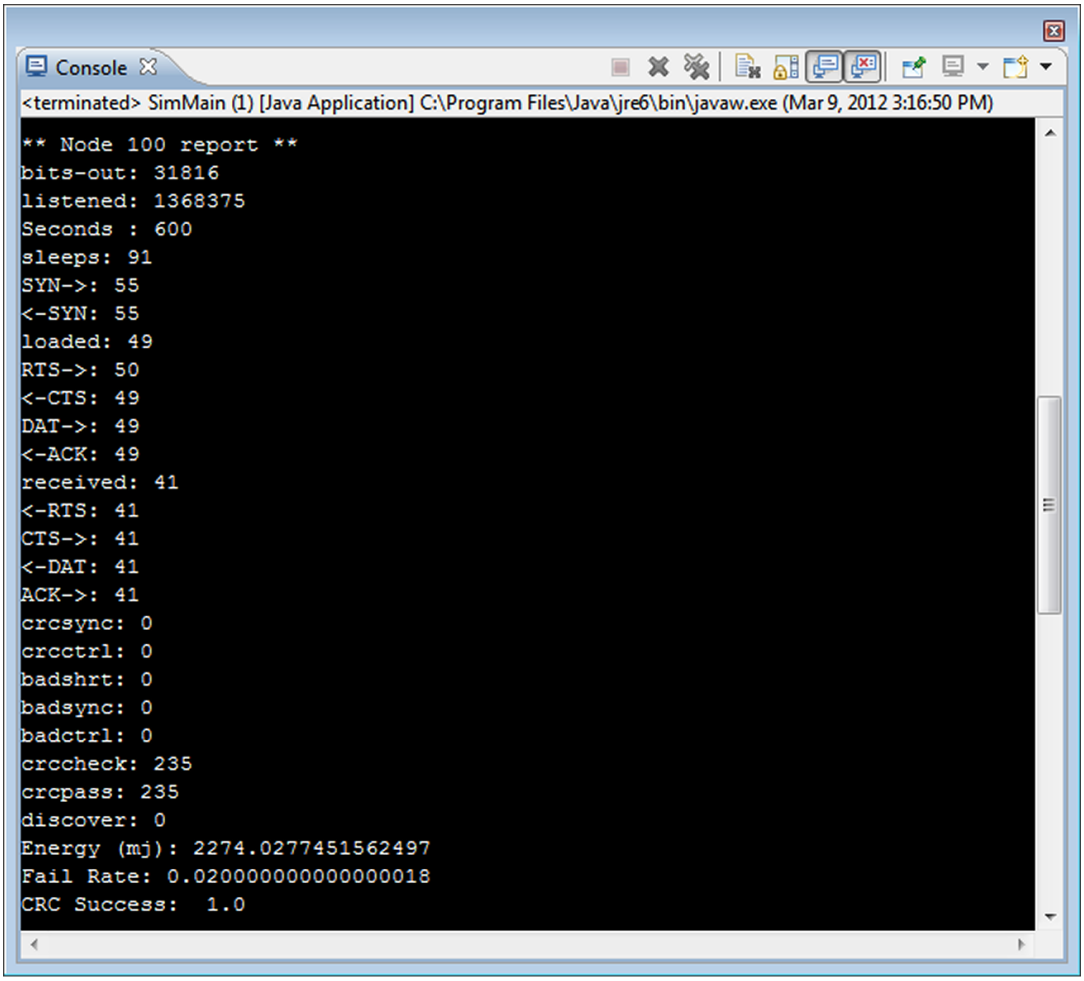

FIGURE 3.3. Simulation Report: This is a simplified report for a basic simulation of two SMAC nodes transmitting at full strength with light traffic (a few packets per minute) and a worse-case interference model. After 10 minutes, this node loaded 49 data packets, and transmitted all of them successfully (with one re-transmission due to RTS collision). The energy expenditure was 227.4 millijoules per minute. The single collision produced a data transmission failure rate of 2% as measured by outgoing RTS divided by incoming ACK packets. All incoming SYNC, CTRL and DATA packets passed the CRC check.

3.4.1 Simulation Launcher

Currently, the simulation and node parameters are set by editing constants (stored in one convenient location) and recompiling the simulation before running it. We plan to replace this with command line input that can use a GUI as an interface to launch the simulation with the desired parameters and save simulation setups for easy recall and repetition. Parameters include detailed physical and MAC layer parameters as well as node number and location. We can also implement randomly generated test networks based on a density parameter.

3.4.2 Interactive Simulation

We are also interested in developing the simulation as an interactive real-time graphical application. This could be accomplished by simply causing the simulation to pause for the correct amount of time after each tick before continuing to the next tick. We may also choose to run the simulation at consistent multiples of real-time to observe simulated network activity in more or less detail depending on what is being tested.

With a real-time graphical simulation, we can graphically represent packet transmissions, interact with nodes by pausing and adding or removing them from the simulation, changing their location and performing other changes manually. With the physical simulation expanded to include sensing, we can initiate sensing events by adding an event with a mouse click in real-time or while paused. The intention of this would be to generate a reasonable model of test traffic with ease. These changes are already possible within the simulation design except for the lack of an interface to realize the changes in mid-simulation. For example, adding a node is as simple as having the simulation object instantiate a new node according to any sort of trigger and adding it to the node list.

3.5 Remarks

With a carefully designed simulation model, we can learn many things about a MAC protocol before testing it on hardware devices. While a simulation model will always be less accurate than reality, it may still be quite useful and informative about timing details such as synchronization and message latency, and energy consumption (especially relative consumption). Effective transmission distance is probably best left to testing with physical nodes since it is heavily dependent on the particular environment.

We are currently preparing a separate publication about the novel simulator described in this chapter. It will contain some additional background material not present here, the details of the simulation in this chapter, and a subset of experimental data from chapter 5 that illuminate the utility and functionality of the simulation.

In the next chapter we present RSMAC in detail. RSMAC is a modified form of SMAC designed to be resistant to intelligent (cluster-based) jamming attacks based on transmission timing.

Chapter 4
RSMAC: Random Sleep MAC

4.1 Introduction

We propose a new synchronization method for loosely synchronous WSN MAC protocols that makes the network robust against energy efficient jamming. It also allows network wide synchronization, preventing any node from taking on an asymmetric energy burden due to maintaining two or more MAC layer schedules. Additionally, random sleep eliminates the need for settled nodes to engage in explicit neighbor discovery, a source of energy waste needed to admit new nodes into a WSN.

The potential downside is variable latency, slightly larger SYNC packets (thus higher energy and bandwidth overhead) and a slightly larger memory footprint for more complex program code and data.

First we will review the core features of SMAC, describe a refined version, then look with more detail at the features that will be changed by the new synchronization scheme. Finally, we explore the secondary implications of changes that define RSMAC on optional features like message passing before moving onto experimentation using our simulator.

4.1.1 SMAC Design Overview

We have developed a refined form of SMAC to represent loosely synchronous protocols in general and conceive RSMAC as a modified form of this refined SMAC. We make basic improvements to the implementation details so that a clearly defined and robust base is provided for developing RSMAC. This also makes experimental comparisons between RSMAC and our version of SMAC more instructive as the differences between them are kept strictly to novel features that rely on additional information transmitted in the upgraded SYNC

packet. Hence, we will begin by looking at key features of SMAC noting our refinements and specifying precise implementation details.

4.1.2 SMAC Core Features

CSMA-CA (RTS-CTS, DATA-ACK) with exponential backoff is used to contend for the medium and exchange packets.

Contention occurs within a fixed window which is a subset of time within a loosely synchronized frame. Outside the contention window, the node sleeps to conserve energy. The sleep time is usually significantly larger than the contention window to extend the network life time. The regular listen and sleep cycle defines a schedule, and the schedule is maintained (against clock drift) by transmitting regular SYNC packets.

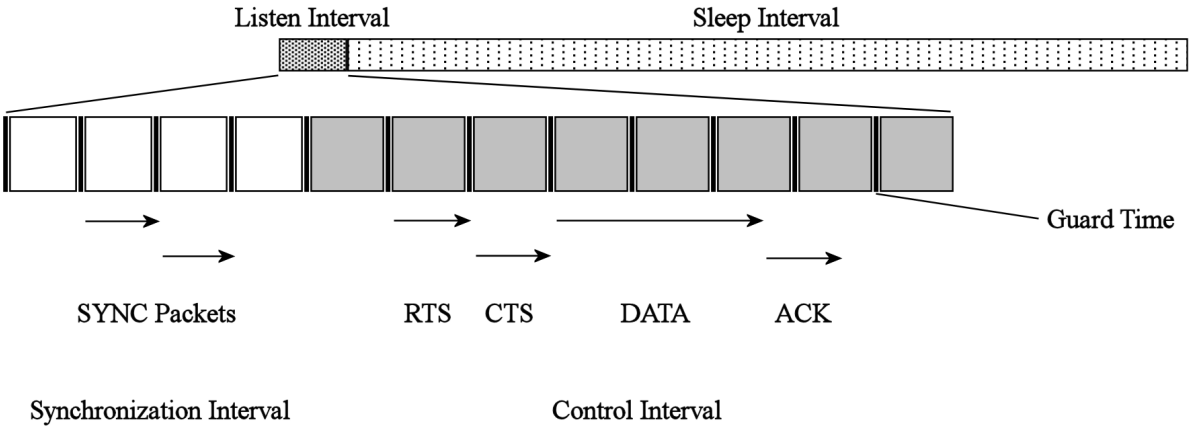

FIGURE 4.1. SMAC Frame: Nodes choose a slot to transmit SYNC or CTRL packets. The slots are sized to match the packets, however once an RTS has been heard, slot boundaries are ignored. If a DATA packet is long enough, it will continue transmitting into the sleep interval.

Nodes must perform neighbor discovery at startup and periodically thereafter by spending an extended period of time listening for SYNC packets. This allows the network to configure dynamically at startup, and reconfigure dynamically if a node is lost, if new nodes

are added, or if synchronization is lost. This is necessary because two neighbors following misaligned sleep schedules may never hear each other. A node tries to minimize the number of schedules adopted, but may adopt several. Nodes with multiple schedules must either send all broadcast packets twice as often (including the regular SYNC packets) or else adopt additional listen intervals (the simpler but more energy hungry of the two choices with present send/receive energy ratios).

Adaptive listen was presented as an optional feature that causes a node to sleep after hearing a CTS–standard operation–but then to wake again when the transmission is finished in case it is the next-hop node in a multi-hop transmission (based on the estimated time). However, this feature is vital to achieve low latency and high throughput in SMAC. Without it, nodes can only send or receive one packet per listen period (regardless of its length and current traffic density) reducing multi-hop latency to roughly the frame size multiplied by the number of hops. The temporary sleep length is based on an estimate of the data transmission length encoded in the RTS packet. We implement and test this feature as an essential part of SMAC and RSMAC rather than an option.

Message passing is an optional feature that breaks large data packets into fragments. The receiver sends an ACK for each fragment allowing a corrupted fragment to be retransmitted without discarding the entire transmission (a possible source of energy waste and latency). Each ACK contains the most recent estimate of transmission duration (the total of which may increase if any fragment transmission fails). We do not test message passing since it presents no new challenges for RSMAC.

4.2 Detailed Review of SMAC and Refinements

When SMAC nodes boot, they organize themselves into virtual clusters where each node in the cluster is coordinated to follow the same basic schedule. The schedule divides time into a

series of frames with each frame consisting of a short listen interval and a much longer sleep interval. This decreases energy use at the expense of higher latency.

The listen interval within a frame is divided into a synchronization period (SYNC) and a control period (CTRL). The SYNC period is for sending and receiving broadcast packets necessary to maintain the schedule in the face of clock drift, and to advertise the existence and schedule so that automatic network configuration can occur.

4.2.1 The SYNC Period

SYNC packets are not broadcast every frame, but only often enough to correct for clock drift. The upper bound on SYNC frequency is one packet per frame and the lower bound is determined by the clock drift rate (the frequency must be high enough to maintain synchronization).

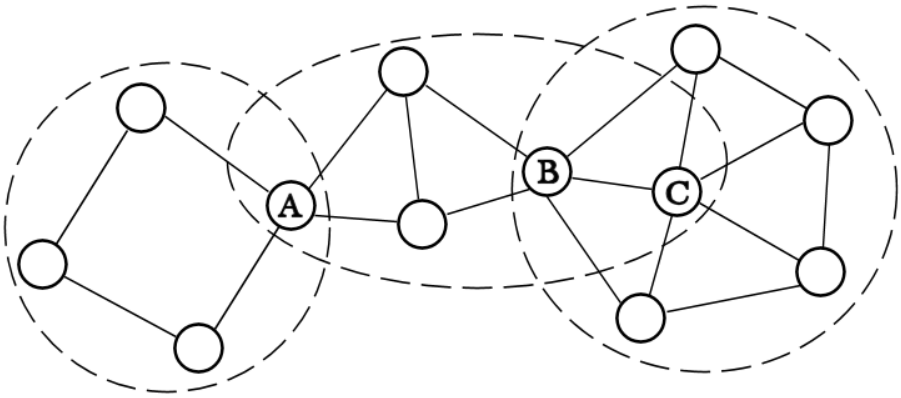

FIGURE 4.2. SMAC SYNC Clusters: Once an SMAC node has a neighbor, the shared schedule is marked as primary. Nodes that discover additional neighbors with other schedules maintain secondary schedules and use more energy. Early failure of B or C would partition the network, and any node may be vital to the sensing application. Hence, we must promote energy fairness when possible.

The SYNC period is divided into S_{num} slots where $S_{num} > 1$ and is sufficiently large to minimize collisions between neighbors. Each slot is the length of one SYNC packet and preceded by a small guard time to pad for clock drift between synchronization. All nodes

listen for the full SYNC period except during a slot when it transmits a SYNC packet. If two nodes choose the same SYNC period to transmit in, there is $1/S_{num}$ chance of transmitting in the same slot causing the packets to be missed by both transmitters and any neighbor that can hear both packets (a collision). This could lead to slight desynchronization if it occurs many times and uncorrected clock drift accumulates. However, the network can tolerate some drift. A successful SYNC transmission will occur with very high probability before significant desynchronization occurs and with virtual certainty before complete desynchronization occurs.

The SYNC packet maintains synchronization by sending the time remaining until the sender's sleep period starts. With known transmission time (based on fixed SYNC packet length) subtracted from this relative time stamp, the receiving node can make any adjustment needed to remain in SYNC with its neighbor.

4.2.2 The CTRL Period

CTRL packets are sent during the CTRL period which follows the SYNC period and is also divided into slots. The number of slots (and thus the size) of the CTRL period must be sufficiently large to allow reliable transmission and contention between multiple nodes without being so large as to use energy unnecessarily. Higher node density will require more slots to keep the rate of RTS collision sufficiently low during times of high traffic.

CSMA-CA is used within the CTRL period for contention and collision avoidance solving the hidden terminal problem for DATA transmission by sending short RTS packets. Nodes contend for the medium by choosing a random slot (slot size equal to CTRL packet size) from within the current backoff period (which defaults to 1 slot), listening until that number of slots has passed, and then transmitting if another node did not begin transmitting first. The node starts the transmission with an RTS packet, which ideally informs all listeners

that a transmission will occur and which node is the target. The targeted node stays awake (even if the sleep interval arrives), transmits a CTS packet and then waits for the original node to transmit the DATA which is the length of a CTRL packet plus the payload of data. The data is replied to with an ACK. Note that the original RTS packet is subject to collision caused by hidden terminals. Loose synchronization, besides allowing for coordinated sleeping, decreases collisions during exponential backoff.

SMAC contains on optional feature whereby long DATA packets are be broken into fragments with an ACK from the receiver punctuating each fragment. This continually informs other nodes within range of the sender and receiver that a transmission is occurring and should not be interrupted (attaining higher transmission reliability at the expense of latency and minimum energy consumption per transmission, though not necessarily overall energy consumption). This is made more vital by possibility that two nodes may not be synchronized if they are on the border of two different virtual SYNC clusters. That is not an issue in RSMAC because we maintain network wide synchronization.

Alternatively, the maximum packet size can be lowered, a long packet could be broken into separate packets (each requiring the full RTS/CTS exchange), or a long packet can be transmitted without division (risking the entire packet transmission failing due to unexpected interference). The optimal choice depends on the application and network characteristics.

4.2.3 Packet Fields and Size

The SYNC packet fields are length, type, Node ID, Sync ID, remaining SYNC interval, and a cyclical redundancy code (CRC). The length (in bytes) is a physical layer field that tells the radio when to stop accumulating bits (and may be used by other layers). Type is used to distinguish SYNC packets (and is also the first field in CTRL packets to distinguish different types of CTRL packet). Node ID identifies the sender. A destination is not specified

since SYNC packets are broadcast for all nodes. Sync ID indicates the original source of the schedule that is being followed and functions as a schedule identifier distinguishing different schedules for border nodes that maintain more than one allowing the formation of virtual clusters. Remaining SYNC interval functions as a relative time stamp for the schedule. The CRC is used for error detection.

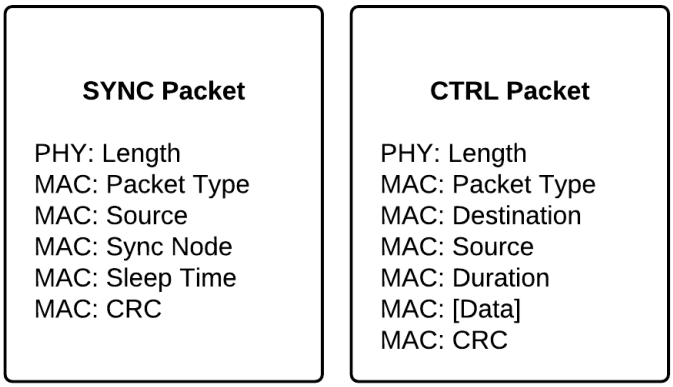

FIGURE 4.3. SMAC Packet Fields: Packet size depends on field size of Source, Destination and Sync Node (which are determined by max network size).

The CTRL packet fields are similar to SYNC fields: length, type, destination, source, duration, and CRC. A DATA packet is identical except the payload occurs after duration and prior to the CRC. Destination is included since CTRL packets are unicast, and SYNC information is left out (Sync ID and remaining SYNC interval). Duration is the expected length of the total transmission (if it goes as planned) and is a function of the DATA payload length and the fixed CTRL packet lengths.

The size of each field may be dependent on operating system characteristics, and total packet size will be dependant on the radio hardware and the physical encoding scheme. We will discuss packet size in greater detail in relation to specific hardware radios and physical layer implementation details in the next chapter.

4.2.4 Selection of CTRL Slot and Random Exponential Backoff

We explained above that each node chooses a slot within the current backoff size to begin contention. Backoff is triggered when RTS collision occurs due to the hidden terminal problem or identical slot selection. The sending node transmits the RTS, listens for the CTS in the next slot and then schedules a new RTS if the CTS is not heard. The slot after the CTS should have occurred is considered to be *slot 0* and then a new slot is chosen by the following method. The backoff b is a randomly chosen integer such that $0 \leq b \leq (2^c - 1)$ where c is the number of prior collisions.

Random exponential backoff was originally designed to be used in CSMA-CA algorithms without significant energy constraints. The purpose is to gracefully handle network congestion allowing traffic to continue without accidentally coordinating repeated collisions. The number of collisions c is expected to increase proportionally with traffic density hence the backoff window b increases and eventually allows traffic to contend without collision. We question the use of the basic algorithm in conjunction with coordinated sleeping without performing optimizations.

We consider a modification where the initial random slot is biased toward the earlier contention slots instead of randomly chosen from all remaining slots. The reason is that if two nodes choose the same late slot, there may not be many slots left before sleep in which the backoff algorithm can function. We'd prefer a collision in the earlier slots. Additionally, the sooner a transmission occurs, the sooner overhearing nodes can sleep and save energy. If backoff is needed and a value of b is chosen that exceeds the sleep boundary, the node simply goes to sleep and saves the packet for the frame beginning after the sleep period. However, if half of the last slot passes with no transmission, each node will randomly make one last attempt to transmit with a probability of $1/c$ so as not to waste the last slot before sleep

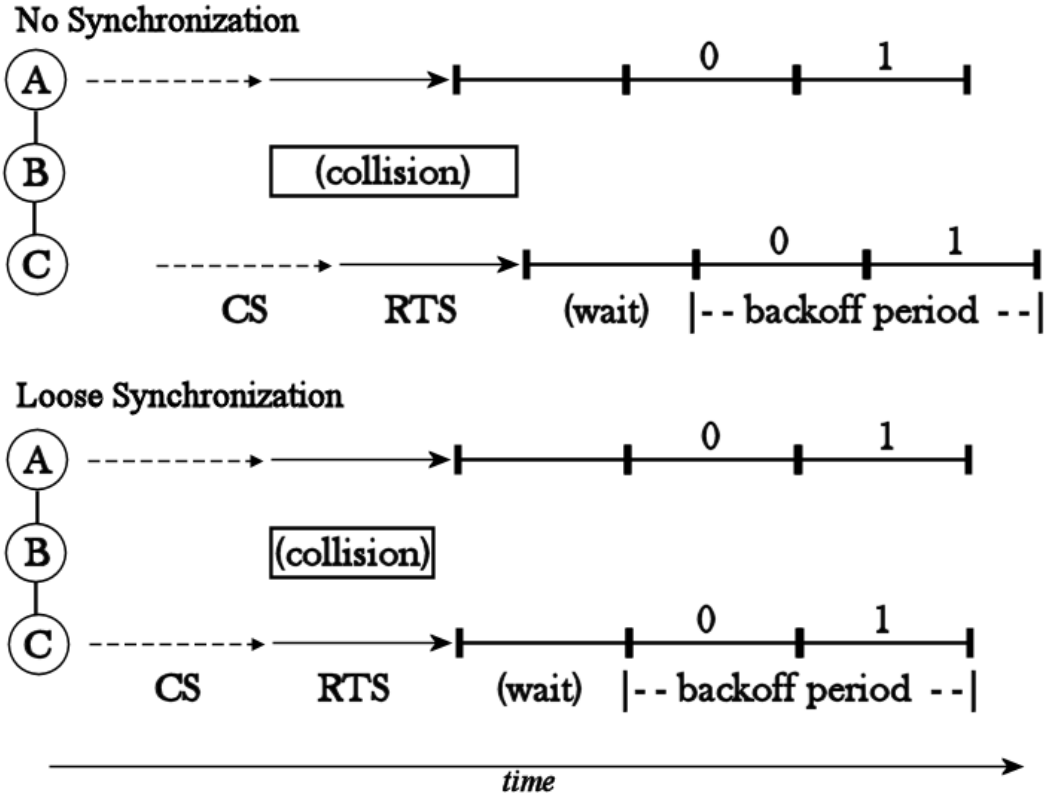

FIGURE 4.4. Synchronized Backoff: The first example shows basic 802.11 and other CSMA-CA protocols in which there is no slot synchronization. Packets from A and C are hidden from each other and may collide at Node B. RTS collision requires a wait to ensure a CTS isn't coming then a backoff period. The later of the two nodes is guaranteed not to win contention (a reasonable outcome) and has a 50% chance of causing a second collision based on whether the early node chooses slot 0 or 1. The total chance of success in the first backoff round is only 25%. Neither A nor C knows whether it is the late node, so this cannot be avoided. In the same example with loose synchronization, there is a 50% chance of success and both nodes have an even chance of winning contention. If nodes become unsynchronized, they continue to function but with the reduced efficiency of the first example (until synchronization is restored).

(recall that if a transmission begins in the last slot, the listeners will stay awake to finish a transmission if they hear the start of it).

The SMAC original SMAC experiments set the SYNC and CTRL period to a sizes to 15 and 31 slots without discussion or analysis to support the decision. Since the slot number defines the listen interval which dominates energy consumption, we seek to shorten it as much as possible. If the number of slots is very low compared to the number of neighboring nodes in a particular part of the network, those nodes could become so congested that transmission rates drop precipitously. We note that this could be solved by extending the backoff count into the next frame allowing nodes to be optimized for the far more common condition of low traffic and moderate density. We also expect that WSN routing protocols are density aware and schedule traffic flows to avoid congestion when possible. We will perform some tests for analyzing the number of SYNC and CTRL slots under different conditions in the next chapter.

4.2.5 Guard Time and Sleep Time

The guard time is simply a buffer to guard against clock drift that occurs between SYNC and CTRL packet slots. All nodes listen during the guard time before each slot, but will not transmit until the guard time expires. The small amount of extra time spent listening must be balanced against frequency of SYNC packets. The longer the guard time, the fewer sync packets must be sent to keep clocks synchronized within acceptable limits. The SYNC rate and guard time must be set with the clock drift rate in mind. Guard time is proportional to clock drift rate. As noted above, SMAC continues functioning even if synchronization difference temporarily exceeds the guard limit, so we don't have to set the guard time aggressively.

As an example, the *maximum* drift measured on a MICA2 mote is $40\mu s$ (about 80 percent of the time it takes to transmit one bit: $52.0833\mu s$), hence we can choose a guard

time of 200μs and maintain synchronization in the worst case with a SYNC packet every 5 seconds or less. However, we could set the guard time for a more average case to conserve energy. It also possible to use statistical analysis to estimate and account for excessive clock drift. We will test with a guard time of 208.3μs for our MICA2 simulations.

The node sleep time dominates the frame occurring after the last CTRL slot and extending to the end of the frame. The duty cycle is defined as the listen interval divided by the total frame size (listen plus sleep interval) expressed as percentage.

4.2.6 SMAC SYNC and Neighbor Discovery

It may be useful to SYNC more frequently than necessary for correcting clock drift in order to shorten the listen period for reliable neighbor discovery. We define reliable neighbor discovery as a period encompassing the longest possible time between any two SYNC packets plus one frame and in which the discovering node does not respond to CTRL packets. There is an optimal setting for SYNC frequency that minimizes the combined energy of sending SYNC packets and performing reliable neighbor discovery. This depends on the number of SYNC slots which itself depends on node density. There is a trade off between the length of reliable neighbor discovery and packet transmission and latency, because reliable neighbor discovery requires the discovering node to ignore any incoming data transmissions and listen exclusively for SYNC packets. We can use reliable discovery for some temporary period when nodes are booted so that they join or form a network quickly, and less reliable (short) neighbor discovery to repair unexpected synchronization errors.

4.2.7 Virtual Cluster Formation

In its original form, SMAC forms clusters based on a few rules. When a node boots up, it listens for other schedules for a period of time before adopting its own schedule and beginning to transmit SYNC packets. The transmitted information directly related to synchronization is

remaining SYNC interval, Node ID, and Sync ID. When a node discovers its first neighbor, it adopts the neighbor's schedule and abandons its own. Once a node has at least one neighbor, any new neighbor with a different Sync ID will result in secondary schedules being stored in a schedule table. A SYNC packet may indicate that a neighbor has adopted a different schedule which must be reflected in the schedule table, and the node must consider abandoning its own schedule again if it eventually happens that no other nodes follow it.

4.2.8 Intercluster Communication

With one or more secondary schedules, a node will wake to send periodic SYNC packets to the secondary node without any additional listening. However, if it has a transmission to send to a neighbor on a secondary schedule, it must wake and choose a random contention slot from the neighbors CTRL window, perform carrier sense in the previous slot to ensure a transmission is not in progress and spend extra time listening when backoff occurs. This causes an asymmetric increase in energy use that scales linearly with the number of secondary schedules. The virtual clusters will tend to remain in place indefinitely, so this can affect network lifetime if the overburdened nodes are part of exclusive routes or vital to the sensing application. Separate virtual clusters may also have listen intervals that partially overlap causing them to interfere with each other regularly (they would likely have misaligned slots and SYNC packets may be transmitted in the CTRL window and vice versa).

4.2.9 SMAC Packet Encryption

We recall here that the intelligent jamming attacks we are working against use statistical analysis of the timing of network traffic, not detailed knowledge of the packet and its contents. These cluster based attacks assume that the victim node is already using packet encryption. Without encryption, a class of attacks that uses SYNC and CTRL packet information can be implemented. However, since RSMAC may have benefits outside of security, we will test

both protocols with and without packet encryption as an energy expense. We discuss the energy and timing cost of software encryption in the next chapter.

4.3 RSMAC: Adding Random Sleep and Global Sync

The primary purpose of RSMAC is increased security, particularly against the threat of energy efficient jamming nodes. In solving this problem for loosely synchronous protocols, we were forced to transmit additional information in the SYNC packet and consequently felt compelled to look for ways to reduce energy consumption. We discovered a way of using the additional information in the upgraded SYNC packet to achieve a network wide loose synchronization that eliminates the need for any node to maintain more than one schedule. This also eliminates the problem of two schedules randomly colliding at virtual cluster borders. RSMAC improves network lifetime by saving energy at nodes with the highest MAC layer energy consumption. With global synchronization, the virtual cluster grows to encompass the entire network so that each node has a symmetric energy burden at the MAC layer. Additionally, the formal neighbor discovery period is not needed after an initialization period, so we also save energy and improve transmission reliability by abandoning it.

The additional information transmitted in the SYNC packet is the seed state of the pseudo-random number generator (PRNG) which functions as an index into a cycle of pseudo-randomly generated frame lengths. With a long enough cycle, the frame length becomes unpredictable for outside observers preventing efficient jamming. This extra transmission will require more energy overhead to maintain synchronization. The PRNG must produce sufficiently long patterns to eliminate the possibility of statistical analysis and that depends on the PRNG algorithm and size of the seed state. So we expect the packet size to grow, but the exact percentage increase depends on length of the packet with all headers

and preamble and the size of the PRNG seed chosen. We discuss this in greater detail in the context of implementation on specific hardware and software in the next chapter.

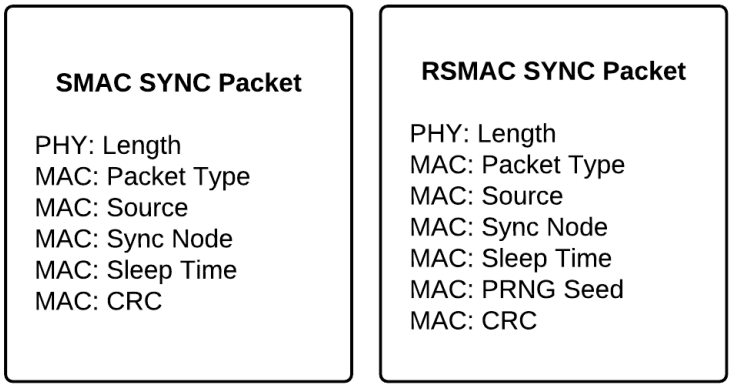

FIGURE 4.5. RSMAC SYNC Packet Fields: We must add the pseudo-random number generator's seed state to the SYNC packet.

4.3.1 Using a Random Number Table

Although we have explained the algorithm in terms of a PRNG that uses the CPU to calculate random numbers, on current sensor motes a more likely situation is to use a table of randomly generated numbers and transmit the index into the table in place of the seed state. PRNG tables are commonly used on WSN motes to save energy and CPU time [15]. For our actual implementation, we will use a table of random numbers (independent from the table used to generate random numbers for standard functions such as choosing slots). As an example, a table with a cycle length of 2048 at 2 bytes per sleep length would take up 4KB and require transmitting an 11 bit index as the PRNG seed state (or 2 bytes if rounded up). This is only 3 percent of the program flash memory on a MICA2 mote. If we can save energy for a small increase in program memory consumption, the tradeoff is generally worth it on energy constrained motes.

4.3.2 Short Cycle of Random Numbers

Although the packet size may increase only slightly due to a long period PRNG seed, we can shorten the packet by using the shortest seed state possible for the PRNG and packing the bits. However, a very short cycle time could allow the network's pseudo-random frame cycle to be observed by a highly sophisticated jammer. Even so, in a loosely synchronized network, SYNC packets are not transmitted every frame and are transmitted at a randomly chosen slot as well. This should functionally increase the uncertainty and make a short PRNG cycle more difficult to detect. It will also increase the number of unique encrypted packets that must be observed. Precisely how short a pattern is safe with loose synchronization depends on the sophistication of the jammer and is a question we leave to future jamming research. We will test with a transmitted seed state of 2 bytes. For a short table, this leaves unused bits that we could fill with random values to further confound analysis of encrypted SYNC packets (although a well chosen encryption algorithm may eliminate the need for this).

4.3.3 Following a Random Sleep Schedule

It is a relatively trivial matter for a node to continue following a pseudo-random sleep schedule that it is already following. When calculating the length of its next sleep period, instead of reading a fixed value, it reads the next PRNG value. When a node receives a SYNC packet, it checks to ensure that the transmitted seed value (functioning as an index into the schedule cycle) matches its own. If the seeds match, this confirms the two nodes are in the same part of the schedule cycle so it is safe to make updates for clock drift. If the seeds do not match, the node ignores the SYNC packet.

The length of the random sleep interval should be specified to occur between some minimum and maximum with a uniform distribution. The value halfway between the minimum would represent the average length of the sleep time and can be used to calculate

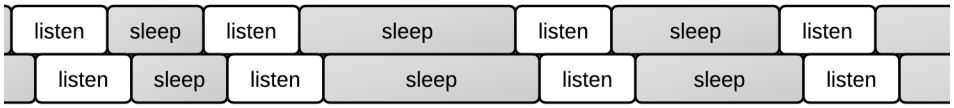

FIGURE 4.6. RSMAC Random Sleep: Listen and Sleep pattern for two nodes that are significantly out of sync. Notice the length of the listen interval is fixed. In normal operation, nodes should stay much more tightly synchronized than this graphic suggests. The node hearing a SYNC packet first will alter it's *time until next listen* to match the sender's schedule, thus resynchronizing.

the average duty cycle. We use a minimum value near zero since this is necessary to enable reliable implicit neighbor discovery (see figure 4.9 on page 60).

4.3.4 RSMAC Initial Neighbor Discovery

The the initial neighbor discovery period functions similarly in SMAC and RSMAC. The length of neighbor discovery time is predefined as a minimum value instead of a precise value. The discovery period may encompass several frames. Since a pseudo-random schedule will probably not match this minimum length exactly, the actual discovery time will exceed the minimum by as much a the maximum randomly chosen frame length.

If a superior schedule is found (according to the global sync algorithm below), the neighbor discovery time may have to be recalculated according to the random schedule generated by the new seed. Recalculated discovery should also meet or exceed the minimum length minus the time passed since discovery started.

4.3.5 RSMAC Implicit Neighbor Discovery

The explicit neighbor discovery period is used when a node is starting up and until a node has gone a predefined amount of time without adopting a new schedule. This is to facilitate network formation and attaching new nodes to an existing network.

However, neighbor discovery must still occur in SMAC throughout the network lifetime so that the network can reorganize itself automatically if the network changes. In SMAC,

this requires the continuing usage of an explicit discovery period. The reason is that with a fixed duty cycle, neighboring nodes with different schedules may never be awake at the same time.

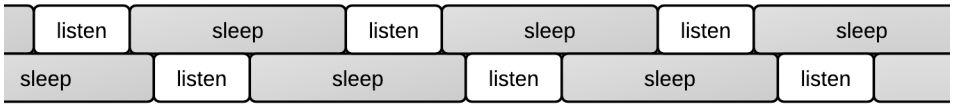

FIGURE 4.7. SMAC Coordinated Miss: SMAC neighbors have the same constant frame length and may never discover each other without explicit discovery.

In RSMAC the randomized duty cycle means that nodes with different schedules have a chance of randomly overlapping duty cycles which may eliminate the need for explicit neighbor discovery if the frequency of random overlap is high enough.

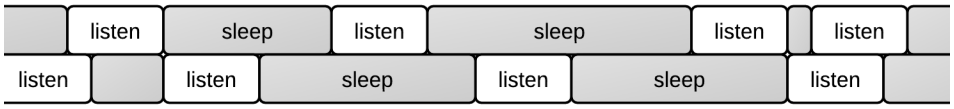

FIGURE 4.8. RSMAC Random Neighbor Discovery: two completely unsynchronized nodes will eventually have partially overlapping listen intervals.

The probability of two unsynchronized listen intervals randomly overlapping so that the SYNC interval of one node is completely within the listen interval of the other depends on the size of the listen interval and SYNC interval (which are fixed) and the maximum size of the uniformly distributed random sleep interval. Provided the minimum sleep time is near zero, under these conditions the probability of SYNC period overlap is given by: $(listen\_interval - sync\_interval)/max\_sleep$. Listen minus SYNC is identical to the size of the CTRL interval, and since we define our duty cycle by the average sleep length and always use a minimum random sleep of nearly zero, we can more simply and usefully express the formula as $ctrl\_interval/(2 \cdot avg\_sleep)$.

We might also have success with as little as a single SYNC slot of the sender overlapping with the listener. The listen interval overlapping is not enough for neighbor discovery to take place however. A SYNC packet must be transmitted by one node and heard by the other during overlap. This transmission may take place in the listeners CTRL interval, and hence collisions may occur (with likelihood increasing proportionally with traffic density).

The minimum probability is the chance of one sync slot overlapping and a sync packet being transmitted in that slot, while the maximum is the chance of the entire sync interval overlapping and a sync packet being transmitted anywhere during that frame. However, this does not take into account the chance of collision.

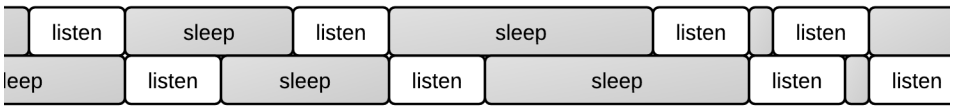

FIGURE 4.9. Coordinated Random Miss: If the minimum random sleep length is longer than the listen interval, two nodes may miss indefinitely just as in SMAC if they become desynchronized by more than the listen interval but less than the length of the minimum sleep interval. We set the minimum listen interval near zero to enable reliable implicit neighbor discovery.

When a node discovers a node with a different schedule via this method, it can adopt the schedule immediately according to the global synchronization algorithm, or if the nodes are already settled into the global schedule, then the nodes will use the algorithm for securely restoring lost global synchronization.

4.3.6 Global Synchronization Algorithm

As we noted in the section above on SMAC synchronization, SMAC forms virtual clusters according to several rules. RSMAC instead propagates a single schedule throughout the network and maintains it by using the PRNG seed as a unique identifier of the schedule. This requires a complex algorithm to ensure that the root schedule propagates to the entire

network, that it is maintained, that new nodes can join, and that it is robust against failure, tampering and jamming.

We describe the algorithm for global synchronization below. Synchronization is created and maintained through SYNC packets which transmit the PRNG seed, remaining SYNC interval, the Node ID of the sender, and the Sync ID (ID of the node originating the schedule being transmitted).

1. Each node boots in the neighbor discovery period and listens for SYNC packets. If a node hears a SYNC packet before it has a schedule, it adopts that schedule without question and continues discovery for the allotted time.
2. If a node does not hear any SYNC packets during initial discovery, then it chooses a starting PRNG value thereby creating its own schedule. It uses its own unique Node ID as its Sync ID.
3. A node will abandon its own schedule for an incoming schedule if the incoming schedule has a lower Sync ID.
4. If a node hears a higher Sync ID, it broadcasts SYNC packets during the SYNC interval defined in that packet for a few frames. This helps the network achieve global synchronization more quickly.
5. If a node hears a packet with equal Sync ID, it checks to verify that the PRNG seed is identical to its own. This is the expected situation in a settled network and the receiver can safely use the packet's timestamp to refine its sleep time for clock drift.

Using this algorithm, every node in the network should eventually follow the schedule of the node with the lowest Node ID. Nodes added to a settled network should quickly join because the initial neighbor discovery period is long enough that a neighboring SYNC packet

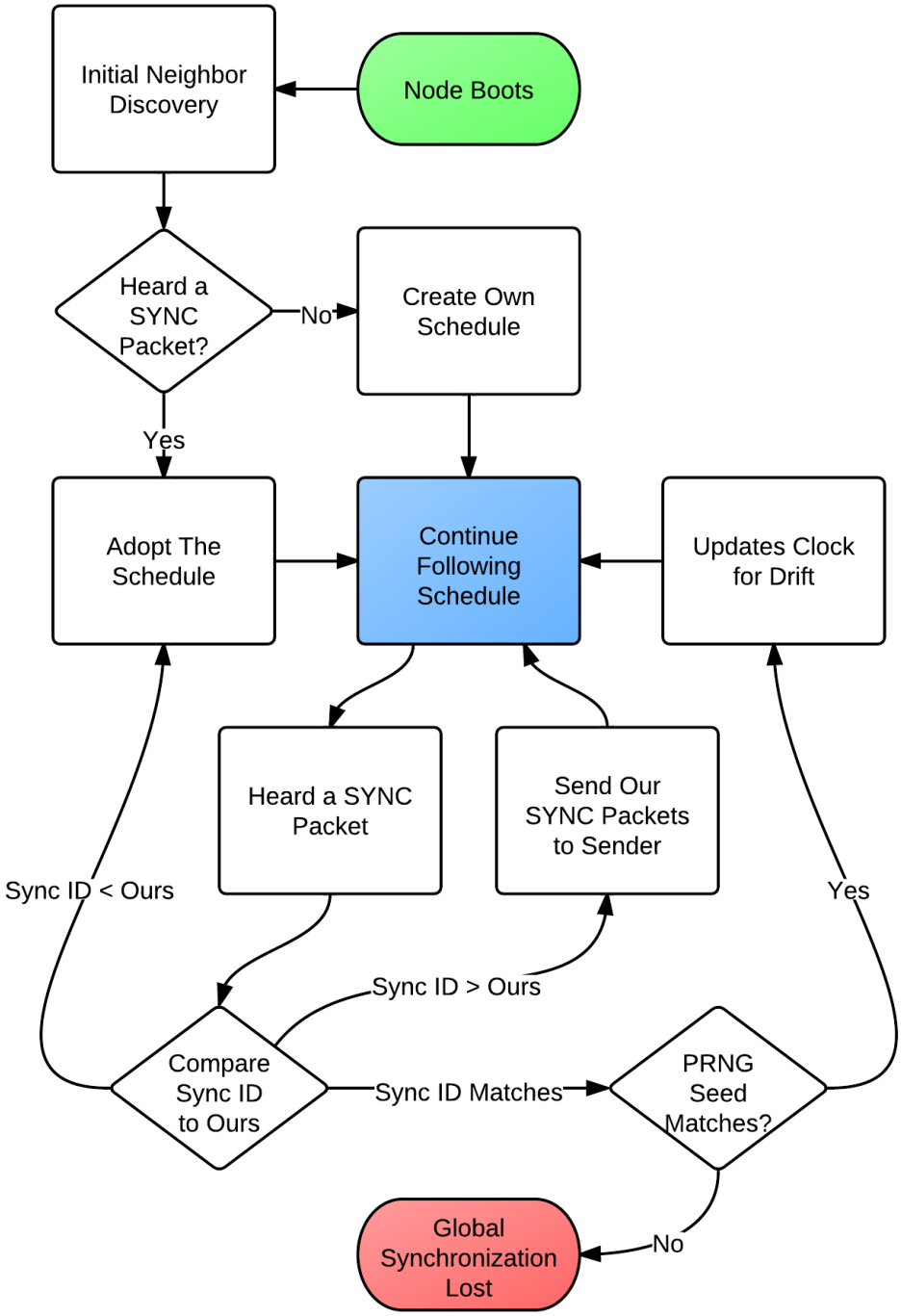

FIGURE 4.10. Global Sync Algorithm: The nodes boots in the initial neighbor discovery period and eventually settles into what is presumably the global schedule. If the PRNG check fails in a settled network, the node begins the process to restore lost global synchronization.

is almost sure to be heard. New nodes should be deliberately given a Node ID higher than the lowest ID in the network. This is not strictly necessary, but if it does not join the existing network during initial discovery and the new node has the lowest ID, it will propagate its new schedule through the already settled network.

After an initial timer has expired, a node will quit participating in periodic neighbor discovery. This is by design since there should be no new schedules to adopt once the global schedule has propagated, and new nodes will join the existing network using their initial discovery period. If two neighbor are somehow following a different pseudo-random schedule after settling, their SYNC intervals will randomly align periodically working as a functional replacement for neighbor discovery allowing the node to follow the procedure below for securely restoring lost global synchronization. After a longer initial period (on the order of several minutes) nodes no longer adopt schedules from nodes with lower SYNC IDs. This is a security feature.

4.3.7 Securely Restoring Lost Global Synchronization

If a node hears a packet with equal Sync ID, but the PRNG seed does not match it's own, this may be an indication that the network has somehow become partitioned into two or more virtual clusters which have the same Sync ID and hence will not merge according to the previous algorithm. This should be a rare occurrence. It might occur if a node that forms an exclusive path fails and is not replaced for a while (allowing clock drift to accumulate to the point that the two virtual clusters no longer have matching PRNG seeds despite having the same Sync ID).

We propose a method of restoring synchronization under these conditions that is robust against tampering by deceptive jammers. The algorithm for restoring lost global synchronization causes a node that detects lost synchronization to undergo a verification

process with the sender of the packet that was out of sync. If successful, the node adopts the new schedule and propogates it to any neighbors it had. Those neighbors recursively propagate the new schedule to the rest of the cluster that was also following the old schedule.

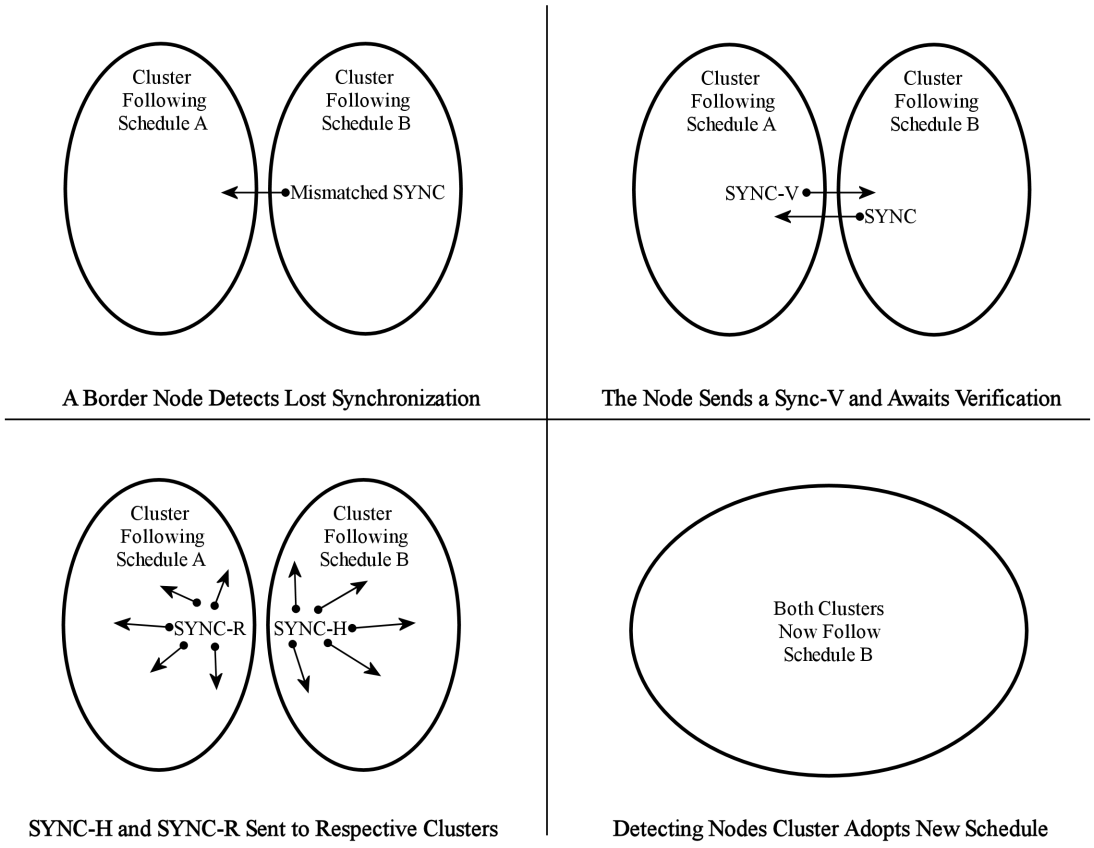

FIGURE 4.11. Global Sync Restore Example: A border node of one cluster will initiate the restoration process. If all goes well, one of the two clusters will join the other cluster.

1. A node that hears an out of sync neighbor with identical Sync ID temporarily adopts the secondary $schedule_B$ from this SYNC packet along with its own $schedule_A$. It broadcasts a verification request in a SYNC slot of $schedule_B$. This would be heard by any node in range that is part of the $schedule_B$ virtual cluster. If there is no apparent response, it tries once more in the next frame. The purpose of this packet is to simply increase the SYNC rate of the $schedule_B$ node in order to increase the speed of recovery.

We call the verification packet SYNC-V. It is a standard $schedule_B$ SYNC packet with a special value in the type field indicating that it is a SYNC-V rather than an ordinary SYNC packet. We will introduce a few similar packets below that are also distinguished only by the type field.

2. A node hearing a SYNC-V simply increases its SYNC rate temporarily and will not heed a SYNC-R (reset request) for a period of time. This keeps it from accidentally going through the reset process that the sender of the SYNC-V request initiated. This is necessary in case the SYNC-V sender is not an exclusive connection to the virtual cluster following $schedule_A$ which might allow a reset to propagate into the $schedule_B$ cluster at the same time that $schedule_B$ is propagating into the $schedule_A$ cluster. The verifying responders transmit $schedule_B$ SYNC-H packets for a few frames and sets its cool-down timer. A node with an active cool-down timer treats special SYNC packets as ordinary, and does not send a SYNC-V when discovering an incorrect PRNG seed.

3. A $schedule_B$ node hearing a SYNC-H propagates it and *holds* its schedule by setting its cool-down timer. It also propagates the SYNC-H for a few frames. These nodes are simply waiting for the network to settle.

4. When the node on $schedule_A$ receives 2 SYNC packets with the correct seed and at the expected time according to $schedule_B$, it has confidence that it has detected a legitimate node in a different virtual cluster. It sets its primary schedule to $schedule_B$ and takes it upon itself to propagate a $schedule_B$ SYNC-R (reset request) through the $schedule_A$ cluster.

5. A node newly following $schedule_B$ converts its cluster members from $schedule_A$ by sending $schedule_B$ SYNC-R packets at a high rate for many frames (determined by the duty cycle). Nodes that hear a SYNC-R adopt $schedule_B$ provisionally and wake

to listen for additional SYNC on that schedule for verification. This similar to the SYNC-V exchange, except the nodes hearing a SYNC-R skip the step of sending a SYNC-V to trigger an increased SYNC rate (a node sending SYNC-R is already using a higher rate). As with SYNC-V verification, if the $schedule_A$ node hears 2 SYNC packets after the SYNC-R at the expected time, they adopt $schedule_B$ and engage in converting their neighbors for a short period as well (spreading the SYNC-R through the $schedule_A$ cluster).

6. Nodes that have undergone a schedule reset or heard the SYNC-H packet ignore new schedule resets for a specified cool-down period so that the separated clusters have time to merge and settle before considering additional orphaned clusters. This aids in the functioning of the algorithm in also prevents malicious nodes from triggering a sync restore attempt repeatedly (wasting energy). All nodes continue normal communications on their primary schedule while resynchronization occurs in the background (though their communications is limited to the virtual cluster they are a member of until the reset propagates).

The length of the cool-down period depends on the expected network size. We will test this in our simulator under different conditions.

4.4 Remarks

Although RSMAC was inspired by the need to resist intelligent jammers, while adding the random sleep feature we discovered a method for globally synchronizing the network and eliminating explicit neighbor discovery. This should improve energy fairness and reduce energy consumption, perhaps by enough to make up for the additional energy consumed by larger SYNC packets and algorithmic complexity. Global synchronization has additional benefits as well. For example, mobile nodes can travel throughout a network and remain loosely

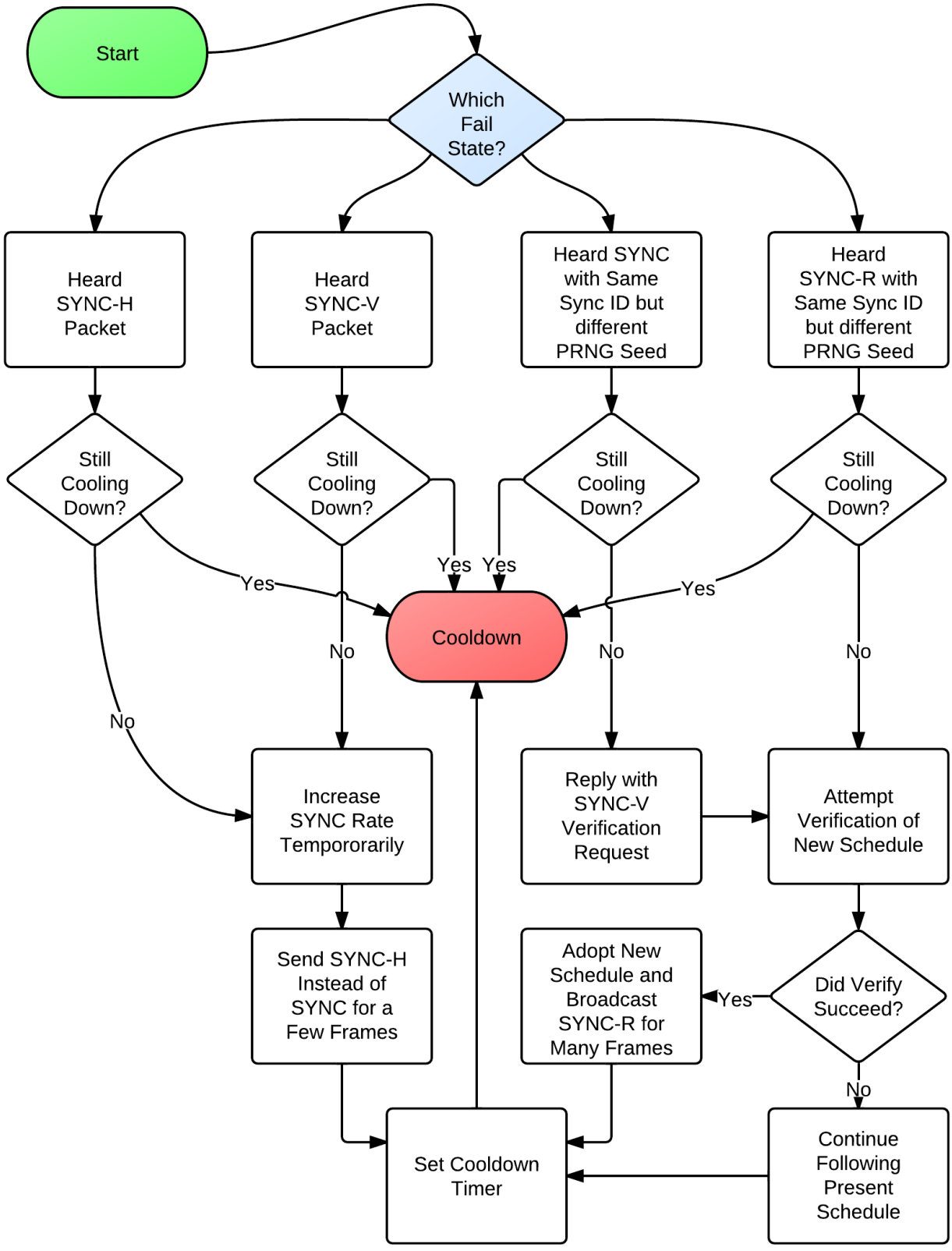

FIGURE 4.12. Global Sync Restore Algorithm: In the rare event of lost global sync, a node detecting an alternate schedule tries to verify and adopt it.

synchronized regardless of location. We will determine the basic energy and performance tradeoffs by performing preliminary testing in our novel simulator.

We are currently preparing a separate publication describing RSMAC. It will contain a summary of the background material in chapters 1 and 2, the details in this chapter, and the experimental results from chapter 5 that illuminate the functionality of the protocol.

Chapter 5
Experimentation

5.1 Experimental Setup

We begin by explaining the details of our simulation setup, including the precise way in which energy is measured, transmission interference is handled and transmission success is measured. We start with an analysis of the hardware we are simulating.

5.1.1 Radio Hardware Model

Energy use, timing characteristics, bit rate, and radio signal interaction are the most important parameters in testing a MAC protocol in software. We review energy use in detail, and then note other elements. We use published information about typical WSN motes to form the basis of our energy use model for radio and CPU [10, 11, 13, 1, 2].

Low complexity radio hardware such as the Chipcon 1000 [10] supports straightforward encoding schemes such as Manchester encoding. Manchester encoding uses two symbols per bit for redundancy and error detection. The baud rate is 38400 which makes the bit rate 19200 since overhead is 2 symbols per bit. The energy ratio for transmit, receive (listen) and sleep for the cc1000 at 915 MHz ranges from 27000:10000:1 to 10000:10000:1 (depending on the transmission power which can range from 5 dBm to -20 dBm). This energy ratio is common in WSN motes with the cc1000 or similar radio chip. We use a ratio of 20000:10000:1 range that approximates prior testing [45, 33] and is near the maximum transmission power. Interestingly, when testing is done on hardware, energy is generally estimated using such a model. The reason is that using physical energy monitors on every mote in a test network is generally not practical (motes do not come with hardware for measuring energy usage).

High complexity radio hardware such as the Chipcon 2420 [11] supports much higher data rates and complex encoding such as O-QPSK (offset - quadrature phase shift keying). The transmit, receive, sleep ratio is closer to 20000:20000:1 for this chip and encoding scheme. Note the receive power per second is twice as high per second and about equal to typical transmit power. However, the bit rate is ten times higher (250 kbps vs. 19.2 kbps) so the node can support a much lower duty cycle and ultimately conserve more energy while maintaining throughput and low latency. SMAC and RSMAC are intended for simpler hardware represented by the cc1000 that is available on the smallest WSN nodes such as Mica 2 and Mica 2 Dot [3], so we use the cc1000 specs in our current testing model.

To simulate radio interaction, we use an aggressive model in which a small amount of signal overlap destroys a packet. This is justified because intelligent jamming is about detecting and exploiting MAC timing, hence in using an aggressive interference model we're giving the jammer every advantage and focusing on confounding the timing of the jammer rather than relying on physical characteristics (such as jammer distance) to partially mitigate jamming. This also means that bad MAC layer timing (slight packet collisions due to uncorrected clock drift) will also fail with a very high rate in our simulation.

To simulate timing, we give each simulated node an independent software clock and simulate random clock drift based on physical measurements. For the MICA2, we have a maximum drift of $40\mu s$ and drift tends to be consistent over the short term according to research done by Maroti et al. Hence, we simulate MICA2 drift rates by having each node maintain a drift rate ranging from $-20\mu s$ to $+20\mu s$ plus or minus random jitter of 0 to 2 μs as measured by Maroti et al [30]. In summary, we use tx, rx, radio-sleep, cpu, cpu-sleep ratio of 20000:10000:1:8500:10, a bit rate of 19200 bits per second, and the interference and clock drift models described above.

5.1.2 SMAC and RSMAC Packet Comparison

The logical SYNC packet size for SMAC is 9 bytes. The packet fields and size in bits are: length (8), type (8), source (16), sync-node (16), timestamp (8) and CRC (16) [44]. For RSMAC, we add a 16 bit PRNG seed for a 12 byte packet. However, we must add a preamble to this to get the physical packet. The preamble on the cc1000 must be set to one of four settling states that listen for at least 23, 34, 55 or 98 chips. That means preambles must be at least 2, 5, 7 or 13 bytes long depending on transmission strength and distance. In practice, preambles are longer than the minimum for reliable transmission. We use a minimal preamble of 3 bytes for SMAC and RSMAC. A minimal preamble gives SMAC a slight advantage in overhead comparisons (a larger preamble would make the percentage increase in RSMAC packet size less significant). The final packet sizes are 12 bytes and 14 bytes after the preamble is added (13 bytes for CTRL packets, described in more detail below).

5.1.3 Packet Encryption

TinySec is a component of TinyOS that has been expanded over the years to include different encryption options [23, 22]. It defaults to Skipjack encryption which uses an 80 bit key and has very low energy overhead but relatively high memory overhead. Shepherd explains Tiny Encryption Algorithm (TEA) in detail which is optimized for 32 bit CPUs and operate on 64 bit blocks [37]. The complexity of the algorithm depends on the number of cycles in which permutation and substitution are performed. The recommended number of cycles is 32, but Shepherd shows that 16 cycles is realistically enough. Lee et al present a review of WSN packet encryption including software algorithms, hardware AES support in the cc2420, and sensor optimized encryption using Skipjack and XXTEA, a corrected form of TEA [27].

We have a difficult choice between XXTEA and Skipjack. Skipjack uses much less energy per byte, but requires 12 times the RAM and 3 times the program memory on the Mica

2. That is, TinySec's Skipjack uses 10 KB (almost 10%) of the program memory on a Mica 2 and 0.6 KB (15%) of the 4 KB of RAM. XXTEA is also more efficient when encrypting larger packets which may be useful if we encrypt data as well as headers (which we probably want to do), while Skipjack scales linearly with additional data. Although the memory requirements are high, we choose Skipjack since the energy savings vs XXTEA are between 4:1 and 2:1 (packet size ranging from 16 to 128 bytes). It takes approximately 220 μs to encode or decode an 8 byte block with Skipjack on a Mica 2 mote, however this can be done concurrently with transmission and reception. For example, for transmission, encryption can occur while the preamble is being sent with one 8 byte block taking approximately half a byte's worth of transmission time to encrypt, it is not difficult to stay ahead of the transmission speed). Finally ,we note that energy cost of hardware AES encryption on the Mica Z's cc2420 radio controller is so low as to be negligible [27]. It's about 1% of software encryption time/energy or less). Clearly hardware encryption is far more desirable when possible.

Since efficient block encryption ciphers are intended to work on 64 bit blocks, we have to address the question of packet size and what part of the packet to encrypt. SMAC SYNC packets are 9 bytes and the length field cannot be encrypted (it would be rendered useless to the physical layer), so we encrypt the last 8 bytes, including the CRC. Ideally, we would not encrypt the CRC because it might disqualify the packet before the costly decryption algorithm is run, however the cost of transmitting even 1 extra byte exceeds the cost of decrypting. For the RSMAC packet, we can encrypt from type to RNG seed leaving the CRC unencrypted. If we want DATA encryption using a 64 bit block cipher, the payload must conform to 8 byte increments. Using the energy model established, we will wait with the CPU active for 4 ticks to simulate the time taken for encryption. This will cause the appropriate amount of energy to be recorded. Encryption changes the send and receive

ratios for 64 encrypted bits in each MAC header from 20000:10000 to about 20531:10531 representing a relatively small increase in energy usage (2.5 and 5 percent respectively).

The CTRL packet fields for SMAC and RSMAC are identical at 10 bytes logical and 13 bytes physical (plus data bytes in a DATA packet). The packet fields and size in bits are: length (8), type (8), destination (16), group (8), source (16), duration (8), CRC (16) [44]. The group field designates the receiver group for broadcast packets and duration is used to specify the total transmission length (so that an RTS or CTS contains some hint of the length of the expected DATA transmission allowing temporary sleep). Since there is little sensitive information here, we may only wish to encrypt the data. If we did encrypt the header starting at the type field, the 64 bit block cipher would have two bytes left to encrypt which would encompass the CRC for non-DATA packets and the first two bytes of data otherwise. If DATA payload encryption was desired, it would have to begin at the third byte and the data length would have to be 8 byte increments plus 0, 1 or 2 bytes (precisely 2 would leave the CRC unencrypted which would optimal for long DATA packets).

5.1.4 Reducing Packet Header Size

Packet headers could be made more efficient by packing some fields to the shortest necessary length instead of following byte boundaries. This requires a small amount of CPU overhead for unpacking packed fields, but considering the cost of using the radio it should be well worth it. For SMAC packets, we cannot pack without decreasing the length below 64 bits. The RSMAC packet remains long enough to pack if we encrypt the CRC field. The type field encodes 8 types in RSMAC which fits comfortably in 4 bits. Node ID fields may be reduced in software to fit the maximum expected network size. A network size of 2^16 would be extraordinary for a WSN, however 2^8 could be limiting. We could use 11 bits to represent 2^11 nodes. The relative timestamp is transmitted as the number of slots remaining in the

senders SYNC period. Based on the packet size, packet arrival time and known slot size, the receiver can calculate the clock state of the sender. The field only needs to be large enough to represent the maximum number of slots. The original implementation of SMAC used 15 SYNC slots (a large amount needed only for very dense node configurations since SYNC traffic is periodic and based on a fixed frequency) which would require only 4 bits to represent. Since packet size must fall on byte boundaries, we can provide 6 bits which is more than enough. In conclusion, an efficient packet size for RSMAC might be 12 bytes. Since we are implementing a 64 bit block cipher, we do not test with the reduced packet size. However, we note that the Skipjack block cipher is ideal for constrained WSN hardware and RSMAC can use a reduced packet size even in this case, while SMAC would require padding the packet to RSMAC size or a cipher that supports arbitrary sized data (which are less CPU and memory efficient and less secure).

5.1.5 Jammers

We use constant jamming as a baseline [43] and cluster based jamming [26] to determine the efficacy of RSMAC in achieving its primary goal. We will use the deceptive jammer of Xu et al to test robustness of global synchronization in the face of captured, rebroadcast SYNC packets. Repeated packets may be a significant threat to an unprotected synchronization algorithm.

5.2 Demonstrating the Simulation Accuracy

First we perform tests of SMAC with our MICA2 model and compare to published results. This demonstrates the accuracy of the simulation model for message latency, throughput and energy usage. We will demonstrate the flexibility of the simulation by testing with different duty cycles, number of slots and sync rate before moving to RSMAC testing.

5.2.1 SMAC Transmission Latency

We begin with a straightforward test of message latency that duplicates SMAC latency testing in [45]. We use a ten-hop network (11 nodes) arranged in a straight line with Node ID ascending for each hop. The source node (lowest ID) generates a 100 byte data packet and sends it to the next highest Node ID via the standard RTS/CTS exchange until the data reaches the sink node (highest ID). The source sends 20 packets with 12 seconds between packets (ensuring that each message travels from source to sink prior to the next message being sent). The listen interval is 115 ms divided into sync and ctrl periods at a ratio of 1:2 (37 ms and 79 ms). The duty cycle is 10% (producing a frame size of 1.15 seconds and a sleep interval of 1037 ms). All 11 nodes are synchronized to the same schedule (a simple feat that is accomplished by powering the nodes on one at a time allowing the initial neighbor discovery period to cause each succeeding node to join the existing network).

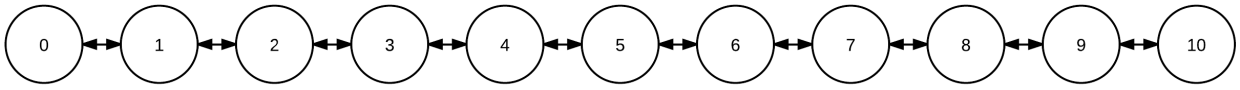

FIGURE 5.1. Ten-Hop Test Network: A simple ten-hop linear network serves to test latency and adaptive listening in SMAC. Each node can hear only its immediate neighbor. Packets are generated at node 0 and sent to node 10.

First we tested SMAC with adaptive listening disabled. This means that if a transmission begins during the listen period, it will continue until finished, even if interrupted by the start of the sleep interval. With adaptive listening enabled, any node that hears an RTS or CTS will set it's virtual carrier timer and sleep for the length of the transmission. When it awakens, even if the ordinary sleep interval has begun, it will listen on the chance that it is

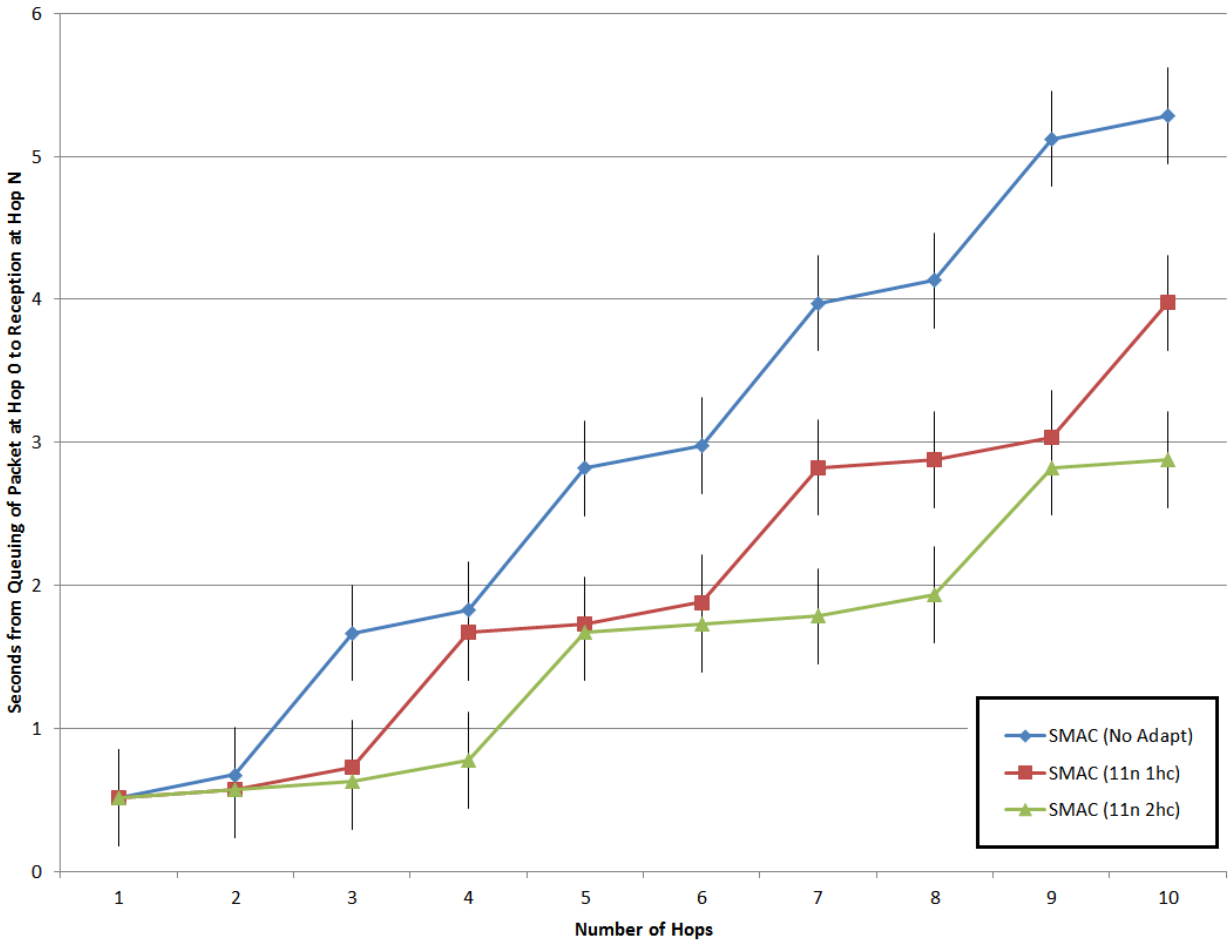

FIGURE 5.2. SMAC Latency with No Congestion): This chart shows mean cumulative latency (and the standard deviation) per hop for messages traveling across 10 hops (using testing parameters outlined at the beginning of this subsection). First we tested SMAC with the adaptive listening algorithm disabled. A pair of nodes will continue an exchange that was begun before the sleep interval was reached but go to sleep immediately after. This allows two complete exchanges per frame (whether the linear ten-hop network, or erroneous network is used). With adaptive listening enabled, the third hop hears the CTS from the second hop prior to entering sleep mode, so it wakes up and listens for the start of a new transmission even though sleep mode has already begun. This allows a packet to travel 3 hops per frame. If each node can listen and transmit 2 hops away but nevertheless insists on routing the packet through all 10 hops (instead of the more direct five-hop path) a data packet can travel 4 hops per frame. This would be an unusual situation in an actual sensor application since a routing algorithm would ordinarily use the shortest path.

the next hop in a multi-hop transmission. This serves to reduce multi-hop message latency. Under our test conditions, this allows for a packet to move 2 hops per frame.

When Ye et al performed this test with physical MICA motes, they used the lowest transmission power and placed motes at one meter intervals in a straight line. Their intention was to create the linear ten-hop network shown in figure 5.1. However, we believe the nodes were too close allowing each node to hear the next 2 hops instead of just one. Rather than use a routing algorithm, the nodes simply incremented the node ID to route a packet to the next hop (so packet routing did not take advantage of this two-hop connectivity). However, two hop connectivity does affect the adaptive listening algorithm. Ye et al did not detect this problem resulting in reported latency rates that were artificially low. We report some results for this erroneous network and mark it as *11n 2hc* on the graphs (with the correct graph being *11n 1hc*). We detected this problem immediately when our simulation only achieved 3 hops per frame and further analyzed the simulation output to discover the cause of the discrepancy. By duplicating the erroneous network configuration (figure 5.3), we duplicated the results.

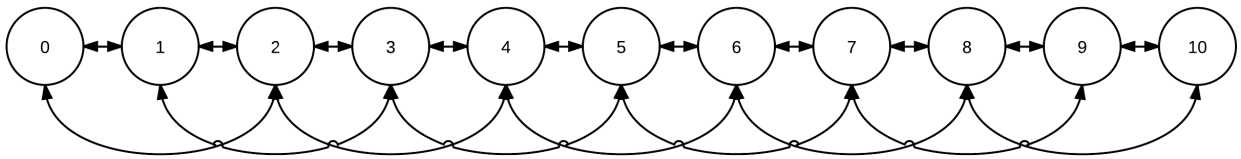

FIGURE 5.3. Erroneous Ten-Hop Test Network: Ye et al placed nodes too close together and accidentally formed a network with two-hop connectivity. The test protocol routed each packet through all 10 hops, but each two-hop node heard CTS packets they should not have heard and took advantage of adaptive listening to send 4 packets per frame instead of 3, reducing the mean cumulative message latency.

Using our simulation output logs, we can take a closer look at how the erroneous two-hop connectivity caused Ye et al to underreport SMAC latency (and report incorrect results in other tests as well). In figure 5.4 we show the timing of each CTRL and DATA packet, the start of the sleep interval and the beginning and end of each node's virtual carrier sense period (which is followed by a brief Adaptive Listen period). Node 4 never hears any CTRL packets during the normal CTRL period which is a prerequisite for engaging in virtual carrier sense.

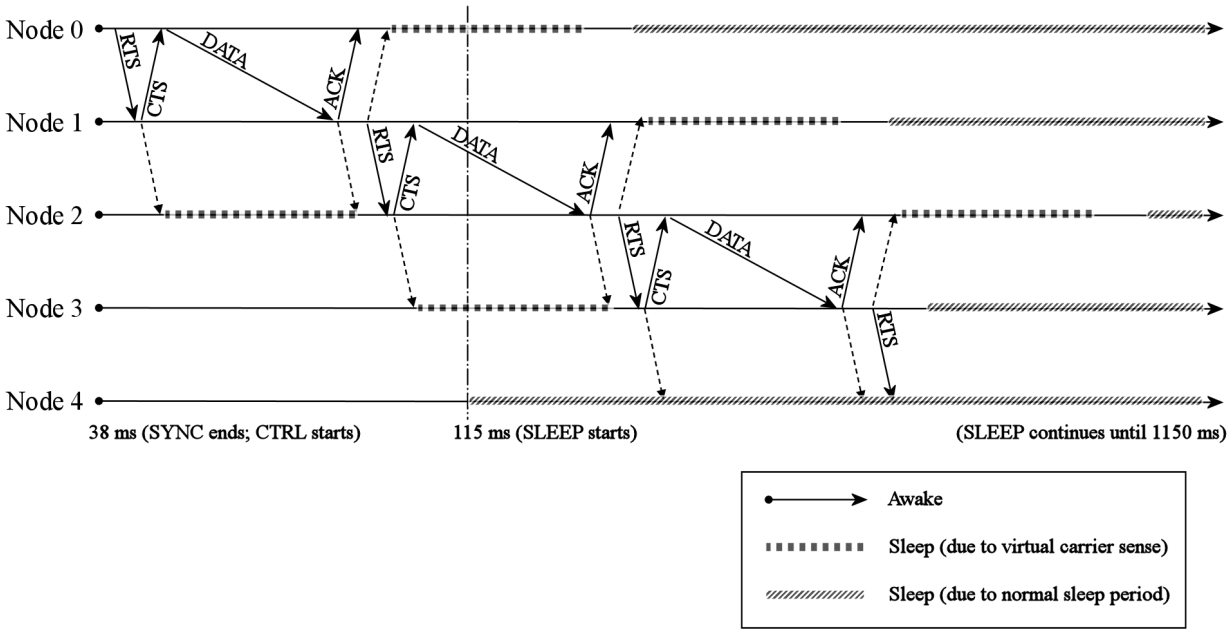

FIGURE 5.4. SMAC Three-Hop Transmission: Nodes 0 to 2 send an RTS and complete a transmission, but node 3 cannot transmit to node 4 because it is not awake to hear the RTS. It does not hear any CTRL packets, never enters a virtual carrier sleep, and cannot wake for adaptive listening.

Ye et al tested SMAC throughput (bytes per second) on the same 10-hop network. We will test throughput on the corrected linear 10-hop network and also the erroneous network. After the throughput tests, we only report results for the originally intended 10-hop network shown in figure 5.1 (in addition to other configurations as noted).

5.2.2 SMAC Throughput

We test throughput using the settings described at the beginning of the previous subsection, however the packet rate is raised to the maximum so that all 20 packets are queued at the same time and nodes will have to contend with each other to transmit packets. This test matches high traffic testing done by Ye et al for SMAC.

First we look at the latency again since we expect it to change under high traffic conditions. Figure 5.5 shows the latency for high throughput tests. Latency goes up significantly since there is a very large queuing delay as 20 packets sit idle in the queue of node 0 waiting for the network to clear up. The packets latency is measured from the moment the packets hit the node 0 queue, so the increase in average latency will be considerable. We also look at the throughput in bytes per second which is a relevant measurement only when straining the channel capacity. Figure 5.6 is generated from the same simulation results as figure 5.5 and the results are related to one another in a similar way (we describe this in more detail in the figure captions).

Interestingly, our simulated results for the corrected linear network match the results reported by Ye et al suggesting that for these tests, their network was connected correctly (a ten-hop linear network without the two-hop connectivity error). It may be that their physical environment was not well controlled causing the expected linear connectivity during some tests and the incorrect two-hop connectivity during other tests. This highlights a major advantage of working in simulation. We can control the simulation environment with reliability and precision. We can also test a variety of configurations with ease as we will show in the following sections. For WSN testing with physical motes, it is common for researchers to use a very limited number of test networks and configurations, likely due to the difficulty and time consumed by physical network setup.

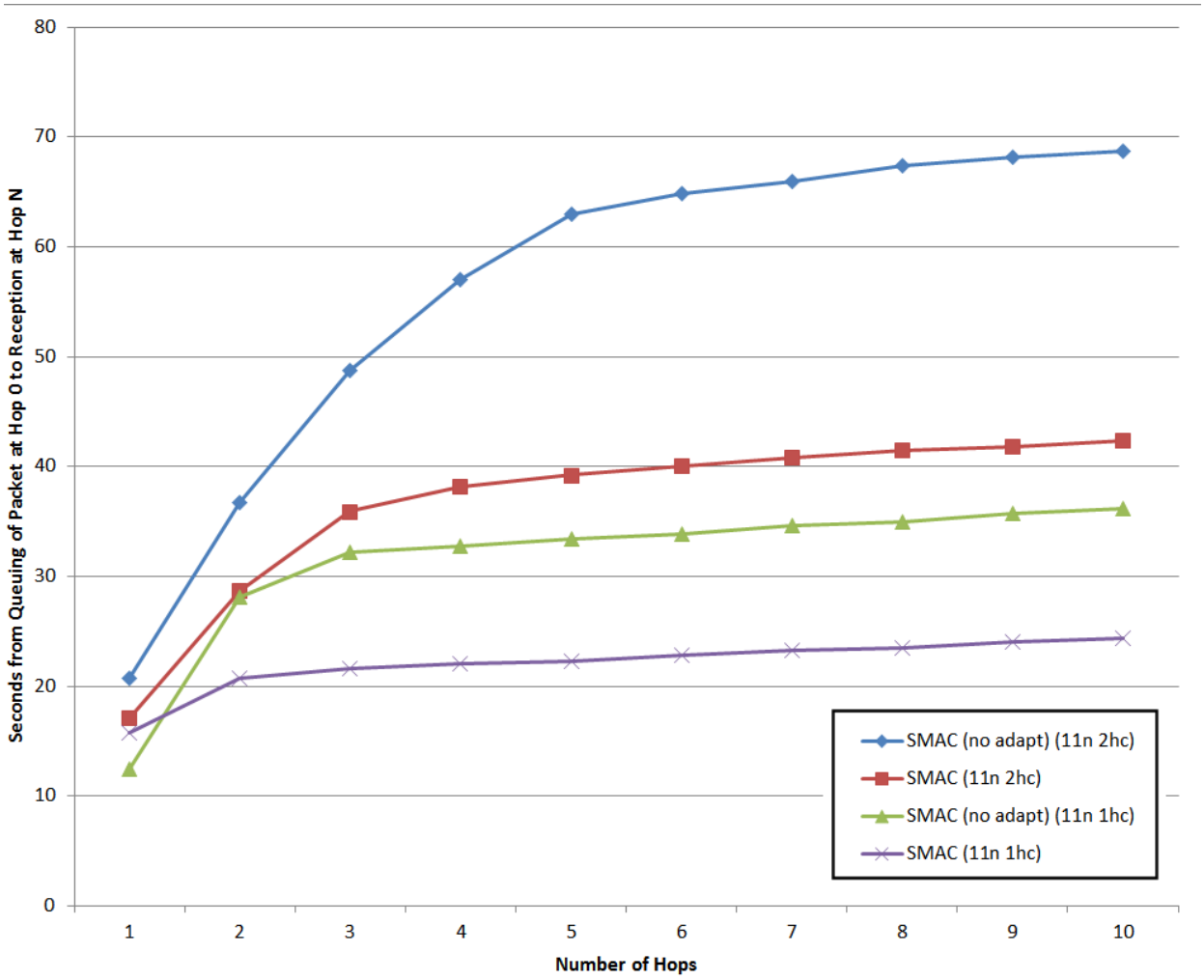

FIGURE 5.5. SMAC Latency with High Congestion): With high congestion, SMAC performs best in the corrected network with linear connectivity. The tests using the erroneous two-hop network both of higher latency than SMAC even without adaptive listening. This is not surprising because the extra connectivity makes it more difficult to win the medium since each node is contending with 4 neighbors instead of 2. In the network with two-hop connectivity, a proper routing algorithm would skip every other node resulting in better performance since each hop would only be contending with two transmitting neighbors. After a few hops, packets escape the congested area of the network (node 0 with its backlog of 20 packets) and do not accumulate much additional latency per hop.

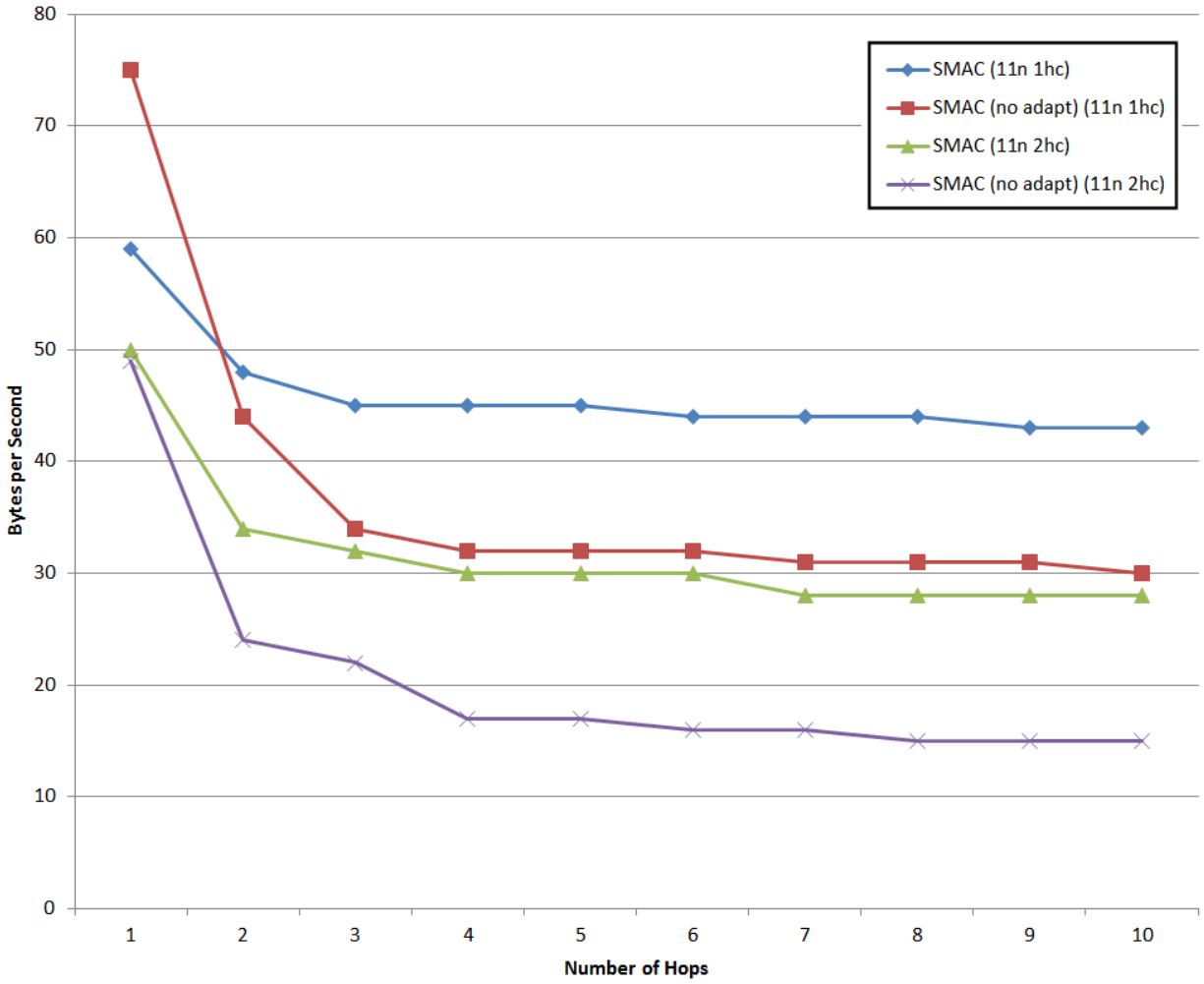

FIGURE 5.6. SMAC Throughput Test: Latency closely correlates with throughput during times of high congestion, so we again expect to see SMAC with and without adaptive listening perform better in the ten-hop network with linear connectivity than in the ten-hop network with erroneous two-hop connectivity. The reasons are the same. Routing linearly through all ten hops while being able to hear two hops away results in a more difficult time winning contention since each node competes with 4 neighbors instead of 2. We see SMAC without adaptive listening achieving slightly better performance at the first hop again. This is because the lower throughput of SMAC without adaptive listening takes a little extra time to bring the network to full congestion. As with latency, packets escape the congested area around node 0 after a few hops and throughput consequently levels off.

5.3 SMAC Optimizations and Energy Consumption

Ye et al did not perform additional testing for energy consumption of SMAC. Instead, they simply used the 10% duty cycle of their tests of SMAC with and without adaptive listening and compared with SMAC at 100% duty cycle (roughly equivalent to IEEE 802.11). They determined that adaptive listening uses an insignificant amount of additional overhead and provides performance comparable to the 100% duty cycle under high loads and energy saving comparable to the 10% duty cycle under low traffic loads. We agree with this finding in the case of a 10% duty cycle.

However, Ye et al did not report any testing on the energy cost and trade-offs of the synchronization algorithm which is roughly a third of the energy use at their 1:2 ratio. The 1:2 ratio also does not consider the extra energy used by neighbor discovery which Ye et al define as a 10 second period every 2 minutes. Neighbor discovery at this high a rate would change the SYNC to CTRL listen ratio to as bad as 2:1 if the network has very light traffic. This makes synchronization by far the greater energy consumer in SMAC.

Ye et al also did not comment on asymmetric energy use of nodes that maintain two or more schedules. With a SYNC period of 37 ms as used in their tests, 6 or 7 SYNC packets could be transmitted per frame! With a SYNC frequency of 1 per 13 seconds, this SYNC period is considerably larger than necessary. We will perform new tests with a SYNC period of 16.4 ms (about 43% the original size), which is still large enough to accommodate higher connectivity than a ten-hop linear network. For real applications, the SYNC period could be adjusted to fit the density of the network.

Ye et al also did not test the potential advantage of using a very short CTRL period which would allow an RTS-CTS exchange to begin immediately while all other nodes go to sleep. Adaptive listening would still allow two complete transmissions per frame (66% per

frame throughput) using this configuration while having the CTRL period reduced to about 28% of 78 ms (21.6 ms).

A reduced listen interval of 38.1 ms (2:3 SYNC:CTRL ratio) gives us a duty cycle less than 4% if we maintain the same frame length of 1.15 seconds. If we increase the duty cycle while maintaining the listen interval, the sleep interval and frame size will shrink. A duty cycle of 5% gives us a frame size of 762 ms which will bring our complete RTS-CTS transmissions per 1.15 seconds to an average of 3. This is half the duty cycle that Ye et al tested with for performance that will theoretically be nearly identical. We will determine the performance with latency testing for this configuration, first using low traffic (12 second separation between each enqueue) then high traffic (20 packets queued immediately). The purpose of these optimizations is to reduce energy usage without significantly reducing multi-hop latency or throughput. We will measure SMAC energy consumption in the subsection following this one.

5.3.1 Latency and Throughput of SMAC with Optimizations

For optimized SMAC latency testing, we use the linear ten-hop network exclusively. As before, the source node (lowest ID) generates a 100 byte data packet and sends it to the next highest Node ID via the standard RTS/CTS exchange until the data reaches the sink node (highest ID). The source sends 20 packets with 12 seconds between queuing each packet for sparse traffic, and queues all packets immediately to simulate dense traffic. The listen interval is 38.1 ms divided into sync and ctrl periods at a ratio of 1:1.44 (15.6 ms and 22.5 ms). The duty cycle is 5% (producing a frame size of 762 ms and a sleep interval of 724 ms). All 11 nodes are synchronized to the same schedule again. The results for SMAC without optimization are copied from the previous data set and to it we add the results for the optimized listen interval in figures 5.7 and **??**.

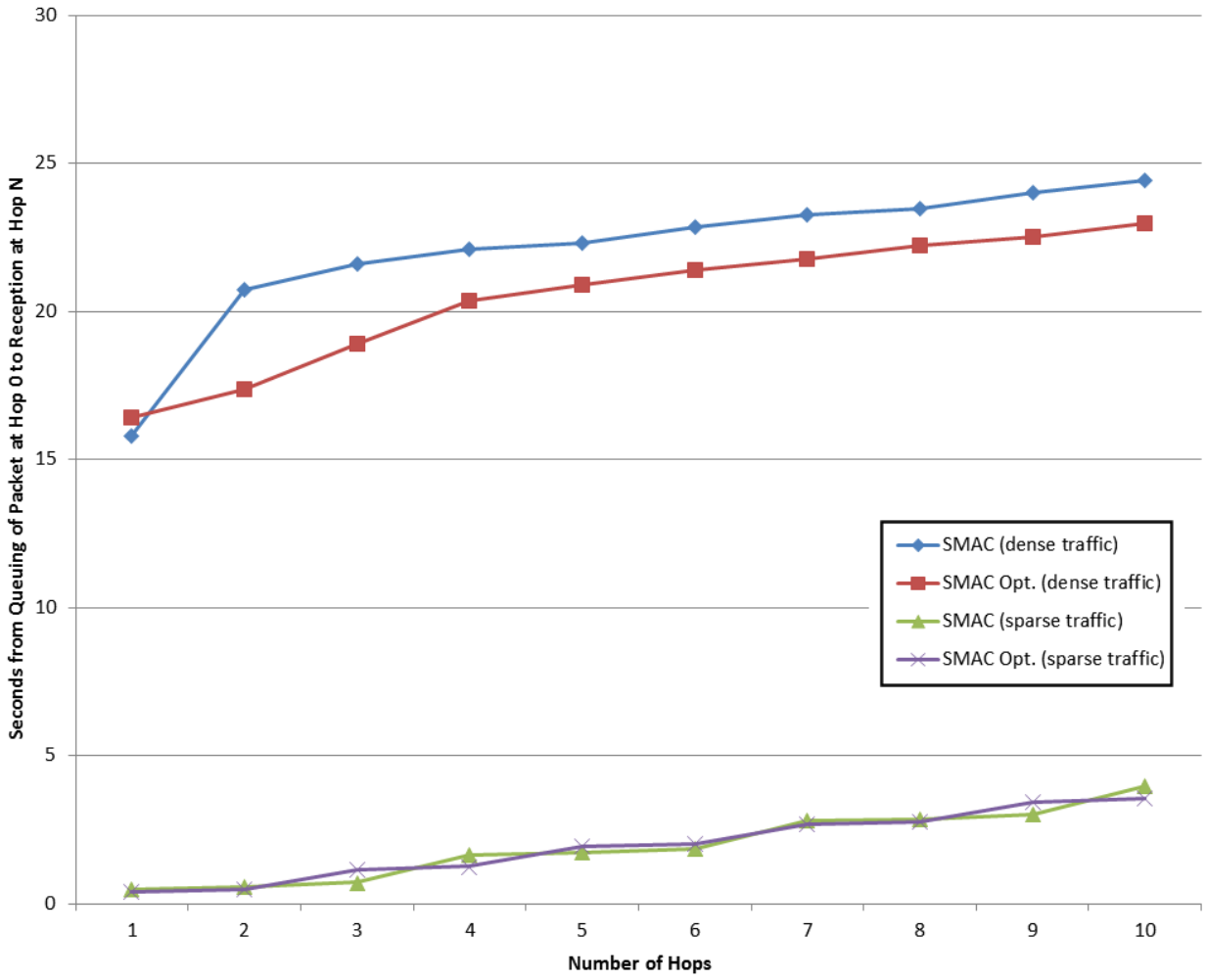

FIGURE 5.7. SMAC Optimized Latency: With the results for dense and sparse traffic on the same graph, the difference in mean packet latency when queuing packets far in excess of available bandwidth becomes immediately apparent. With sparse traffic, the optimized SMAC performs nearly identically to the original version (but with half the duty cycle and hence much lower energy consumption). Packets only travel two hops per frame, but the frame size is shorter allowing for similar multi-hop latency. In the case of dense traffic, we see slight improvements in latency for the optimized settings because more packets are sent via adaptive transmission and listening (which avoids contention with other nodes). As in the prior tests, the additional latency per hop decreases as the packets get away form the congested area of the network.

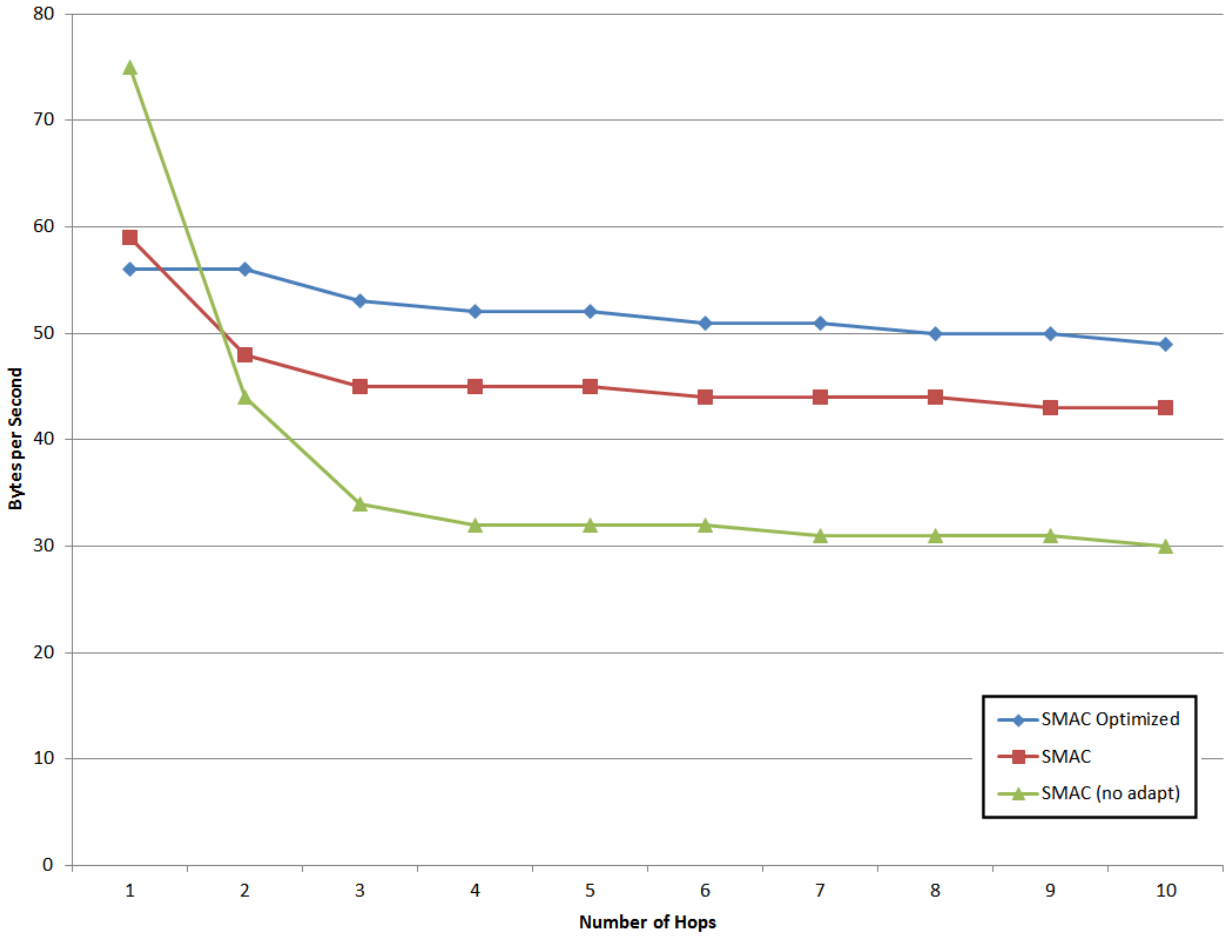

FIGURE 5.8. SMAC Optimized Throughput: We add the results for SMAC without adaptive listening to the chart to illustrate the affect of transmitting via standard contention vs. transmitting via the adaptive listening algorithm. With adaptive listening, the last node to transmit deliberately remains silent. Every other node that could interfere with the adaptive transmission has either heard the CTS and also remains silent, or has entered the regularly scheduled sleep period. With the original SMAC settings, the CTRL period is longer so a larger percentage of the transmissions occur via normal contention. This leads to more collisions and rebroadcasting wasting energy, reducing bandwidth and increasing latency. We note the first hop has high variance which is smoothed out as the hops accumulate. As before, bandwidth naturally worsens with each hop since a delay of any kind decreases the measured bandwidth. The decline levels off as the packets escape the congested part of the network around node 0 (congestion decreasing the packet flow via the collisions and backoff).

The results turn out better than expected. By taking advantage of adaptive listening to decrease the percentage of time spent in contention, we reduce the duty cycle which leads to energy savings and we get a slight reduction in latency when congestion is high. The improved latency and bandwidth during high congestion is a result of a higher percentage of packets being sent via adaptive listening as opposed to standard contention. Standard contention leads to collisions and backoff periods which increase latency, decrease bandwidth and increase energy consumption due to collision.

5.3.2 SMAC Energy Consumption

For energy consumption testing, we use a 3 node network where each node can hear both of its neighbors. We use the energy model for the MICA 2 described in the beginning of this chapter. We vary the data rate from zero packets, to one packet every 8, 4 and 2 seconds. The packets are routed through all three nodes and the energy usage is averaged. We report the usage in "days" based on the energy provided by a pair of heavy duty AA batteries (30.8 Kj). We run the test for 12 minutes, skip the first 2 minutes (since the booting period is not representative of normal energy use) and record 10 minutes of data and estimate the time needed to drain the batteries. A number of factors could raise or reduce the energy consumption in a real world application, but the ratio between the different protocols should remain similar and is the most important thing revealed by the tests.

We test SMAC in its original form (10% duty cycle, 1:2 SYNC:CTRL ratio) and our optimized form of SMAC with 5% duty cycle (yet similar performance). Both configurations are tested with a neighbor discovery time of 13 seconds (matching the sync rate of 13 seconds) and a period of 2 minutes. Ye et al reported using a neighbor discovery period of 2 minutes for any node with one or more neighbors (nodes without neighbors enter discovery even more

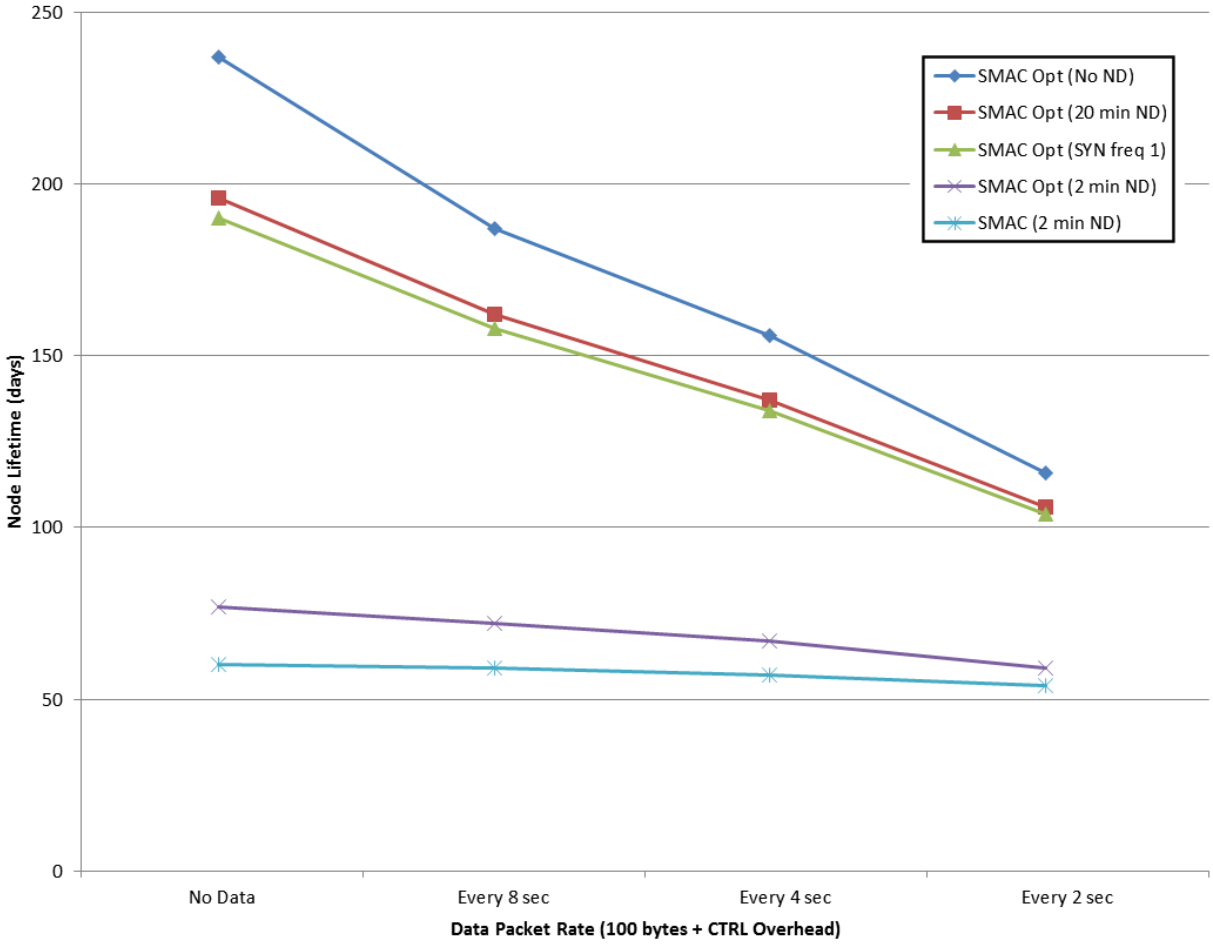

FIGURE 5.9. SMAC Energy Use: We test SMAC's energy use revealing that most of the energy goes to listening for SYNC packets (during the SYNC period or the frequent neighbor discovery period). We test our optimized protocol with the original neighbor discovery settings, then we try it with the SYNC frequency raised to 1. Since neighbor discovery is set equal to the period between SYNC packets, sending 1 per frame allows us to save a considerable amount of energy despite sending SYNC packets 17 times as often. The energy savings are comparable to retaining 1 SYNC per 17 frames and performing neighbor discovery once every 20 minutes instead of once every 2. We can maintain these energy savings with different configurations by increasing SYNC rate neighbor discovery period proportionally (to 2 frames and 4 minutes for example). The line for SMAC Opt with no neighbor discoveries shows the ideal situation in which we could somehow get by without neighbor discovery at all.

often until they find a neighbor). We also perform two tests with modified neighbor discovery to save energy and test optimized SMAC with no neighbor discovery as a baseline.

We show the results in figure 5.9. The first thing to take note of is the enormous amount of energy spent on periodic neighbor discovery in the original SMAC design. It's such large amount that the difference between sending no data at all and a large amount (100 byte packets every 2 seconds routed through every node) is only 10% less. For our optimized SMAC, the difference between the two extremes is 23%. The simplest way to eliminate this excessive energy use is to reduce the neighbor discovery frequency. We tested in our optimized algorithm with neighbor discovery every 20 minutes instead of 2 and produced a much more palatable node lifetime that responds more naturally to increased traffic. Another way to decrease the energy spent in neighbor discovery is to increase the SYNC frequency. This is because reliable neighbor discovery must be as long as the time between SYNC packets (to ensure that every neighbor is discovered). We take the SYNC frequency to the extreme of one SYNC packet per frame (from every node) and achieve energy savings nearly as good as reducing the neighbor discovery frequency to 10% (despite packet transmission costing twice the energy of reception in our energy model). We may be tempted to leave the SYNC frequency high, but this may cause more collisions at virtual cluster borders. So a balance must be struck between the SYNC rate and frequency of neighbor discovery that allows the network to reconfigure within a reasonable time without excessive SYNC traffic.

5.4 Testing RSMAC

We begin by testing RSMAC using similar tests to SMAC (and our optimized SMAC) for latency and bandwidth. Then we test RSMAC global synchronization algorithm and global sync recovery demonstrating the time it takes for global sync or recovery to propagate under different network conditions. We follow this with a look at RSMAC energy consumption com-

pared to SMAC and optimized SMAC. Finally, we demonstrate the effectiveness of RSMAC in resisting cluster-based jamming, and deceptive jammers before presenting conclusions.

Our configuration for RSMAC mirrors the optimized SMAC. However, RSMAC SYNC packets are slightly larger so the listen interval is increased and the duty cycle is increased from 5% to 5.33% to maintain the same *average* frame size. The new listen interval is 40.6 ms instead of 38.1 ms with a ratio of 1:1.241 for SYNC (18.1 ms) and CTRL (22.5 ms) periods. In short, the increased SYNC period decreases the average sleep period (using more energy), but should have no effect on latency or throughput.

5.4.1 RSMAC: Latency and Bandwidth

In figure 5.10 we show the mean cumulative latency and standard deviation for packets sent one at a time across ten-hops. The mean for RSMAC is a little higher than optimized SMAC because RSMAC transmissions are sometimes limited to one or none per frame if the random frame size is very short. Nodes do not schedule adaptive transmissions that cross the frame boundary in order to avoid interfering with the SYNC period. It may be possible to mitigate this increase in latency by allowing a partial exchange (RTS/CTS before SYNC; DATA-ACK afterward), but we have not implemented this improvement as of this writing. The standard deviation for SMAC is constant, but for RSMAC it increases after each random frame (at a slowly increasing rate).

In figure 5.11 we show the mean cumulative latency during very high congestion. We get results very similar to optimized SMAC again, but with latency slightly increased for the reasons described in the previous paragraph (transmissions must stop at the end of frame). The standard deviation for all three protocols is about the same (only standard deviation for SMAC is shown). The reason standard deviation is not excessive in RSMAC is that we have a traffic flow instead of individual packets. Each packet flows across the same randomly

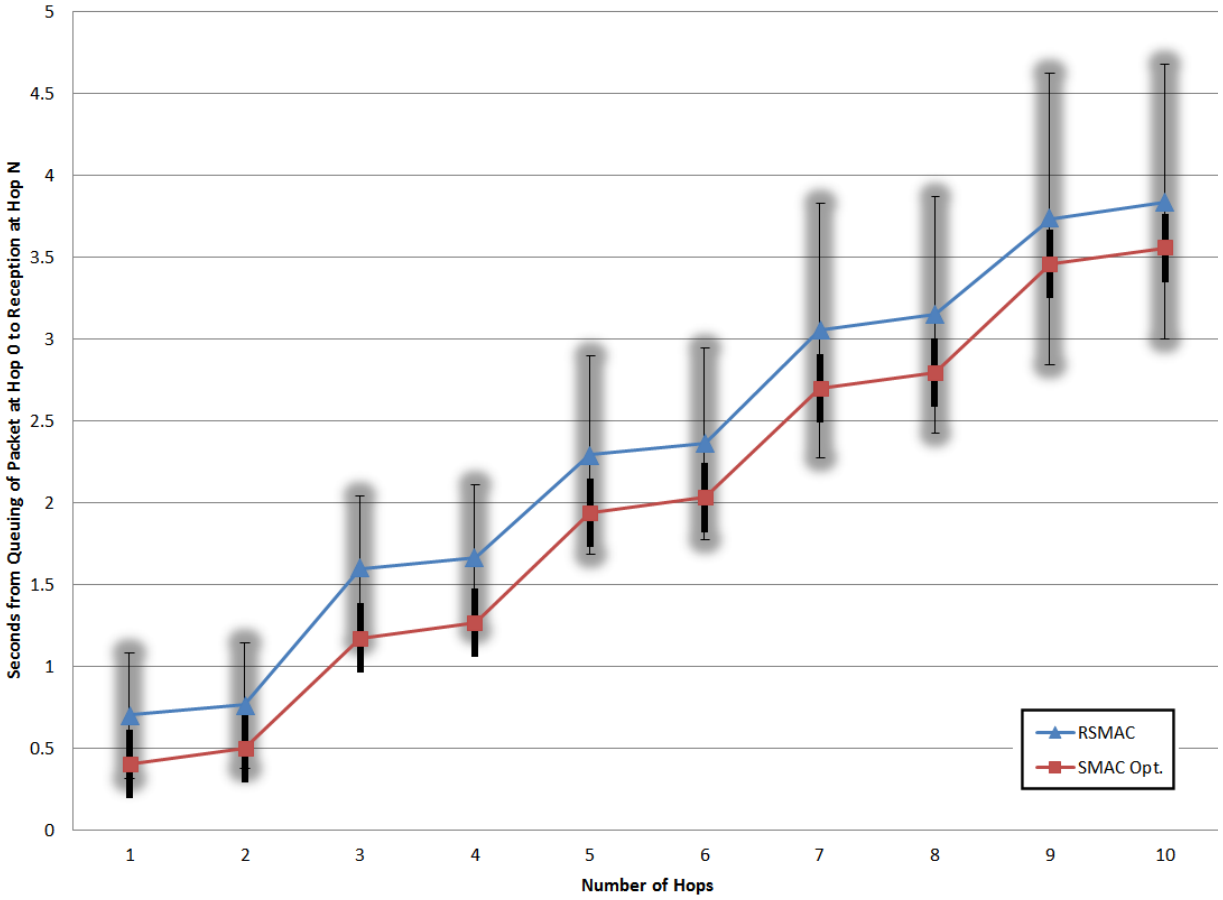

FIGURE 5.10. RSMAC Latency with No Congestion: We show mean cumulative latency 20 instances of one packet transmitted across ten-hops at a time. SMAC has a roughly constant standard deviation which is due to queuing delay on the first hop (the application may queue the packet during the sleep or SYNC interval requiring a wait), while the standard deviation for RSMAC increases with each hop due to randomness in the frame length. However, the increase is less in proportion to the cumulative mean. In other words, for one hop, there's a good chance to get the highest or lowest possible latency, but with each additional hop it becomes less and less likely.

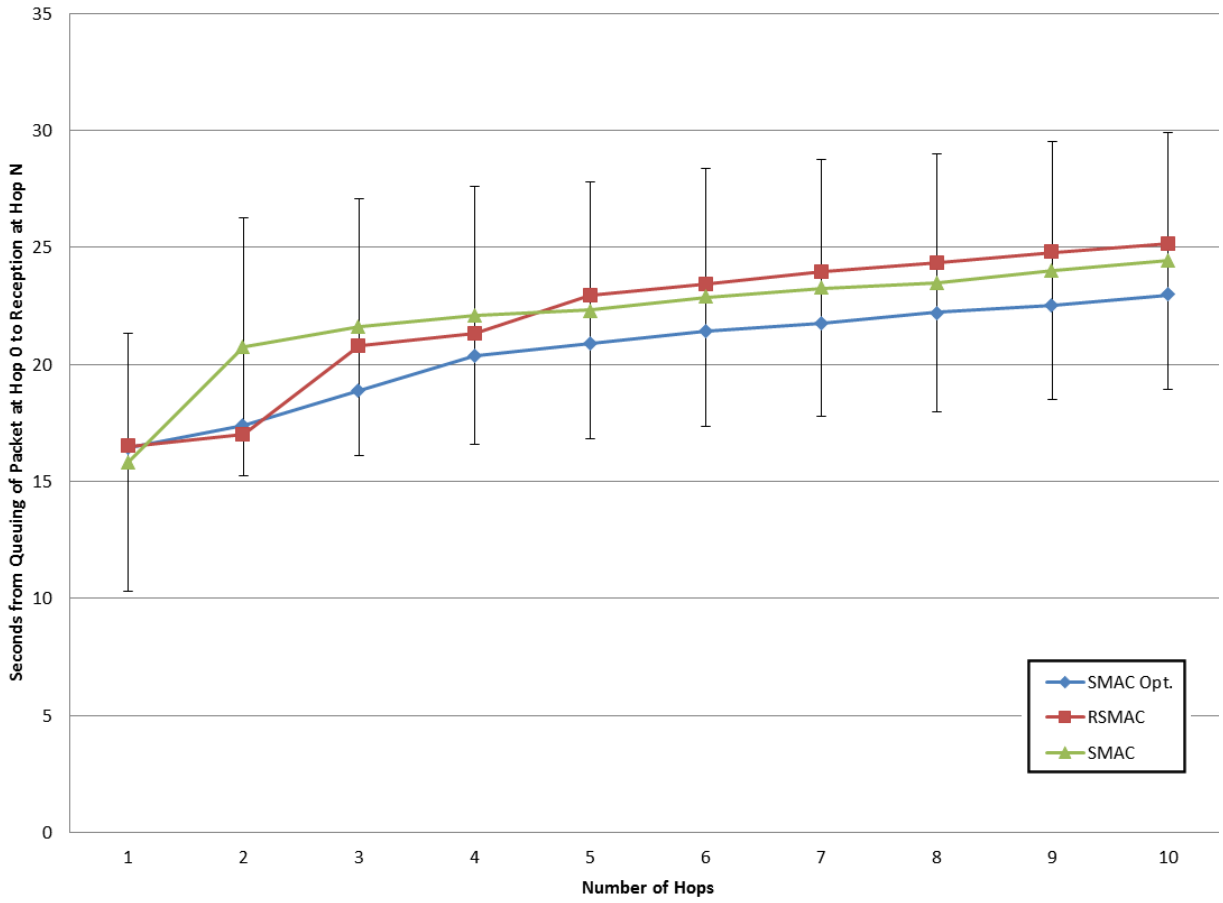

FIGURE 5.11. RSMAC Latency with High Congestion: Cumulative latency for a stream of 20 packets causes the randomness of RSMAC to be smoothed out. RSMAC starts off well, but performance eventually parallels SMAC. The reason for the good start is the variation in the packet rate at the source. SMAC consistently sends 2 or 3 packets in the first frame which allows for good first hop latency but results in immediate contention surrounding the first hop. RSMAC packets are sometimes forced to wait due to an unusually short frame length, this spreads contention out over more hops allowing for better initial performance but the same long term slope.

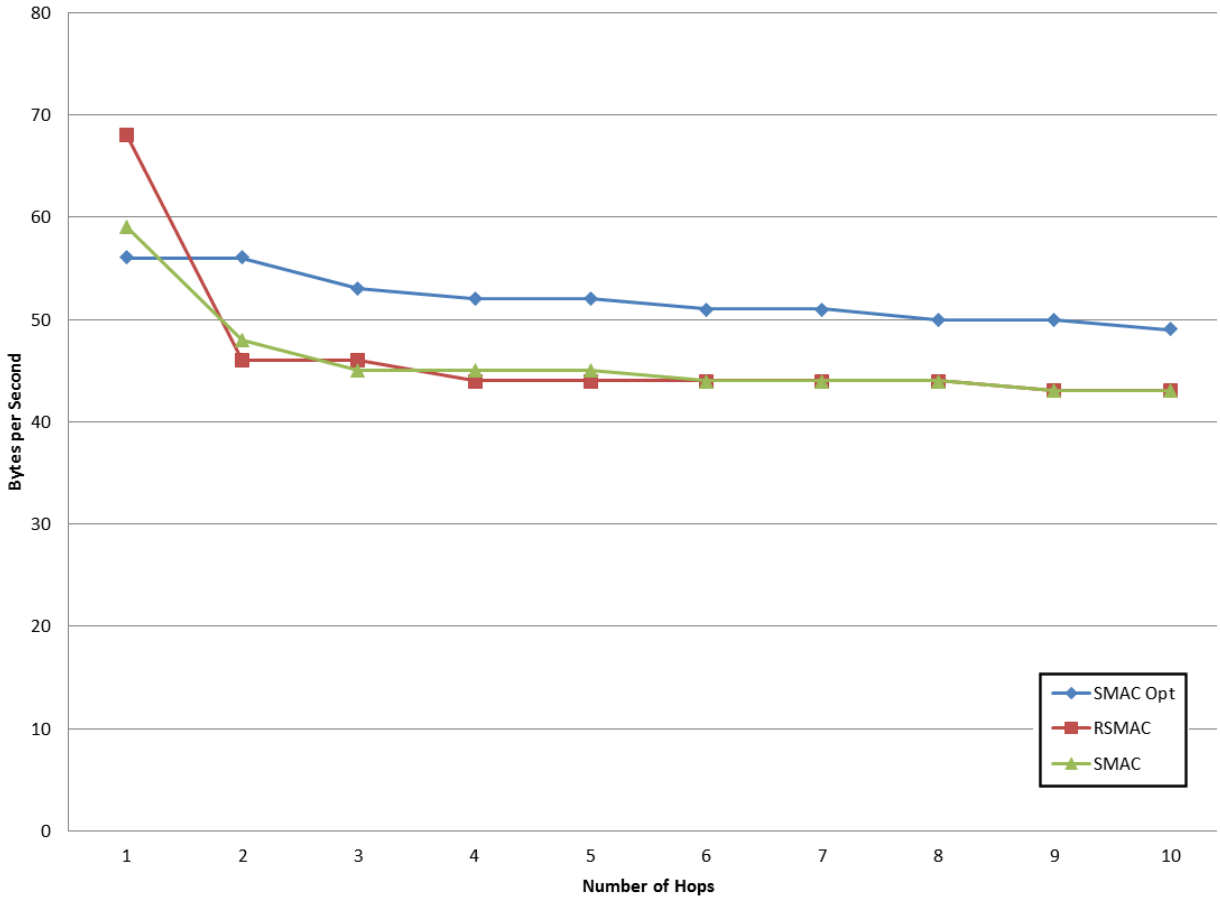

FIGURE 5.12. RSMAC Throughput: RSMAC throughput mirrors SMAC because each wastes some time for different reasons compared to optimized SMAC. SMAC spends more time in contention which causes collisions and backoff in highly congested networks, while optimized SMAC and RSMAC have very short control periods and do not waste time in contention. However, RSMAC instead wastes time waiting for short frames to pass before scheduling a transmission. A transmission will never be scheduled in any version of SMAC if the total time exceeds the frame boundary, but the sleep time is generally long enough in SMAC for this problem never to arise. In RSMAC, the random sleep interval is sometimes short enough that only one or even no packets can be sent until the next frame. Our testing is done with 100 byte packets, but RSMAC would perform more closely to optimized SMAC if data packets were shorter (TINY OS data packets max out at 30 bytes for example).

sized frames as its neighbors except for one or two frames at the end which are different. With 20 packets averaged over ten hops, the difference is not statistically significant.

Lastly, in figure 5.12 we show bandwidth for RSMAC. Just as latency for RSMAC is worse in both cases bandwidth also suffers compared to optimized SMAC. The reason is the same. The random frames are sometimes short enough to preclude adaptive transmission and even ordinary contention. Recall that SMAC's bandwidth is lower because a greater percentage of its packets are sent during the contention period (instead of via adaptive listen). Sending during contention while congestion is high results in collisions and wasted efforts.

In summary, the mean latency and bandwidth in RSMAC always parallel that of SMAC, but the standard deviation of latency for sparse packets increase with each hop. For traffic flows, the differences are averaged out across several hops hiding the randomness.

5.4.2 RSMAC: Global Synchronization Time

We test global synchronization for RSMAC using four different network configurations and measure the time it takes randomly booted nodes to synchronize globally. The variation in boot time is one minute. Nodes that start in isolation will adopt their own schedules and nodes that start up with a neighbor already active will usually adopt a neighbors schedule immediately. The schedules will randomly sync up momentarily and if the node with the higher Sync ID sends a SYNC packet, the target node adopts the schedule. Eventually all nodes must adopt schedule 100 (the lowest node ID in the network). We record the time of the last node adopting schedule 100 as the time needed for global synchronization in the test network.

A higher frequency of SYNC packets will result in faster global synchronization with a higher energy cost. We must choose a SYNC rate that provides reasonably fast config-

uration and also detects failed global synchronization within an acceptable period of time (covered in the next section. We note that different SYNC rates should not alter the DATA transmission rates significantly, but will have some effect on energy usage. We use a SYNC frequency of about 1.5 seconds (1 SYNC every 2 frames) to improve the time performance of random neighbor discovery and sync restore. Based on our testing of optimized SMAC, we are confident that the benefit of quicker global sync and recovery outweigh the small increase in energy cost. We will also test with the maximum SYNC rate (1 per frame) on the most dense network to test the effects of overloading the SYNC period.

Each test was done ten times on the networks below and the number of frames until global sync is shown in figure 5.15. Note: we refer to the node with lowest ID as the root.

- We use 11 nodes in a linear configuration with node 100 on one end. Precisely ten hops are needed for schedule 100 to reach the furthest node.
- We use 25 nodes in a 5x5 grid placed close enough for 4 way connectivity with node 100 in one corner. At least eight hops are needed for schedule 100 to reach the furthest corner.
- We use the same network above but place node 100 in the center. At least four hops are needed for schedule 100 to reach each corner.
- We use 25 nodes in a 5xt grid placed closer together to provide 8 way connectivity with node 100 in one corner. Each corner can be reached in four hops (including the far corner).
- We use the network above but increase the SYNC rate to 1 per frame. This will test overloading the SYNC period (3 slots vs. 8 neighbors transmitting). Once the network is fully synchronized, this would be counter-productive due to excessive collisions, but prior to full synchronization it may help achieve global sync faster.

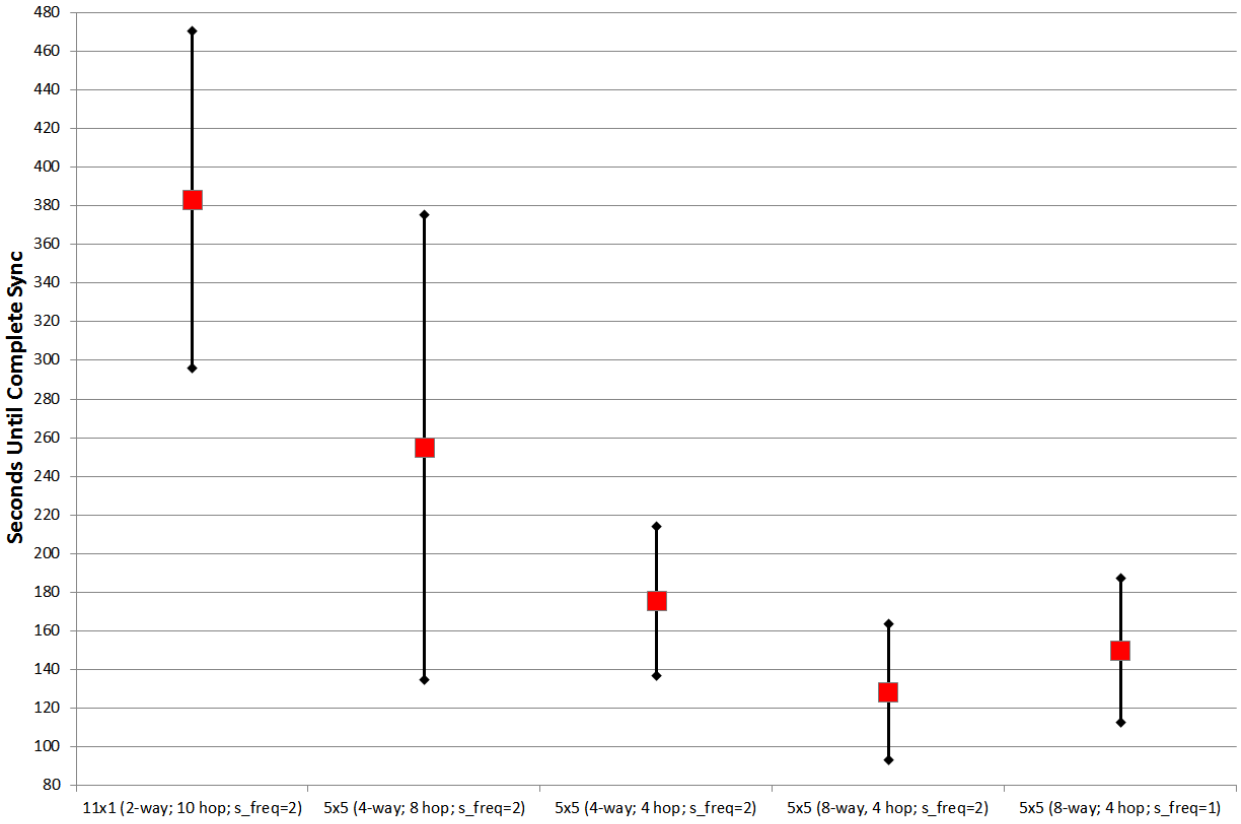

FIGURE 5.13. RSMAC Global SYNC Test: We test global synchronization on five networks and report the number of 762 ms frames required to SYNC. Each test is designed to have a particular length from the node with lowest ID (root) to the furthest hop. In the first test (leftmost column) global sync across 10 linear hops takes the most time. In the second test mean time to synchronize a 5x5 4-way connected network (8 hops max) is significantly lower because of the higher connectivity for interior nodes. This allows the broadcast sync packets to propagate the schedule more quickly. In the third test we use the same 5x5 network but move the root node from the corner to the center resulting in much better performance (hops reduced from 8 to 4). For the last two tests we use 5x5 networks with 8-way connectivity and test with our preferred SYNC frequency of one per 2 frames and then a SYNC every frame. We note that the time worsens in the last test because the SYNC period isn't large enough to handle connectivity that high (8 neighbors). Depending on the expected network density, a network should be configured with different SYNC rates to reduce collisions.

We expect the total synchronization time to decrease with the maximum distance from the node with the lowest ID (we refer to this as "hops" in figure 5.15. This is what we see. The mean global sync time correlates roughly with the maximum distance from the root and networks benefit from higher connectivity until the incoming SYNC packets become too much for the length of the SYNC period. In the last test (right-most column), sync proceeds more quickly at first but as more and more nodes begin to sync and share the same schedule, SYNC congestion increases making the overall performance worse (via collisions destroying many SYNC packets). The linear network would have benefited from the higher SYNC rate due to its low connectivity, but the 5x5 with 8-way connectivity was maximized out at a rate of 1 every 2 frames. Depending on the connectivity, the SYNC rate should be adjusted. Nodes can dynamically adjust the SYNC rate based on the number of neighbors they have as well (since we know SYNC frequency has little effect on total energy consumption).

5.4.3 RSMAC: Global Sync Recovery Time

For this test, nodes all boot at the same time and sync quickly. The networks are artificially partitioned so that each half synchronizes and settles separately with different schedules (yet both with Sync ID 100). After global sync and settling, we place a node in the center that joins the networks triggering the global synchronization restore algorithm once two SYNC periods randomly line up. The probability of the SYNC period lining up depends on the size of the various intervals and the sync rate as described in chapter 4. We record the time from placing the trigger node to completely restored global synchronization. So the recorded time includes the time taken for random neighbor discovery to occur and the time taken for the algorithm to run.

We use two test networks. First the 11x1 linear network (5 nodes per half plus the center) which tests global sync restore across 5 linear hops. Next, we perform a similar test

but increase connectivity around the center node from 2 neighbors to 4 neighbors (2 per half). The distance to the most distant node is still 5 hops. In summary, hops are the same and connectivity is better, but there are more nodes to sync. This network is shown in figure 5.14.

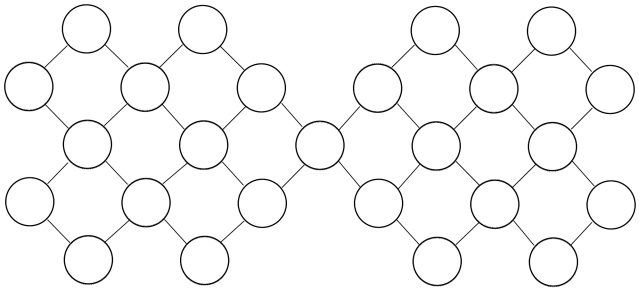

FIGURE 5.14. RSMAC Global Restore Test Network: The node in the very center waits 5 minutes before starting to ensure each virtual cluster has the global schedule with ID 100. Then the center node boots and we measure the time until global sync is restored.

In testing we can see that global sync restore is initiated very quickly and the entire process requires an amount of time similar to initial global synchronization. The restoration algorithm is a bit slower because it requires multiple packets for verification, but this is made up for by newly reset nodes increasing their SYNC rate if it is not already maximized. In the linear network test, synchronization happens a little faster than in the test with higher connectivity. The reason is that there are fewer nodes in total and less SYNC congestion in the linear network. Since the restoration algorithm increases the SYNC rate to the maximum, the linear network benefits more from this increase than the 4-way connected network which already has a lot of SYNC traffic due to the larger number of neighbors. We did not separately test the amount of time it takes for random neighbor discovery to succeed, but it must be a subset of the total time measured here and a glance at test logs suggests a range of 1 to 2 minutes. This demonstrates that, provided the SYNC rate is set to 2, we can equal the

two minute neighbor discovery period of SMAC using RSMAC's random neighbor discovery with far less energy expended. We test energy consumption in the following section.

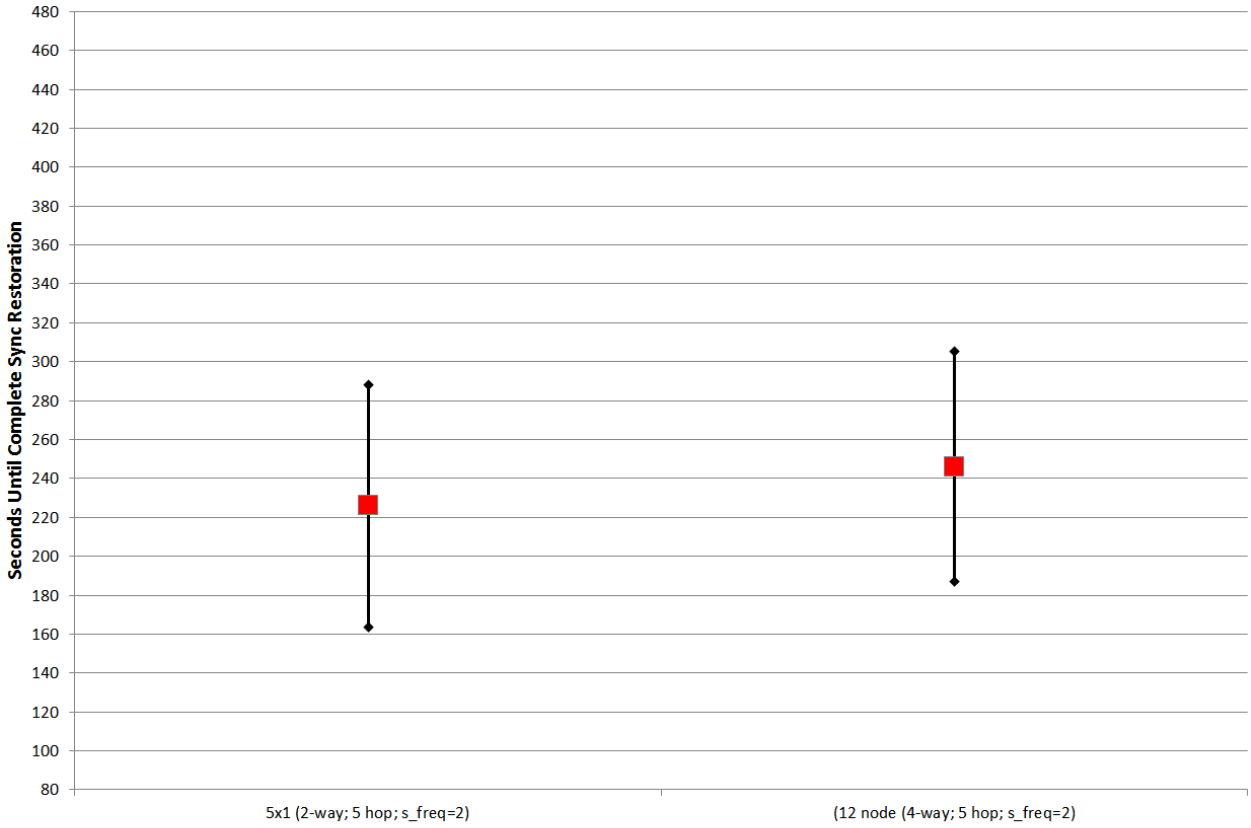

FIGURE 5.15. RSMAC Global SYNC Test: Global sync restoration proceeds only a little slower than initial global sync. Once the random neighbor discovery, the algorithm actively restores sync as quickly as possible using specialized packets and an increased sync rate.

5.4.4 RSMAC: Energy Consumption

For energy consumption, we use the same testing methodology that we used to test the optimized form of SMAC. We expect to get results that are similar to optimized SMAC with optimized neighbor discovery since we are using similar settings. Note that we do not measure energy consumption for the global sync and restore algorithms. The reason is that the situations are very infrequent and the difference comes in the way of a slight increase

in SYNC packets for a brief period. The extra energy use would be insignificant over the node lifetime. For SMAC we did not account for multiple schedules (asymmetric energy use) which is a hidden cost that can significantly harm network lifetime by draining energy faster from certain nodes.

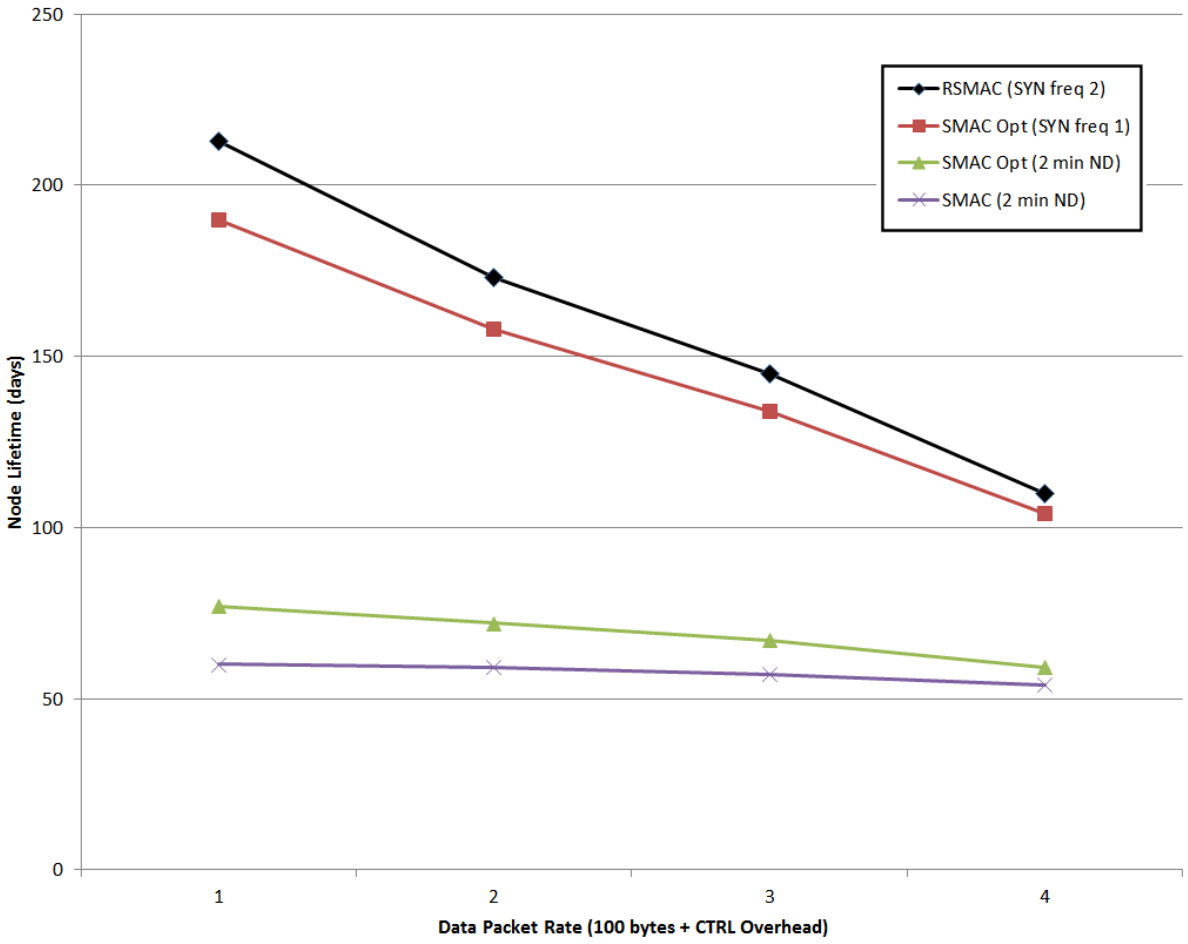

FIGURE 5.16. RSMAC Energy Use: Due primarily to savings in neighbor discovery, RSMAC saves more energy than even our optimized SMAC protocol. We recall here that RSMAC also does not have asymmetric energy use since all nodes share a loose global schedule. RSMAC may enjoy the energy savings, or use it to increase the duty cycle a little and regain the slight performance losses discussed in the RSMAC latency testing.

Figure 5.16 shows that RSMAC outperforms SMAC significantly with our tested configuration. Let us recap the optimizations for SMAC's neighbor discovery period. We found that SMAC reliable neighbor discovery (equal to frames per sync * the frame size) requires the least energy usage for a SYNC rate of 1. The extra SYNC packets simply don't compare to the extra time spent listening (so using a SYNC rate of 2 produces worse energy performance due to the longer discovery period). Hence, the only way to further optimize is to reduce the frequency of neighbor discovery from 2 minutes to a longer value. However, RSMAC implicit neighbor discovery, with a SYNC rate of 2 performs comparably to 2 minute explicit neighbor discovery. Implicit neighbor discovery is a win-win proposition for RSMAC. The energy savings can be used to regain the slight performance losses discussed in the latency and bandwidth section (by increasing the duty cycle or in other words, reducing the size of the average sleep interval). Finally, we cover the original reason for developing RSMAC.

5.5 Resistance to Jamming

The unexpected benefits of global synchronization and eliminating neighbor discovery have justified the development of RSMAC. However, our original purpose for developing RSMAC was to mitigate intelligent jamming attacks. It would be very surprising if the cluster-based jammer described by Law et al [26] could make some sense of the randomized frame size of RSMAC, but we ran tests to demonstrate the effectiveness.

5.5.1 Cluster-Based Jamming

We used our refined version of RSMAC and optimized SMAC described above for jamming tests. We tested with moderate traffic among 3 nodes with a jammer present (all nodes can hear each other) and we measured the number of failed transmissions based on RTS sent without response. For SMAC, the cluster based jammer produces a 100% censorship rate

with a life time of 110 days vs. 146 days for SMAC. Law et al reported slightly shorter lifetime for the jammer and also near 100% censorship rates when the jammer was sufficiently close to the victim nodes.

More interesting to us is the packet interarrival times for RSMAC. The unpredictability will cause the jammer to choose a jamming period that will clearly fail to match the RSMAC schedule. Instead, the jammer will function as a periodic source of interference whenever an RSMAC frame happens to synchronize with the jammers chosen period. This serves to reduce the jammer to the effectiveness of periodic jamming against an unknown schedule. The cluster based jammer settles on a false period and RSMAC provides the randomness producing the same effect as a random jammer transmitting at some consistent fraction of time [43]. The period could be increased high enough to match a constant jammer providing 100% censorship but a life time of only 3 or 4 days using the MICA 2 hardware and 2 AA batteries.

As an example run, our test version of the cluster-based jammer set a cluster size equal to about half the RSMAC average frame size after 64 packet analysis. This produced a censorship rate of 50

5.5.2 Deceptive Jamming

We also performed tests with a deceptive jammer recording a packet and periodically broadcasting it in hopes of interfering with the global SYNC and SYNC restore algorithms. However, the verification feature and cooldown successfully protect RSMAC by causing it to ignore a deceptive jammer. The deceptive jammer instead functions as another random jammer (since it does not know the RSMAC schedule) causing random interference based on its transmit duty cycle.

Chapter 6
Conclusions

We developed a novel simulation that focuses on the particular needs of Wireless Sensor Networks MAC protocols and offers many avenues for expanded use. We implemented a detailed model for the cc1000 radio chip and demonstrated the model's utility by carefully analyzing the tests performed by Ye et al for SMAC. With the aid of the simulation, we discovered an error in the network setup used by Ye et al for initial latency testing. We conclude that the controlled nature of a simulation environment is ideal for initial stress testing of MAC protocols and that one must be careful to control physical testing environments.

In addition to the simulation core which manages the interaction of the nodes composed of linked simulation modules, we developed a variety of modules, not all of which were used directly to generate test results in this dissertation. We have developed the following modules to date:

- Hardware models for the cc1000 radio used on the MICA 2, and initial prototyping for the cc2420 used on the MICA Z.
- CSMA, SMAC and RSMAC protocols
- Constant, Random, Reactive, Deceptive and Cluster-based jammers
- Test applications for generating, routing and reporting test traffic
- Support tools for presenting and analyzing data and logs

We also went on to use the simulation to test optimizations for SMAC and used the optimized SMAC as a basis to develop the new RSMAC protocol.

We found that we can significantly optimize SMAC by reducing the CTRL listen interval so that there is only enough time for a few CTRL packets. Transmissions can be started but are finished and followed with a second transmission per frame via adaptive listening. By replacing a great portion of the contention listening time with adaptive listening, idle nodes spend much more time sleeping, offering a reduction in energy spent managing DATA traffic. Throughput is reduced, but energy savings are so great that we can shorten the frame time to match the latency for isolated (congestion free) transmissions. This optimization actually improves latency and throughput slightly in times of high congestion, because adaptive listening eliminates a great part of the normal contention. In summary, we improve DATA transmission performance for high congestion while using nearly 50% less energy.

We also demonstrate that the SYNC listen interval can be shortened significantly without compromising loose synchronization and still accommodate a very high density network at the originally reported SYNC rate of 1 per 13 seconds (1 per 11 frames with a 1.15 s frame). However, we also demonstrate that a low SYNC rate requires a proportionally long period for reliable neighbor discovery and that sending SYNC packets more frequently always saves energy by shortening neighbor discovery, even up to the limit of one SYNC packet per frame. With the original settings, neighbor discovery and idle listening during the SYNC interval consume such a large percentage of energy that constant high congestion has little effect on network lifetime. However, a SYNC rate of 1 per frame can cause SYNC congestion in a dense network, so SYNC rate cannot be increased to the maximum. Instead, we must use a longer neighbor discovery time negative impacting robustness and network self-configuration performance. With neighbor discovery optimized, network lifetime is proportionally much more reliant upon frequency of data transmissions. Our fully opti-

mized form of SMAC doubles the network lifetime under persistent moderate traffic without sacrificing the neighbor discovery performance significantly. Lifetime is tripled for low traffic.

We used the optimized SMAC as the basis for RSMAC and added the randomized schedule for security against intelligent jammers. We tested traffic flows and discovered a slight decrease in performance due to the variable sleep interval. A very short interval introduces a small amount of latency because the node must pause data transmissions for the SYNC period following a sleep interval. During high congestion, throughput is similarly reduced from the level of SMAC with optimizations down to the level that SMAC achieved originally (a 10% reduction). However, energy savings due to implicit neighbor discovery make RSMAC more efficient than even optimized SMAC with the shortest explicit neighbor discovery performed every 2 minutes. RSMAC implicit neighbor discovery with a duty cycle of 5.33% and SYNC frequency of 1 per 2 frames results in 1 to 2 minutes for random neighbor discovery to occur. Hence, we can increase the duty cycle to regain the lost performance while maintaining the new security features and global synchronization of RSMAC. The main penalty of RSMAC is high variability in latency for isolated packets (variability for long traffic flows is not a factor since it is eliminated by averaging).

The randomized schedule also allows for global synchronization via the schedule ID transmitted in each SYNC packet. We tested global synchronization under worst-case conditions where nodes started with a random boot time and formed paired schedules. Global synchronization then used the implicit random neighbor discovery to synchronize the network taking approximately 30 to 45 seconds multiplied by the maximum number of hops from the node with lowest ID. Global sync restoration takes 40 to 50 seconds per hop because verification of schedules requires receiving multiple SYNC packets. In both cases, higher density and connectivity decrease the time required provided the SYNC listen interval does not become

congested by the high density combined with a high SYNC rate. This suggests further work to optimize the SYNC rate dynamically based on local density.

We tested the completed RSMAC against jammers and verified that cluster-analysis of a randomized schedule does not provide any useful information. Jamming effectiveness is reduced to jamming randomly as a percentage of time. Deceptive jammers cannot interfere with synchronization or trigger SYNC restore under normal conditions. In the worst case, a deceptive jammer could capture a series of reset packets and attempt to cause a reset as often as the cooldown timer allows. Such a reset would propagate through the network causing a minor, periodic increase in energy usage, but connectivity would be maintained.

Using randomized schedules has implications for cognitive radio that are worth mentioning. Cognitive radio requires nodes to predict network traffic and transmit in the empty spaces. In that respect, it is functionally opposite to intelligent jamming and similarly confounded by randomized scheduling. This could be seen as a security feature itself because a network using RSMAC simultaneously has security against jammers and functions as a random jammer itself against any nodes that would try to use the sleep periods to establish a secondary (unauthorized) network in the same frequency band.

References

[1] Mica 2 datasheet. Crossbow Technology, 2003.

[2] Mica z datasheet. Crossbow Technology, 2008.

[3] Mica dot datasheet. Crossbow Technology, 2010.

[4] I.F. Akyildiz, Weilian Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *Communications Magazine, IEEE*, 40(8):102 – 114, August 2002.

[5] Sidra Aslam, Farrah Farooq, and Shahzad Sarwar. Power consumption in wireless sensor networks. In *Proceedings of the 7th International Conference on Frontiers of Information Technology*, FIT '09, pages 14:1–14:9, New York, NY, USA, 2009. ACM.

[6] Jinhun Bae, Kitae Kim, and Keonwook Kim. Co-mac: energy efficient mac protocol for large-scale sensor networks. In *Proceedings of the 20th international teletraffic conference on Managing traffic performance in converged networks*, ITC20'07, pages 1072–1083, Berlin, Heidelberg, 2007. Springer-Verlag.

[7] Vaduvur Bharghavan, Alan Demers, Scott Shenker, and Lixia Zhang. Macaw: a media access protocol for wireless lan's. In *Proceedings of the conference on Communications architectures, protocols and applications*, SIGCOMM '94, pages 212–225, New York, NY, USA, 1994. ACM.

[8] Ken Biba. A hybrid wireless mac protocol supporting asynchronous and synchronous msdu delivery services. *IEEE 802.11 Working Group paper*, 1992.

[9] Chiara Buratti, Andrea Conti, Davide Dardari, and Roberto Verdone. An overview on wireless sensor networks technology and evolution. *Sensors*, 9(9):6869–6896, 2009.

[10] Chipcon Products from Texas Instruments. *Chipcon CC1000 Manual*, 2009.

[11] Chipcon Products from Texas Instruments. *Chipcon CC2420 Manual*, 2012.

[12] Kaushik R. Chowdhury, Nagesh Nandiraju, Pritam Chanda, Dharma P. Agrawal, and Qing-An Zeng. Channel allocation and medium access control for wireless sensor networks. *Ad Hoc Netw.*, 7:307–321, March 2009.

[13] Crossbow Technology. *MPR-MIB Users Manual*, 2007.

[14] C.C. Enz, A. El-Hoiydi, J.-D. Decotignie, and V. Peiris. Wisenet: an ultralow-power wireless sensor network solution. *Computer*, 37(8):62 – 70, 2004.

[15] Aurlien Francillon and Claude Castelluccia. Tinyrng: A cryptographic random number generator for wireless sensors network nodes. In *In Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks and Workshops, 2007. WiOpt 2007. 5th International Symposium on*, pages 1–7, 2007.

[16] J.C. Haartsen. The bluetooth radio system. *Personal Communications, IEEE*, 7(1):28 –36, February 2000.

[17] G. P. Halkes and K. G. Langendoen. Crankshaft: an energy-efficient mac-protocol for dense wireless sensor networks. In *Proceedings of the 4th European conference on Wireless sensor networks*, EWSN'07, pages 228–244, Berlin, Heidelberg, 2007. Springer-Verlag.

[18] Wendi Rabiner Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the 33rd Hawaii International Conference on System Sciences-Volume 8 - Volume 8*, HICSS '00, pages 8020–, Washington, DC, USA, 2000. IEEE Computer Society.

[19] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. *SIGPLAN Not.*, 35:93–104, November 2000.

[20] Christine E. Jones, Krishna M. Sivalingam, Prathima Agrawal, and Jyh Cheng Chen. A survey of energy efficient network protocols for wireless networks. *Wirel. Netw.*, 7:343–358, September 2001.

[21] Rajgopal Kannan, Shuangqing Wei, Costas Busch, and Athanasios Vasilakos. Online algorithms for maximizing quality of link transmissions over a jammed wireless channel. In *Proceedings of the 2009 IEEE International Conference on Pervasive Computing and Communications*, pages 1–6, Washington, DC, USA, 2009. IEEE Computer Society.

[22] Chris Karlof, Naveen Sastry, and David Wagner. Tinysec: a link layer security architecture for wireless sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, SenSys '04, pages 162–175, New York, NY, USA, 2004. ACM.

[23] Chris Karlof, Naveen Sastry, and David Wagner. *TinySec: User Manual*. Computer Science at Berkeley, 2004.

[24] Phil Karn. Maca: A new channel access method for packet radio. In *Proceedings of the 9th ARRL Computer Networking Conference*, 1990.

[25] Yee Wei Law, Pieter Hartel, J. den Hartog, and Paul Havinga. Link-layer jamming attacks on s-mac. In *2nd European Workshop on Wireless Sensor Networks*, EWSN '05, pages 217–225. IEEE, 2005.

[26] Yee Wei Law, Lodewijk van Hoesel, Jeroen Doumen, Pieter Hartel, and Paul Havinga. Energy-efficient link-layer jamming attacks against wireless sensor network mac protocols. In *Proceedings of the 3rd ACM workshop on Security of ad hoc and sensor networks*, SASN '05, pages 76–88, New York, NY, USA, 2005. ACM.

[27] Jongdeog Lee, Krasimira Kapitanova, and Sang H. Son. The price of security in wireless sensor networks. *Comput. Netw.*, 54(17):2967–2978, December 2010.

[28] Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk, and John Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, WSNA '02, pages 88–97, New York, NY, USA, 2002. ACM.

[29] Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk, and John Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, WSNA '02, pages 88–97, New York, NY, USA, 2002. ACM.

[30] Miklós Maróti, Branislav Kusy, Gyula Simon, and Ákos Lédeczi. The flooding time synchronization protocol. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, SenSys '04, pages 39–49, New York, NY, USA, 2004. ACM.

[31] LAN MAN Standards Committee of the IEEE Computer Society. Wireless lan medium access control (mac) and physical layer (phy) specification, 1999.

[32] Miroslav Pajic and Rahul Mangharam. Anti-jamming for embedded wireless networks. In *Proceedings of the 2009 International Conference on Information Processing in Sensor Networks*, IPSN '09, pages 301–312, Washington, DC, USA, 2009. IEEE Computer Society.

[33] Joseph Polastre, Jason Hill, and David Culler. Versatile low power media access for wireless sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, SenSys '04, pages 95–107, New York, NY, USA, 2004. ACM.

[34] Injong Rhee, Ajit Warrier, Mahesh Aia, and Jeongki Min. Z-mac: a hybrid mac for wireless sensor networks. In *Proceedings of the 3rd international conference on Embedded networked sensor systems*, SenSys '05, pages 90–101, New York, NY, USA, 2005. ACM.

[35] Anthony Rowe, Rahul Mangharam, and Raj Rajkumar. Rt-link: A global time-synchronized link protocol for sensor networks. *Ad Hoc Netw.*, 6:1201–1220, November 2008.

[36] Fernando Royo, Miguel Lopez-Guerrero, Teresa Olivares, and Luis Orozco-Barbosa. A fast network configuration algorithm for tdma wireless sensor networks. *EURASIP J. Wirel. Commun. Netw.*, 2010:8:1–8:10, February 2010.

[37] Simon J. Shepherd. The tiny encryption algorithm. *Cryptologia*, 31(3):233–245, July 2007.

[38] Incheol Shin, Yilin Shen, Ying Xuan, My Tra Thai, and Taieb Znati. Reactive jamming attacks in multi-radio wireless sensor networks: an efficient mitigating measure by identifying trigger nodes. In *Proceedings of the 2nd ACM international workshop on Foundations of wireless ad hoc and sensor networking and computing*, FOWANC '09, pages 87–96, New York, NY, USA, 2009. ACM.

[39] Suresh Singh and C. S. Raghavendra. Pamas-power aware multi-access protocol with signalling for ad hoc networks. *SIGCOMM Comput. Commun. Rev.*, 28:5–26, July 1998.

[40] K. Sohrabi, J. Gao, V. Ailawadhi, and G.J. Pottie. Protocols for self-organization of a wireless sensor network. *Personal Communications, IEEE*, 7(5):16 –27, October 2000.

[41] Yanjun Sun, Omer Gurewitz, and David B. Johnson. Ri-mac: a receiver-initiated asynchronous duty cycle mac protocol for dynamic traffic loads in wireless sensor networks. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, SenSys '08, pages 1–14, New York, NY, USA, 2008. ACM.

[42] Gilman Tolle, Joseph Polastre, Robert Szewczyk, David Culler, Neil Turner, Kevin Tu, Stephen Burgess, Todd Dawson, Phil Buonadonna, David Gay, and Wei Hong. A macroscope in the redwoods. In *Proceedings of the 3rd international conference on Embedded networked sensor systems*, SenSys '05, pages 51–63, New York, NY, USA, 2005. ACM.

[43] Wenyuan Xu, Wade Trappe, Yanyong Zhang, and Timothy Wood. The feasibility of launching and detecting jamming attacks in wireless networks. In *Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*, MobiHoc '05, pages 46–57, New York, NY, USA, 2005. ACM.

[44] Wei Ye, John Heidemann, and Deborah Estrin. A flexible and reliable radio communication stack on motes. Technical report, Tech. Rep. ISI-TR-565, USC Information Sciences Institute, 2002.

[45] Wei Ye, John Heidemann, and Deborah Estrin. Medium access control with coordinated adaptive sleeping for wireless sensor networks. *IEEE/ACM Trans. Netw.*, 12:493–506, June 2004.

Vita

David Trammell was born in New Orleans, Louisiana. He finished his undergraduate studies at Nicholls State University in December 2002. In January 2004 he came to Louisiana State University to pursue graduate studies in computer science. He began working full time as a Systems Manager in August 2008 while staying enrolled in the LSU graduate program. He is currently a candidate for the degree of Doctor of Philosophy in computer science, which will be awarded May 2012.