December 2009

A Comparative Study of Energy Efficient Medium Access Control Protocols in Wireless Sensor Networks

Andrew Zakaria Keating
Worcester Polytechnic Institute

Brian Scott Bates
Worcester Polytechnic Institute

A Comparative Study of Energy Efficient Medium Access Control Protocols in Wireless Sensor Networks

A Major Qualifying Project

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

in Computer Science

by

Brian Bates

Andrew Keating

December 2009

Approved:

Professor Robert Kinicki, Advisor

Abstract

Wireless sensor "motes"—small embedded systems equipped with radios for wireless communication—are almost always expected to operate unattended for long periods of time, relying upon batteries as a power source. As the vast majority of energy consumed by motes can be attributed to radio usage, the development of medium access control (MAC) protocols which seek to minimize energy consumption is an important area of wireless sensor network research. Recent MAC protocols reduce energy usage by placing the radio in a low-power sleep state when not sending or receiving transmissions. This project investigates energy usage in three such protocols—AS-MAC, SCP-MAC, and Crankshaft–on physical sensor hardware. It additionally presents BAS-MAC, an energy-efficient protocol of our own design. We evaluate our implementation of these four protocols on TelosB motes over multiple sensor network topologies. Our evaluations show that in single-hop networks with large send intervals and staggered sending, AS-MAC is best in the local gossip and convergecast scenarios, while SCP-MAC is best overall in the broadcast scenario. We conjecture that Crankshaft would perform best in extremely dense hybrid (unicast and broadcast) network topologies, especially those which broadcast frequently. Finally, BAS-MAC would be optimal in networks which utilize hybrid traffic with infrequent broadcasts, and where broadcasting is performed by motes that do not have an unlimited power source.

Acknowledgments

We would like to thank Professor Kinicki for his exceptional guidance in advising this project. We would also like to thank AS-MAC author Jun Bum Lim for his technical assistance.

Contents

List of Figures

List of Tables

Executive Summary

The purpose of this investigation was to perform a fair and thorough physical evaluation of recent medium access control (MAC) protocols for wireless sensor networks (WSNs). WSNs are used to solve many real-world problems, such as periodic monitoring, event detection and tracking. They are commonly deployed in medical, agricultural and military applications, among others. Wireless sensor "motes"—small embedded systems equipped with radios for wireless communication—are almost always expected to operate unattended for long periods of time, relying upon batteries as a power source. As the vast majority of energy consumed by motes can be attributed to radio usage, the development of MAC protocols which seek to minimize energy consumption is an important area of wireless sensor network research. This project involved the implementation of four energy-efficient WSN MAC protocols and a fair evaluation and analysis of their energy consumption.

Many sources of energy waste exist in wireless sensor networks. Listening to a wireless channel while no transmissions are occurring, known as idle listening, occurs very frequently in the popular Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) wireless MAC protocol. This energy usage is not critical in wireless communications where dedicated power sources are present, but in battery-powered WSNs, it is completely unacceptable and severely decreases network lifetime. Another source of energy dissipation, known as overhearing, occurs when a node in a WSN receives a transmission intended for another recipient. The growing popularity of packet-based radios, in which an entire packet must be received before its recipient header field can be inspected, demands innovative overhearing avoidance strategies. When two nodes attempt to transmit simultaneously on the same channel, their transmissions collide and the message cannot be properly received. These collisions are costly in WSNs, where either both nodes must expend energy to retransmit (again with the risk of further collision), or in protocols which don't utilize acknowledgment-based reliability, the transmissions may be lost forever. Early WSN MAC protocols sought to minimize energy waste by placing the radio in a low-power sleep state when not sending or receiving transmissions, a process known as duty cycling. The next generation of protocols introduced low-power listening, a technique that employs transmission preambles and channel polling to significantly reduce idle listening. The current state of the art for WSN MAC protocols consists of hybrid solutions which employ their own variations of these two techniques.

It can be difficult to determine the best protocol for a given application. Often experiments performed in the literature are biased in favor of the authors' protocols and conveniently overlook significant weaknesses. After carefully analyzing dozens of academic publications presenting energy-efficient WSN MAC protocols, it was determined that the most advanced ones were SCP-MAC, Crankshaft and AS-MAC. The first obstacle

to overcome was the fact that only one of these three protocols was available on the Internet, and it was a platform-specific implementation for an outdated version of the popular TinyOS sensor operating system. The solution to this problem was to create protocol implementations standardized around a common reusable framework–the MAC Layer Architecture (MLA). This process involved acquiring a deep understanding of each protocol from the literature and performing careful analysis of which MLA components could be shared across protocols. The three MAC protocols were implemented on the most recent version of TinyOS (TinyOS version 2.1) and their correctness was validated by reproducing the experiments performed by their authors. While implementing AS-MAC, it was realized that through a relatively minor modification–namely the addition of variable-frequency broadcasting slots–the protocol could become much more robust in some situations. This realization led to the invention of Broadcastable AS-MAC (BAS-MAC), which was evaluated alongside the three aforementioned protocols.

Accurately measuring energy consumption in wireless sensor networks is a complicated task. Several energy measurement strategies are employed in WSN publications, each with an associated set of strengths and weaknesses. Mathematical modeling and software simulation are sometimes sufficient for estimating energy consumed by simple tasks, but the complexity of WSN MAC protocols and the uncertainty associated with wireless signal propagation demands more robust methods. Thus ten Crossbow TelosB wireless sensor motes were acquired to perform physical energy consumption experiments. Duty cycling protocols cause the radio to shift between five states–idle, sending, receiving, starting and stopping–all of which consume different amounts of energy. To measure the amount of energy consumed in each state, hooks were inserted into the TelosB's radio drivers which signaled general-purpose input/output (GPIO) pins to indicate changes in the state of the radio. Using an oscilloscope's digital input lines to monitor the GPIO pins and its analog current probes on either side of a 10.2 Ω resistor, voltage drop was measured and used to calculate current flow in each radio state. Developing a TinyOS component which computes and records the time duration the radio spends in each state made it possible to calculate energy usage over a variety of traffic patterns.

The next challenge was creating a set of experiments capable of evaluating the four protocols without bias. The first major decision with regard to experimental design was to only consider single-hop network topologies in which routing was not a factor. Additionally, the experiments were designed to minimize the probability of transmission collisions. Eliminating these factors creates the opportunity to draw very certain and specific cause-and-effect conclusions from the experimental results. Three common WSN traffic patterns were investigated–local gossip, broadcast and convergecast. A total of 48 experiments were conducted over varying network sizes and offered loads, with each three-minute experiment being repeated five times for increased statistical significance. It was found that AS-MAC is generally optimal for scenarios which strictly utilize unicast (single-recipient) traffic, as a result of its ability to completely eliminate overhearing.

SCP-MAC, which suffers from large amounts of overhearing, was found to perform best in broadcast-only situations. Additionally, it is speculated that SCP-MAC is capable of performing well in any wireless sensor network which utilizes byte-based radios, as opposed to packet-based. Crankshaft performs best in extremely dense hybrid (unicast and broadcast) network topologies, especially those which broadcast frequently. Finally, BAS-MAC was found to be optimal in networks which utilize hybrid traffic with infrequent broadcasts, and where broadcasting is done by motes that do not have an unlimited power source.

This project was successful in achieving an unbiased energy consumption evaluation of recent energy-efficient MAC protocols from literature, and demonstrating that, as of yet, no single MAC protocol is ideal in all situations. In addition to the evaluation of existing protocols, this project presented Broadcastable Asynchonous Scheduled MAC (BAS-MAC), an energy-efficient WSN MAC protocol for hybrid-traffic networks utilizing infrequent broadcasts. Further, a methodology for comparative evaluation of WSN MAC protocols, including TinyOS implementations for AS-MAC, BAS-MAC, Crankshaft and SCP-MAC which utilize a common reusable framework can be extremely useful for future research in this area.

1 Introduction

A Wireless Sensor Network (WSN) is a collection of sensor nodes, commonly referred to as motes, which can utilize wireless radio communication to solve real-world problems. There are many possible applications of wireless sensor networks, including intelligent buildings, facility management, surveillance, and precision agriculture [21]. As one example, an elephant herd migration pattern tracking application could consist of a number of dispersed motes throughout a habitat which are able to sense and communicate data about elephant movements.

Wireless sensor motes are usually composed of simple hardware components, not much more than a micro-controller, a wireless radio, memory, a battery, and appropriate data gathering sensors. This presents a number of unique challenges to overcome when designing an effective sensor network. One such challenge is extending the lifetime of the network for as long as possible. Network lifetime is a broad term, which may mean many things, including the time until the first mote in a network fails, the time before a certain number of motes in the network fail, or the time until an end-to-end path in the network is broken. However, disregarding hardware failure, all of these scenarios hinge on the energy consumption of the motes in the network. In some applications, such as habitat monitoring, motes are connected to a limited power source, such as a battery, and it is infeasible to replace the power source of individual sensor motes when they run out of energy. Other constraints on these types of applications, such as throughput or latency, may not be as important as mote lifetime or energy usage. Therefore, it becomes necessary to design WSN applications to consume as little energy as possible.

This project investigates and compares state of the art wireless sensor network MAC protocols. The primary goal of the project is to comparatively measure energy usage across the AS-MAC [19], SCP-MAC [39], and Crankshaft [8] protocols, as well as BAS-MAC, our own extension of the AS-MAC protocol. In particular, we ran energy measurement experiments in a number of single-hop network scenarios, including broadcasting, converge-casting, and local gossip communication. Energy usage is measured to determine which protocol is best suited for a particular energy-conscious sensor network application. These experiments reveal the relative strengths and weaknesses of each protocol, and show that no one protocol is best for all sensor network tasks and traffic patterns.

Section 2 of this report introduces the background material needed to understand the topic of wireless sensor networks in the research space relevant to our experiments. This includes discussions on the underlying languages and frameworks used for software development on sensor motes, common network topologies in wireless sensor network research and applications, descriptions of the MAC protocols investigated in our experiments, and current methods for measuring energy. In section 3 we discuss our research methodology,

which includes our implementations of the MAC protocols and the other software needed to run our experiments, a description of the energy measurement techniques employed, descriptions of our experiments, and the MAC layer parameters used in these experiments. Section 4 displays the results of our experiments for the local gossip, convergecast, and broadcast scenarios. We state the conclusions of our research efforts in section 5 and discuss possible directions for future research in section 6.

2 Background

In a wireless sensor network, much of the energy consumed is through use of the radio. In fact, it costs roughly the same amount of energy to transmit 1K of data 100m as it does to perform 3 million MIPS instructions in a general purpose processor [30]. Not just transmitting data, but simply having an actively listening radio incurs significant energy usage. However, motes like the MicaZ and TelosB are equipped with radios that can enter a low-power sleep state when not being used. These radios, when in a sleep state, draw current at a rate that is three magnitudes less (μA as opposed to mA) than when they are active [15, 16]. Table 1 shows a comparison of current draw for different hardware components and states for the TelosB and MicaZ motes.

Mote	Radio (Active)	Radio (Sleeping)	Processor (Active)	Processor (Sleeping)
TelosB	23 mA	1 μA	1.8 mA	5.1 μA
MicaZ	19.7 mA	1 μA	8 mA	< 15 μA

Table 1: Comparison of TelosB and MicaZ current draw [15, 16]

These observations lead to the conclusion that radio components should be active only when absolutely necessary and that more computationally complex networking protocols should be used in order to reduce the time that a radio is active. One such place to make energy efficient optimizations is within the medium access control, or MAC, layer of the network stack. In a WSN, the MAC layer is responsible for coordinating when the different entities on the network access their shared communication medium, in this case, a particular wireless channel. Since the MAC layer resides above the physical hardware layer and directly controls when the radio is turned on and off, an energy efficient MAC protocol has the potential to drastically reduce the amount of energy used by the radio and increase mote lifetime.

The remainder of this chapter discusses the current research, tools and technologies in the rapidly developing field of wireless sensor networks. The section begins by first introducing important terminology in the research space. It then examines WSN operating systems, programming languages, network topologies, MAC protocols, software frameworks and energy measurement techniques.

2.1 Terminology

The knowledge of a key set of terms is crucial to understanding WSN MAC protocols and their fundamental issues. The first important concept is duty cycling. The duty cycle of the radio represents the percentage of time that a mote activates its radio in order to send or receive a message. A powerful technique in designing an energy efficient MAC protocol is to reduce the duty cycle of a mote as much as possible, since radios usually consume the majority of the energy in wireless sensor network applications. Decreasing the duty

cycle can be achieved by reducing idle listening, which refers to times in which a mote is actively listening for an incoming message but not receiving one.

Another term that we use throughout this report is that of a wakeup slot. A wakeup slot refers to the interval of time in which a mote turns on its radio in order to listen for an incoming message. It is important to note that this is different from the wakeup interval for a mote. We define the wakeup interval of a mote to be the amount of time in between wakeup tones. As an example, if a sensor mote were to turn on its radio and listen for a message for 5 ms before going back to sleep, and it repeats this process once every second, then the wakeup tone would be 5 ms and the wakeup interval would be one second.

When dealing with wireless MAC protocols, it is possible for a mote in the network to listen for and receive a message that was not intended for it. This phenomenon is known as overhearing, and can lead to wasted energy.

A significant difference between wireless sensor networks and other communication networks is the manner in which contention over the shared transmission medium is resolved. Although different MAC protocols are free to employ various forms of contention resolution, the protocols studied in this investigation all handle contention in a similar fashion. Namely, contention is resolved in an interval of time known as the contention window, which is divided into equal sized slots. When a mote wishes to send a message, it chooses a particular slot, usually based on a discrete probability distribution. During this slot, the mote checks the transmission medium (called the channel). If the channel is busy, the mote does not send the message, and instead tries to send the message at a later time. If the channel is free, then the mote transmits over the remainder of the contention window.

In this form of contention resolution, a collision occurs when two or more motes choose the same contention slot, and this slot also happens to be the earliest slot chosen by any of the motes. In this case, these motes all believe that they have access to the channel, causing them to all send their message at the same time. This causes message collisions that usually result in failed transmissions. Depending on the behavior of the wireless MAC protocol, these collisions may cause the sending mote to retransmit the message.

2.2 Wireless Sensor Network Operating Systems

Due to the computation and memory constraints of the hardware on a sensor mote, it is infeasible to install a traditional operating system onto a mote. Moreover, since motes usually only run a single application, many of the benefits of a complex operating system, such as multi-threading, process-scheduling, virtual memory management, and dynamic allocation of memory, are not utilized. TinyOS [26] is an operating

system designed specifically for wireless sensor networks. TinyOS uses a component-based architecture in which discrete programmatic components are "wired" together in order to create a working application. This wiring allows TinyOS to perform simple hardware abstractions. The execution of a TinyOS application is event-driven; code sections are triggered from responses to the mote's outside environment, such as sensors, timers, and the radio. TinyOS also accounts for the limited resources available on mote hardware, with only a 400 byte operating system overhead.

The most recent stable version of TinyOS is 2.1.0. Version 2 of TinyOS is a complete "clean slate" redesign from the first version. It includes new structures and interfaces for many operating system components which were found to be fundamentally flawed in version 1, and the two versions are not backwards compatible. Some of the major redesigns are in the areas of storage, network protocols, resource management, task scheduling, booting, timing, and the addition of a low-power radio stack.

2.2.1 TinyOS Event Model

A TinyOS application consists of a discrete number of components that are connected through interfaces. However, unlike traditional programmatic interfaces, TinyOS interfaces are bi-directional. In addition to the ability for the consumer of an interface to issue a command to its provider, providers have the ability to signal event callbacks on consumers. This relationship allows for a split-phase, non-blocking paradigm to emerge–the same paradigm that is typically seen in hardware.

In the split-phase model, the consumer of an interface signals a request for a particular service. The provider of the interface is then responsible for performing the service, and signaling an event to the consumer when the service has completed. Programmatically, in order for these operations to be non-blocking, TinyOS introduces the concept of tasks, which are code segments that are scheduled to run at a later time. Tasks are put in a run-to-completion, first-come, first-serve queue. In essence, one component calls a command on another component. The receiving component creates a task to be completed, and when this task is eventually run to completion, it signals an event back to the original calling component.

2.2.2 nesC

TinyOS is written in nesC, a customized dialect of the C programming language [6]. The nesC compiler accepts the command, event and task constructs of TinyOS, and the development of nesC and TinyOS have been highly intertwined. NesC complies its raw code into a form that the gcc compiler can use, after which

gcc builds the TinyOS application into bytecode that can be run on a mote or in a simulator.

Because nesC is a specialized version of the C language designed for use on wireless sensor networks, it contains some fundamental differences from that of traditional C. One of the largest differences is that nesC is a "static language" and does not allow for the dynamic allocation of memory. This prevents memory fragmentation and run-time memory errors, such as null pointer exceptions. In addition to this, nesC does not allow function pointers. This allows the compiler to know the call path of an application at compile time and perform more rigorous optimizations on the code, resulting in crucial memory savings.

2.3 Sensor Network Topologies

Different applications of wireless sensor networks operate over a variety of mote placements, forming network topologies with unique characteristics and requirements. Here, we introduce three network topologies which are commonly utilized in WSNs–the star, grid and line. It is often most practical to perform research over simple employments of these network organizations. However, in reality these three topologies tend to be building blocks for more complex, hybrid topologies. The concept of a base station or sink (these terms are used interchangeably) is important in understanding WSN network topologies. In most cases, base stations have significantly more computation and energy resources available than other members of the network. They are often used as aggregation points in data gathering applications, and can also serve as gateways, for example if they are connected to the Internet.

2.3.1 Convergecast and Broadcast Traffic Patterns

A star network is a common WSN topology consisting of a central base station surrounded by one or more motes at a distance of a single hop. The advantage of this topology is its simplicity, but it runs the risk of flooding the base station, and it also has a single point of failure. The task of routing packets between nodes is not a concern in this network layout, which can result in significant energy savings for some applications. The star topology is the central concept of clustering, or organizing a group of nodes around a common "sink" node, an area of WSN research with clear energy conserving implications.

Star network topologies often employ two common traffic patterns–convergecast and broadcast. In the convergecast traffic pattern, leaf nodes transmit inwards toward the base station. In the broadcast traffic pattern, the base station broadcasts outwards toward the leaf nodes. By studying these traffic patterns within a star topology, one eliminates the complex effects of multi-hop routing from an energy usage analysis.

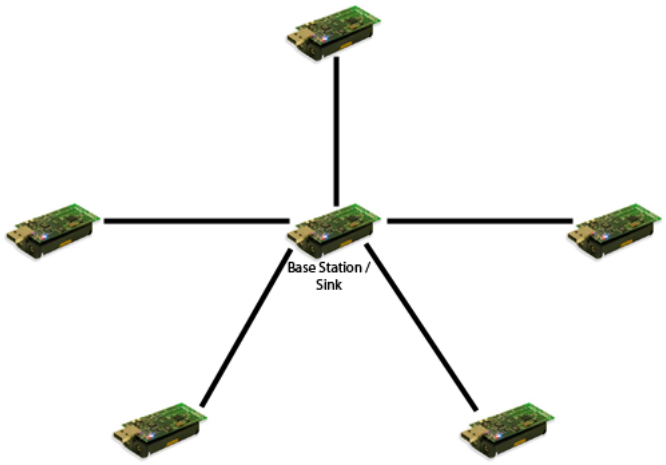

Figure 1: Star Topology

2.3.2 Line and Grid Topologies

The grid and line topologies are multi-hop network organizations which do not share the same convenience as the star topology, in that they must be concerned with routing.

A line topology is a simple multi-hop network which represents an end-to-end path between nodes. In this topology, information is routed through neighboring nodes for communications to and from the base station. An analogy to the line topology is a circuit-switched path in a wired network.

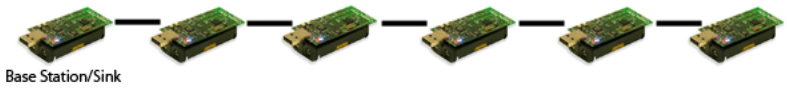

Figure 2: Line Topology

A grid topology adds a degree of complexity by incorporating multiple paths into the network organization. A common application of the grid topology is in agricultural sensing–for example, monitoring soil conditions in a field of crops. One can imagine a rectangular grid of sensor nodes which periodically collect moisture sensor readings, then transmit them to a base station at the corner of the grid for processing.

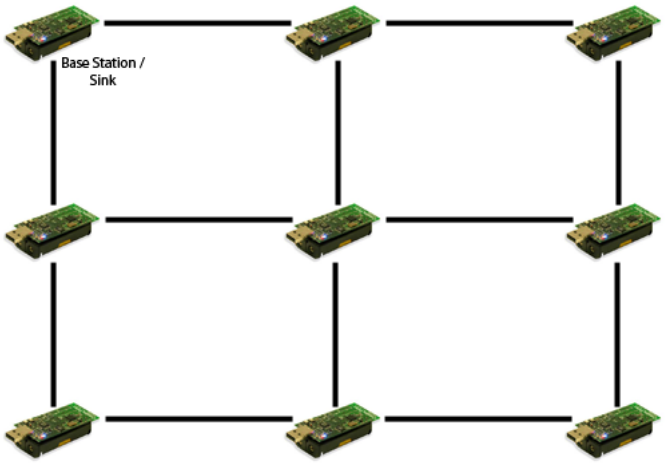

Figure 3: Two-Dimensional Grid Topology

2.3.3 Local Gossip

The local gossip traffic pattern occurs within a pairwise network topology, in which each node pairs itself with another. In this traffic pattern, nodes only communicate with the other node in their pair. This topology is not focused around a central base station, and it is mostly employed in ad-hoc networks.

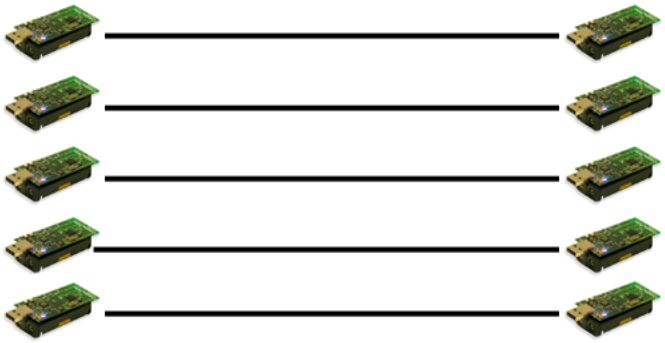

Figure 4: Local Gossip

2.4 Early Wireless Sensor Network MAC Protocols

The high energy cost of radio usage has motivated research in energy-efficient Medium Access Control (MAC) protocols. Several sources of energy waste exist in WSN radio communication, such as idle listening,

overhearing and collisions. Early solutions to the problem of conserving energy at the MAC layer can be split into two categories: TDMA-based and CSMA-based protocols.

2.4.1 TDMA Protocols

Time Division Multiple Access is a mechanism for sharing a communication channel in which time slots are assigned to different users for sending and receiving. TDMA very naturally supports duty cycling, as a mote can keep its radio in the sleep state until its time slot occurs. These protocols work well in theory. However, the dynamic nature of sensor networks detracts from the real-world feasibility of TDMA protocols. For example, if a mote is introduced to an active sensor network running a TDMA protocol, the entire network's time schedule must be updated. Some of the difficulties faced in implementations of TDMA protocols can be combated by employing clustering schemes such as LEACH [9], but in general, CSMA-based solutions are both simpler and more practical.

2.4.2 CSMA Protocols

Carrier Sense Multiple Access (CSMA) is a mechanism for sharing a communication channel in which a transmitter "senses" the channel before attempting to send data. The transmitter will only proceed to send if the channel is found to be available (e.g., another sender is not currently transmitting). A CSMA protocol which includes extra functionality for avoiding collisions–CSMA/CA–is featured in the IEEE 802.11 specification for wireless LANs.

S-MAC [38] introduced the concept of neighbor-synchronized CSMA duty cycling. Designed with periodic-sensing ad-hoc wireless sensor networks in mind, S-MAC seeks to minimize energy consumption with the trade-off of reduced throughput and increased latency. This is achieved by dictating transmission and listening periods through a schedule which is synchronized among neighbors. When a mote is not scheduled to transmit or receive, its radio conserves energy by remaining in the sleep state. In addition to saving energy by sleeping, this also reduces the wasteful reception of broadcasts intended for other recipients, or overhearing. Although the maintenance of the schedule and unnecessary listening during quiet contention periods introduces overhead, S-MAC was a very important predecessor to more modern WSN MAC protocols.

Hill and Culler [10] introduced the concept of preamble sampling (also known as channel polling) for reduced cost of idle listening. This concept has come to be known as LPL, or Low-Power Listening. LPL exhibits very low duty cycles during "quiet" times–periods when no data exchange is occurring. However,

traditional LPL protocols trade reduced receiving energy for increased sending energy. This is because senders must transmit a preamble at least as long as the wakeup interval in order to guarantee their intended receiver gets a message. WiseMAC [4] and B-MAC [29] introduced optimizations to the Low-Power Listening scheme by reducing channel polling length via scheduling. The concepts of Low-Power Listening and duty cycling are central in state-of-the-art CSMA-based WSN MAC protocols; many recent protocols use one or both of these power-saving techniques.

2.5 State of the Art MAC Protocols for Wireless Sensor Networks

A number of more advanced wireless sensor network MAC protocols have emerged in recent years, building on older protocols, as well as focusing more on the specific needs and constraints of WSNs. From these newer MAC protocols, SCP-MAC, AS-MAC and Crankshaft claim to be significantly more energy efficient than their predecessors.

2.5.1 SCP-MAC

The Schedule Channel Polling MAC Protocol [39], or SCP-MAC, aims to reduce the duty cycle for listening radios and adapt to variable traffic loads through use of scheduling and channel polling. SCP-MAC modifies the Low Power Listening (LPL) approach of earlier MAC protocols, scheduling neighboring motes in the network to listen for messages at the same time. This allows for a short message preamble, and on average keeps receiving motes awake for less time. Figure 5 shows the difference in preamble length between SCP-MAC and traditional low power listening. Motes remain awake for less time with shorter preambles because with longer preambles, motes that start listening near the beginning of the preamble must remain listening for the duration of the preamble in order to successfully intercept the packet.

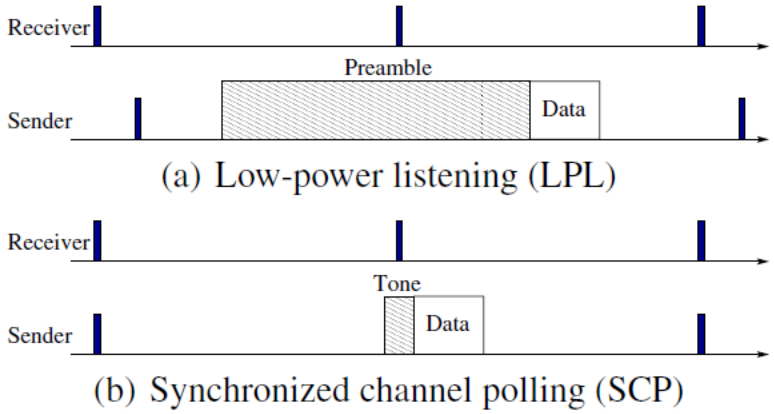

Figure 5: SCP-MAC Preamble Versus Low Power Listening [39]

Synchronization is handled in SCP-MAC in two ways–through an initial bootstrapping phase, and also through the periodic exchange of synchronization packets. When a mote initially wishes to join a network, it needs to discover when the scheduled wakeup time is. To do this, it falls back on the B-MAC [29] protocol, a traditional low power listening protocol. Using this protocol, it sends a message with a preamble long enough to hit the wakeup tone of motes in the network and request synchronization information. Secondly, to account for clock drift over time, SCP-MAC schedules synchronization packets to be sent periodically. However, the synchronization information is designed to be "piggybacked" onto data messages, so if motes communicate at a rate faster than the drift, then explicit synchronization packets become unnecessary.

SCP-MAC also contains a number of optimizations to enhance the protocol. One optimization is adaptive channel polling, in which additional polling slots are dynamically added in order to increase throughput on a bursty network. Another optimization involves using a two-phase contention scheme in order to avoid collisions among competing senders, portrayed in Figure 6. To avoid overhearing, receiving motes in SCP-MAC can inspect header information in the incoming packet, and discontinue listening to the packet if they are not the intended receiver.

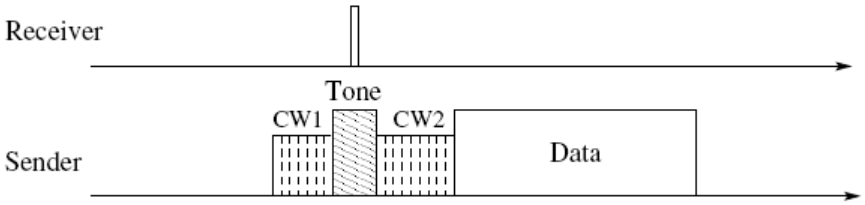

Figure 6: SCP-MAC Two-Phase Contention [39]

However, there are disadvantages to the SCP-MAC protocol, the most prominent being overhearing. Overhearing occurs when a mote listens to a packet for which it is not the intended recipient. Because all

neighboring motes share the same time slot for listening to messages, overhearing occurs often using this protocol. Although the authors of SCP-MAC suggest the optimization where motes for whom the packet is not intended cease to listen after reading a different address in the packet's header, this optimization is not feasible in newer radios, such as the CC2420, which are packet based [19, 39]. A packet based radio must listen to the entirety of a packet before deciphering its contents, and therefore, the mote is unable to prematurely stop listening to the channel and reduce expending energy on overhearing.

2.5.2 AS-MAC

The Asynchronous Scheduled MAC Protocol [19], or AS-MAC, tries to reduce overhearing by assigning motes different time slots in which to listen for packets. However, AS-MAC requires each mote to store synchronization information about all of its neighbors in the network. This information determines when motes wishing to send information should wake up for transmission.

The AS-MAC protocol is divided into two phases: the initialization phase and the periodic listening and sleep phase. During the initialization phase, a mote listens for a pre-determined amount of time for "hello" packets, which contain neighbor information. The neighbor information sent includes the wake-up interval, the hello packet interval for that neighbor, and the wake-up offset for that neighbor. After the new mote is done listening for neighbors, it builds a neighbor table from the information gathered, determines a unique wake-up offset for itself, and transmits this offset to all of its newly discovered neighbors. A common algorithm for determining when the mote's wake-up offset will be is to choose the slot halfway between the largest empty interval among all of the mote's neighbors, an example of which can be seen in Figure 7. Upon establishing this wakeup offset, the mote enters the periodic listening and sleep phase.

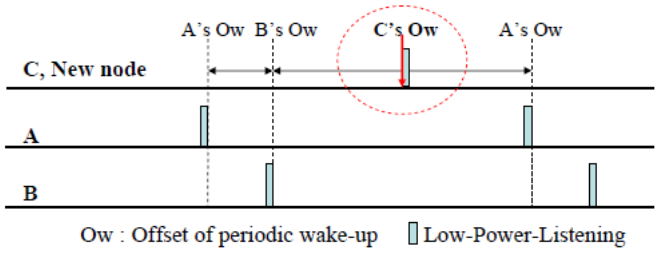

Figure 7: Choosing a Wake-up Slot in AS-MAC [19]

In this second phase, a mote periodically wakes up at its given interval, performing LPL to detect an incoming message. If the mote detects a busy channel, it then listens for an incoming message. However, during one wakeup period in a given interval, known as the hello interval, the mote is first scheduled to

transmit a hello packet. Receiving motes transmit this packet before receiving a message. Senders must also be aware of when the recipient is transmitting a hello packet, and if this is the case, wait until after this transmission to send their data. A depiction of this can be seen in Figure 8. In this diagram, two senders both wish to transmit to a single receiver. The receiver wakes up at its scheduled time and transmits a hello packet, which is received by both senders. The senders then contend for the right to seize the channel, and as the second sender is victorious, it transmits its data to the receiver.

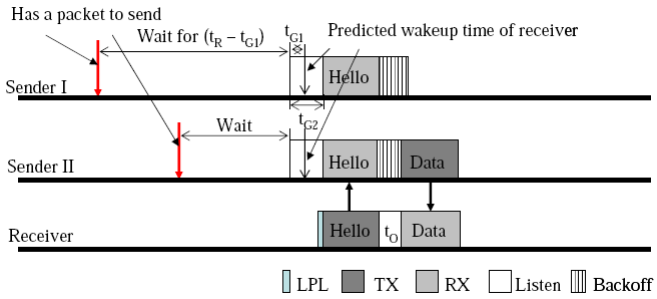

Figure 8: Behavior of AS-MAC at Hello Time [19]

Although AS-MAC reduces overhearing by assigning unique listening slots for receivers, there are a number of disadvantages to the protocol. One disadvantage is the memory overhead associated with maintaining a neighbor table. Each mote must store three pieces of information about each neighbor, and a mote may have many neighbors in a dense network. Another disadvantage of the protocol is the transmission overhead of hello packets. Although this mechanism allows new motes to dynamically enter the network, it also forces all motes to periodically transmit non-application specific data at regular intervals. Third, AS-MAC is inefficient at broadcasting messages. Because each neighbor has a unique wake-up time, a mote wishing to send information to all of its neighbors must send individual messages to each neighbor. This makes broadcasting more expensive as the density of the network increases.

2.5.3 Crankshaft

The Crankshaft protocol [8] aims to reduce overhearing of neighboring motes in a dense network, which is what its authors believe is the main cause of inefficiency in dense networks. In the Crankshaft protocol, time is divided into frames, which are further divided into slots. Slots are partitioned into two types–broadcast slots and unicast slots. Every mote wakes up for all broadcast slots, and additionally, each mote wakes up for one unicast slot. However, instead of using neighbor state, such as the neighbor table in AS-MAC, Crankshaft motes each know when a receiver's unicast wakeup tone is because each mote is given a wakeup

slot depending on its physical address. For example, a mote with an address of three may be given the third unicast wakeup slot. Each slot type has the same structure, with time allocated for contention, the wakeup tone, and data transfer and packet acknowledgment. Figure 9 illustrates two senders attempting to send a message to a single receiver. This process is very similar to the SCP-MAC and AS-MAC protocols.

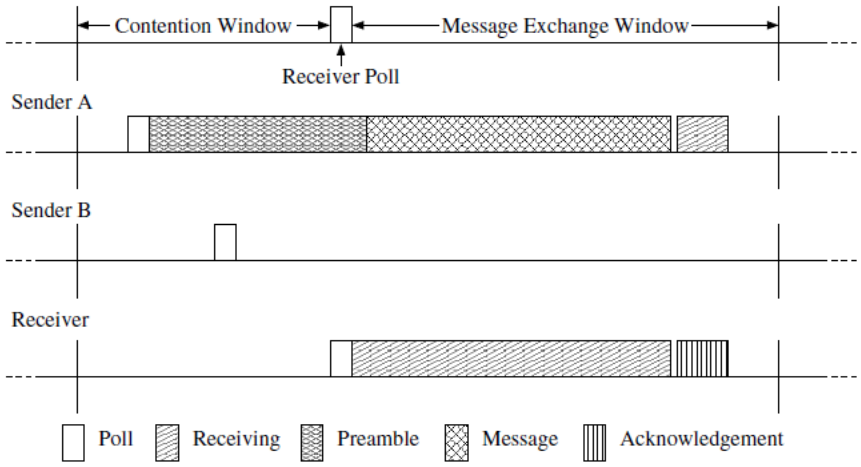

Figure 9: Contention and Message Exchange in Crankshaft [8]

Although AS-MAC and Crankshaft are both MAC protocols that schedule and stagger the wakeup times of receivers, Crankshaft differs from AS-MAC in a number of ways. The most prominent distinction is the notion of broadcast slots. In AS-MAC, there are not broadcast slots, and this makes broadcasting inefficient when using this protocol. Crankshaft, because it is designed for dense sensor networks, has a configurable amount of broadcast slots, which can be set at compile time. The number of unicast slots in Crankshaft is also programmable, and is independent of the number of motes in the network. This means that depending on the number of unicast slots provisioned, it is possible for multiple receivers to wake up during the same slot, which is not possible in AS-MAC. Also, a major difference between Crankshaft and AS-MAC is the absence of a time synchronization mechanism in Crankshaft. AS-MAC uses hello packets to not only introduce new motes into a network, but also to align mote clocks. Crankshaft has no special provisioning for clock alignment, and instead replies on an upper layer to manage this. Lastly, the Crankshaft protocol introduces MAC layer acknowledgments, functionality which AS-MAC and SCP do not have.

2.6 The MAC Layer Architecture Framework

The MAC Layer Architecture [23], or MLA, is a framework to aid in the development of MAC protocols in wireless sensor networks. Rather than creating a number of monolithic MAC protocols to work within the constraints of various applications, MLA aims to provide optimized and reusable components that can be leveraged across many MAC protocols. MLA provides hardware abstractions in its hardware-dependent components, which provide services such as alarming, local time, and radio core access. MLA also includes hardware-independent components which provide services such as channel polling, LPL, preamble sending, time synchronization, slot handlers and low level dispatchers. MAC protocols that implement MLA result in a clean interface, which can easily be swapped in or out of a TinyOS configuration in favor of a different protocol. This makes MLA ideal for testing the same application using different MAC protocols.

The usefulness of MLA is seen in its ability for programmers to write less code in order to implement their MAC protocols, without incurring much overhead. The creators of MLA show that roughly 51% to 73% of MAC code is leveraged using MLA in their implementations of B-MAC, X-MAC, SCP, Pure TDMA, and SS-TDMA, as Figure 10 shows. Additionally, RAM usage with a MLA implementation as compared to its monolithic cousin is within a few bytes, and ROM overhead is under 2KB in most cases.

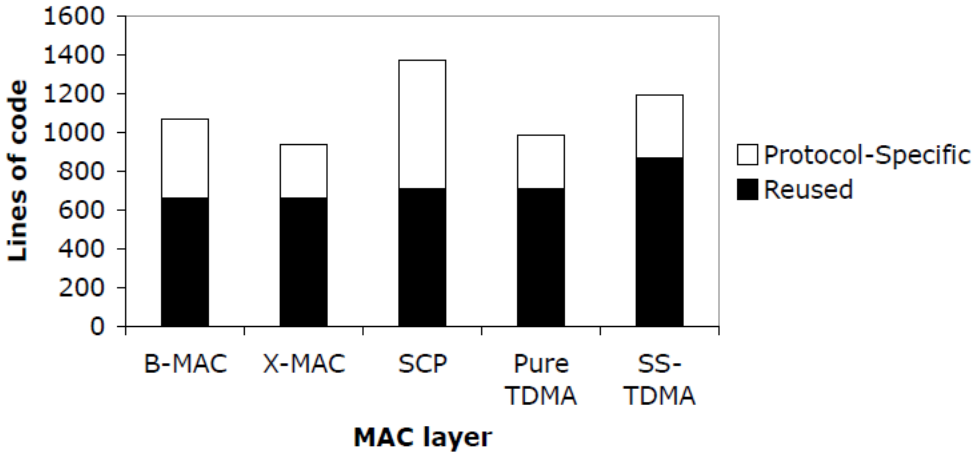

Figure 10: Code Reuse Using MLA [23]

2.7 Power Measurement Techniques

Energy efficiency is a significant concern among several areas of WSN research, including network protocols [19, 33, 35, 39], data gathering and processing [37] and system design [26, 10]. In order to gauge the energy

efficiency of WSN solutions, it is crucial to be able to measure energy consumption.

2.7.1 Mathematical Modeling

A common technique for measuring power consumption of wireless sensor hardware is to employ mathematical models. This tends to be less accurate than other methods, as the interactions between components are often complex. In addition to measurement purposes, mathematical modeling can serve as a useful tool for understanding the intricacies of sensor events. For example, [19] uses proven power measurements of radio operations to model the total energy consumption per second, as shown in equation (1). In this equation, T_{tx}, T_{rx}, T_{lx}, $T_s$and T_{lpl} are percentage times for transmission, reception, listen, sleep and Low-Power-Listening, respectively, and P_{tx}, P_{rx}, P_{lx}, $P_s$and P_{lpl}represent the amount of power drawn in each state.

$$E = T_{tx}P_{tx} + T_{rx}P_{rx} + T_{lx}P_{lx} + T_sP_s + T_{lpl}P_{lpl} \tag{1}$$

Using mathematical models to measure energy consumption is usually a good first step in judging the overall behavior of a protocol, and how and where a protocol expends energy. However, when using a mathematical model to estimate a real-world scenario, it is important to choose parameter values that are possible on physical hardware. Issues such as clock granularity, processor speed, and properties of the radio all put lower limits on timing constraints.

2.7.2 Software Simulation

The Network Simulator (ns-2) [1] is a popular software-based network simulating tool which has been utilized in experiments by the creators of numerous WSN MAC protocols: ZMAC [31], Sift [18], PMAC [34] and RI-MAC [33], among others. This simulator has withstood the test of time, originating as a variant of the REAL network simulator [22] in 1989, and remaining in constant development since then. The Objective Modular Network Test-bed in C++ (OMNET++) [36] is an open-source discrete event simulator which has been utilized in experiments for well-respected WSN MAC protocols such as Crankshaft [8], AI-LMAC [2] and T-MAC [35]. Its object-oriented, modular design is ideal for flexible component reuse.

The simulation community recognizes the value of sharing data. It is very common for researchers to share their work—for example, simulation configurations—with the rest of the community. This is a major benefit of using simulation-based measurements; rarely is it necessary to "re-invent the wheel." Additionally,

many simulators are packaged with useful data interpretation and visualization utilities, which can save time and reduce human error in data processing.

However, simulations are far from perfect. Although some simulation solutions, such as the MiXiM [24] framework for OMNET++, attempt to model some of the environmental factors which affect sensor networks, the complex nature of wireless radio communication cannot be perfectly simulated. Additionally, learning to properly use a network simulator can be very time-consuming; incorrect configuration can easily result in skewed results.

TOSSIM (Tiny Operating System Simulator) [25] is a highly scalable wireless sensor network simulator which faithfully models the behavior of actual sensors. This can partially be attributed to the fact that TOSSIM runs the same code that runs on sensor hardware, with the exception of a few replaced low-level components. It allows researchers to design and execute experiments over wireless sensor networks that would otherwise be expensive and cumbersome.

PowerTOSSIM [32] is an extension to TOSSIM which measures the activity of hardware peripherals during simulation runs. By comparing 60-second runs of 16 TinyOS applications on PowerTOSSIM and a physical sensor device, the average error was found to be 4.7%. While this amount of error may be undesirable in some experiments, the utility of PowerTOSSIM is undeniable. It maintains TOSSIM's ability to perform experiments with large numbers of motes and quickly switch between different network topologies while providing an essential measurement—power consumption.

2.7.3 Hardware Measurements

[28] presents an environment for power measurement which utilizes a digital multimeter to perform run-time power consumption measurements on a mote. In this approach, a current clamp is attached to the mote's power supply line, which produces an output voltage proportional to the current in the line. Output from the digital multimeter is collected and processed by a computer running LabView. The main advantage of this technique over software simulation is accuracy. Additionally, this approach is unobtrusive and overhead-free. However, the hardware required for taking measurements essentially restricts its use to small laboratory sensor networks.

SPOT (Scalable Power Observation Tool) [20], as pictured in Figure 11, addresses the issue of environment-prohibiting equipment requirements by attaching small measurement devices to motes. However, it fails to some solve other problems. First, the issue of gathering energy statistics is suspiciously excluded—no methods of storing or transmitting power readings are explained. Moreover, the measurements taken are somewhat

shallow considering the complexity of the hardware add-on. A seven-minute test of the SPOT device yielded an error rate equal to 3% of the actual energy usage, making it 1.7% more accurate than PowerTOSSIM. Despite addressing some of the weaknesses of more primitive power measurements, the SPOT has room for improvement.

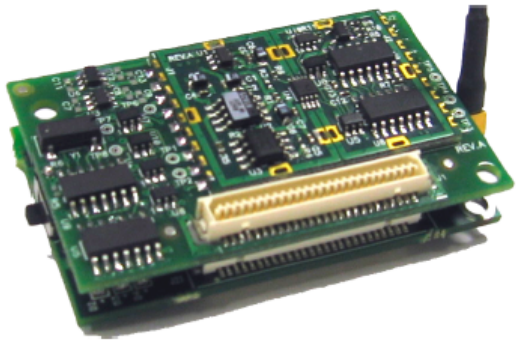

Figure 11: SPOT Module Atop a MicaZ Mote [20]

iCount, pictured in Figure 12 is a hardware energy metering solution which utilizes a switching regulator for real-time measurement of power consumption. As of September 2009, the MAX1724EVKIT switching regulator evaluation kit costs $60 from the manufacturer [12]. The linear relationship between load current and switching frequency of the MAX1724EZK33 makes this integrated circuit an ideal choice for energy measurement. It is also usable in a multitude of deployment environments as a result of its small size.

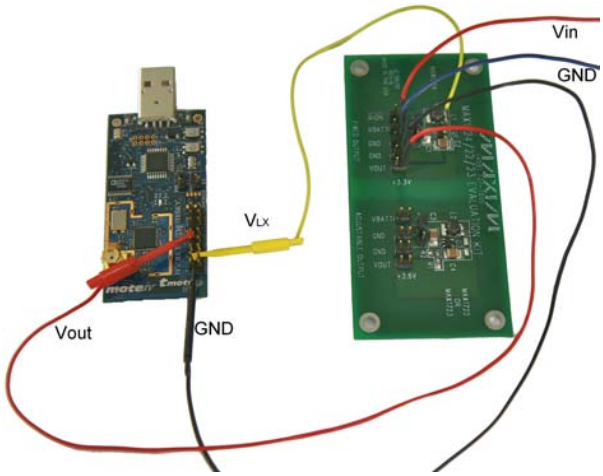

Figure 12: iCount - MAX1724EVKIT (Right) and a TMote Sky Mote [3]

However, the iCount suffers ±20% maximum error over five orders of magnitude (decades) of current draw. Use of this device also requires nontrivial calibration techniques, making it impractical in most situations. Additionally, some sources of overhead exist in iCount:

- Microcontroller-based hardware for the counter contributes a fixed increase to power draw
- The act of counting switches contributes a frequency-dependent increase to power draw due to gates changing state at increasing rates and the resulting charge/discharge cycles of the gate capacitance
- Counter overflows must be periodically serviced by the microcontroller

2.7.4 Quanto

Quanto [5] is a low-overhead software-based energy profiling tool for wireless sensor networks running TinyOS version 2. Quanto tracks hardware usage in real-time on a per-component basis, enabled by modifications to TinyOS device drivers, as portrayed by Figure 13. Optionally, an iCount device may be used in conjunction with Quanto to provide real-time energy readings, in addition to hardware usage statistics. It also supports power measurements over the course of user-defined "activities," which can span over an entire network of motes. For example, Quanto could measure the power consumed in the multi-hop routing of a packet over numerous sensor devices. Each energy sample taken by Quanto is stored in a 12-byte log entry, which can be output to local flash storage or transmitted over the radio.

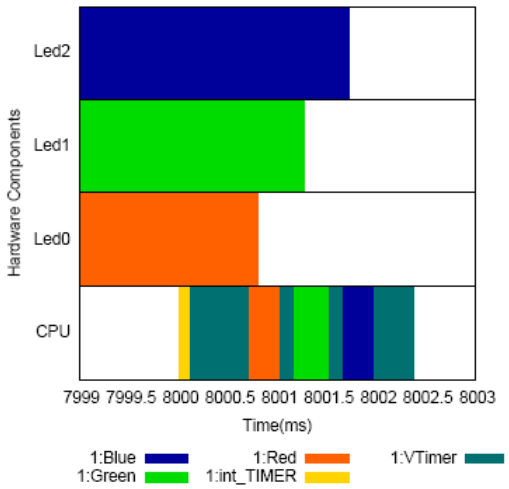

Figure 13: Quanto-generated Visualization of Hardware State Transitions for the Blink Application [5]

Sources of overhead in Quanto include:

- Increased memory footprint caused by 1275 extra lines of code (TinyOS device drivers and measurement/logging functionality).

- Logging of an event to RAM takes 101.7 µs, and the event occupies 12 bytes.
- Data gathered by Quanto needs to be extracted for it to be useful. There are a number of ways to do this with various overhead costs.

While iCount and Quanto are not yet polished enough for widespread use in the research community, they are very promising technologies which bring us closer to the possibility of accurate, low-overhead energy measurement in WSNs.

3 Methodology

The primary goal of this project was to comparatively measure the energy consumption of state of the art wireless sensor network MAC protocols in a physical environment. Our experiments were designed to provide insight into the advantages and disadvantages of the different protocols under local gossip, convergecast, and broadcast network traffic scenarios. We investigated the SCP-MAC, AS-MAC, and Crankshaft protocols, as well as Broadcastable AS-MAC (or BAS-MAC), a protocol that we developed. All protocols were implemented using the MAC Layer Architecture framework, and tests were performed on TelosB motes running the TinyOS 2.1.0 operating system. To measure energy consumption, we created an energy profile through the use of an oscilloscope and modifications to the CC2420 radio drivers. A separate TinyOS component recorded the amount time spent in each radio state during the course of our experiments. The sections below describe the implementations of the four protocols tested, the specifics of experimental methodology, and descriptions of the experiments that were conducted.

3.1 Hardware Selection

Choosing the right sensor mote is an important consideration for any WSN deployment. A number of "off-the-shelf" motes are available, including the Mica2 [14], MicaZ [15], IMote2 [13], and TelosB [16]. Table 2 compares the prices and features of these motes. Because our motes were used for research, rather than a real-world application, the physical size of the mote was not a concern. The ideal mote for our research would be simple to program and have enough memory to store readings from experiments, with a low price. Because of these constraints, we chose the TelosB mote. Besides having a 802.15.4 compliant radio, it supports the TinyOS operating system and contains a USB interface for easy programming. The TPR2420CA model TelosB motes chosen contains a Texas Instruments MSP430 micro-controller running at a speed of 8 MHz, a Chipcon CC2420 radio, 10 Kilobytes of RAM, and 1 Megabyte of flash storage.

Mote	TelosB	Mica2	MicaZ	IMote2
Processor Speed	8 MHz	16 MHz	16 MHz	13-416 MHz
Processor Current Draw	1.8 mA	8 mA	8 mA	31-66 mA†
Flash Memory Size	1 MB	512K	512K	32 MB
Radio	CC2420	CC1000	CC2420	CC2420
Radio Receive Current Draw	23 mA	10 mA	19.7 mA	44 mA
USB Interface	Yes	No*	No*	Yes
Integrated Sensor Board	Yes	No	No	No
Price [11]	$139.00	$125.00	$99.00	$299.00

* Only with a separate interface board

† Upper figure given with a processing speed of 104 MHz, with radio enabled

Table 2: Comparison of Mote Specifications [14, 15, 16, 13]

3.2 MAC Protocol Implementations

Using the MLA framework, we developed four distinct MAC protocols to be tested, which included a versions of AS-MAC, Crankshaft, SCP-MAC, and our own protocol–Broadcastable AS-MAC, or BAS-MAC. SCP-MAC was indirectly implemented as a version of Crankshaft with one broadcast slot, and no unicast slots, similar to [8]. In the sections below, we discuss the implementations of these protocols, as well as the differences between our versions of the protocols and their original descriptions in literature.

3.2.1 AS-MAC

The first protocol re-implemented was AS-MAC, presented in [19]. A complete rework of the code was necessary in order to conform to the MLA framework in which all of our protocols were run.

AS-MAC is broken into three major components–a sending component, a receiving component, and an initialization component, which were wired into the framework provided by MLA. Each component acts independently, and arbitration between the components is handled by a master protocol state. This master state ensures that no two components use the radio simultaneously.

The sending component represents the end-to-end logic of a mote which wishes to send a message. From the application layer, a user requests to send a particular message. Upon this request, the sender determines the next time in which the intended recipient is scheduled to wake up. It sets a timer according to this time and the contention window size. The mote then wakes up to perform contention resolution, and if the channel is free, sends a preamble that lasts the rest of the contention window as well as the length of the receiver's wakeup tone before sending the message.

Although our sending protocol has the ability to send messages during AS-MAC's hello time as well, this feature is not used in our experiments. For simplicity, and to keep the protocols on a more even playing field, we have eliminated hello messages from the protocol. Instead, motes only perform non-hello time wakeups as described in [19].

The receiving component schedules the time at which a mote wakes up and handles all the logic for receiving a message. When a message is received, our component receives an event from the MLA framework. This event is processed to determine if the mote is receiving a preamble, a hello packet, or a data packet. Preambles and hello packets are handled internally by the receive component, while data packets are sent to an upper layer for processing. Like the sending component, although both hello and non-hello time wakeup logic was implemented, only non-hello wakeups are used in our experiments.

The initialization component is responsible for the initialization phase of AS-MAC. During this phase, a mote listens to the channel for any hello messages. When a hello message is received, it is processed, and an entry is made in the mote's neighbor table. After a pre-determined amount of time has passed, which is a settable parameter in our implementation and typically set to the wakeup interval multiplied by the how often hello packets are sent, the initialization component assumes that it has received hello messages from all motes in radio range and begins normal operation of the AS-MAC protocol.

Because of its slow start-up times and our decision to standardize initialization and synchronization across protocols, we implemented a static initialization option which was used in all of our experiments. With static initialization, each mote's neighbor table is pre-populated at compile time. This allows us to control exactly when motes wakeup and ensure that all motes in the network are known to each other.

In order to implement broadcasting in our AS-MAC protocol, an additional layer was wired on top of the sending component. This layer recognizes when the recipient of a message is a broadcast address and sends a unicast message to each neighbor by sequentially going through the neighbor table. Unicast addresses pass directly through this layer.

3.2.2 Crankshaft

The second protocol implemented is a variant of Crankshaft, as described in [8]. In this protocol, time is divided into frames which are further divided into slots. Each frame contains a prescribed number of slots, which are designated as either unicast or broadcast slots. An individual slot is divided into blocks of time for contention, the wakeup tone, data transfer, and packet acknowledgment. Each mote in the network wakes up for every broadcast slot, and a mote wakes up for a particular unicast slot depending on its network address, which is computed via a simple modulo. For example, if there were four unicast slots, nodes with addresses 4, 8, and 12 would all wake up during the first unicast slot, while nodes 6 and 10 would wake up during the third unicast slot. Our implementation of Crankshaft allows the number of unicast slots and broadcast slots to be configured at compile time.

There are a few key differences between our implementation of Crankshaft and its original description in literature. The first difference is the algorithm used to determine the contention slot a mote chooses when it wishes to send a message. The original Crankshaft protocol uses the Sift distribution, which weights senders with an uneven probability distribution, favoring later contention slots. This is used to reduce the probability of two senders choosing the same contention slot in a network with heavy traffic to a particular mote. Our protocol uses a uniform distribution for determining contention slots. However, this is not a significant

issue because our experiments intentionally stagger sending to mitigate collisions. Moreover, the original implementation of Crankshaft on physical motes uses a uniform distribution, similar to our implementation.

The second difference between our protocol and the original version of Crankshaft occurs in the usage of acknowledgments. Although our protocol supports acknowledgments, we decided to disable them throughout our experiments. This is because none of the other protocols studied have inherent support for MAC layer packet acknowledgments. Comparing a protocol that uses acknowledgments to one that does not is difficult, because the lack of an acknowledgment signals a message retransmission that would not occur in a protocol which does not support acknowledgments.

Lastly, the wakeup behavior of our basestation node is different from the original Crankshaft implementation. Originally, Crankshaft was designed to have an optimization in which the basestation in a topology wakes up during all unicast slots, in addition to all broadcast slots. The reasoning behind this is that the basestation node is typically connected to a more permanent power source, and the energy usage of this particular mote is not a concern. However, in our implementation of Crankshaft, the basestation only wakes up for a single unicast slot. This was done because the basestation in our experiments does not represent a WSN basestation in the traditional sense. In our experiments, a node is considered the basestation only because we are in a single-hop topology. For our experiments, we needed to designate special behaviors to a particular mote in order to measure network energy usage in convergecast and broadcast scenarios. This means that our basestation mote is a basestation in name only. In fact, there are many scenarios where messages converge to a mote, or a mote broadcasts to its neighbors. An example of this would be a querying application in which requests are flooded through the network via broadcasts. Because of this, we chose to implement the wakeup behavior of all motes in a similar way.

3.2.3 SCP-MAC

The SCP-MAC protocol used is a variant of our Crankshaft protocol with one broadcast slot and no unicast slots. This means there is one slot per frame, in which all motes periodically wake up in a synchronized fashion.

There are a number of differences between our implementation of the SCP-MAC protocol and its original implementation. One difference between the two protocols is that our protocol does not include the multi-hop enhancement of adaptive channel polling that allows one mote to send multiple messages quickly to another mote. However, the scope of our experiments is single hop messages, and this would add no benefit to SCP-MAC. In fact, the original authors of SCP-MAC do not include this enhancement themselves to

produce their single hop results.

Another difference between our protocol and SCP-MAC is the contention mechanism. In the original version of SCP-MAC, contention is performed in two phases. In the first phase, all senders wishing to send a message contend for a slot in an initial contention window. However, some senders may choose the same slot, and these senders are unaware of the collision at transmission time. Thus, SCP-MAC has the remaining senders contend again to minimize collisions. Our version of SCP-MAC does not contain this optimization of two-phase contention. Although not including this enhancement may seem to give SCP-MAC a disadvantage in relation to its original implementation, the majority of our tests stagger mote communication to avoid contention situations altogether. In this way, the behavior of the contention slots is less important than the number of contention slots, which determines how much energy the protocol will use.

The third difference between these two protocols is the absence of synchronization packets in our version of the SCP-MAC protocol. The original SCP-MAC implementation periodically sends synchronization packets between motes in order to keep clock times within a certain threshold and retain synchronization. Due to the relatively short length of our experiments (in the magnitude of minutes), and the longer amount of time for motes to drift out of sync, we decided not to include synchronization overhead in our version of SCP-MAC, much like we did not include hello packets in our AS-MAC or BAS-MAC implementations.

3.2.4 BAS-MAC

The final protocol that we implemented, Broadcastable AS-MAC, or BAS-MAC, is an extension of the AS-MAC protocol. In this protocol, each mote has a scheduled wakeup interval and offset, similar to AS-MAC. Additionally, each mote is aware of a scheduled broadcast slot and broadcast interval at which all motes are scheduled to wake up simultaneously.

Figure 14 depicts three motes in the BAS-MAC protocol during a non-hello wakeup time. All motes wake up for the scheduled broadcast time, and each mote wakes up in its respective unicast slot. Contention and data transfer are identical to the AS-MAC protocol, both during hello and non-hello wakeup times.

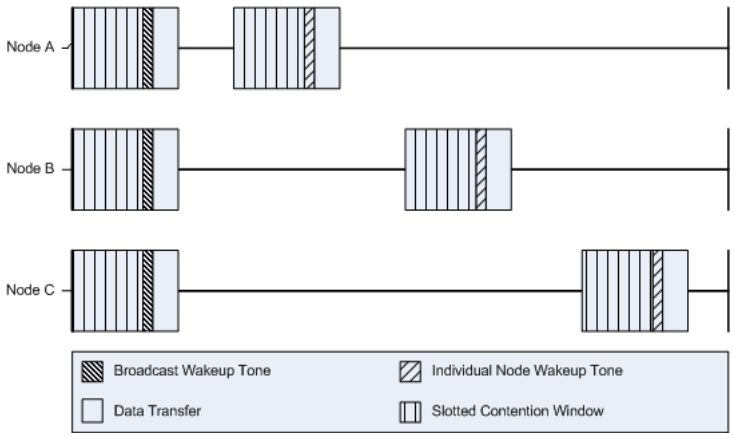

Figure 14: BAS-MAC Scheduled Wakeups in a Three Node Topology

BAS-MAC achieves a synchronized broadcast wakeup tone through a modification of AS-MAC's hello packets. In addition to the information that AS-MAC passes in its hello packets, BAS-MAC also includes two additional parameters: the wakeup interval and the wakeup offset for the broadcast slot. Because AS-MAC hello packets already include the information necessary to synchronize motes with each other, it is easy for a mote to determine the broadcast time after it receives a hello message. Broadcast wakeups are also a factor when determining when a mote should schedule its own wakeup time. When a mote in the initialization phase receives a hello message, it uses all previous individual and broadcast wakeup times to determine when its wakeup time should be, utilizing an algorithm similar to the one employed in AS-MAC.

Although BAS-MAC may appear very similar to Crankshaft, it has some fundamental differences. The major difference between the two is that Crankshaft has fixed frames and slots, where as BAS-MAC, like AS-MAC, uses the concept of intervals and offsets. Although Figure 14 shows a BAS-MAC setup in which all unicast and broadcast tones have the same interval–the length of time between one wakeup and the next–this time does not necessarily have to be uniform as it does in Crankshaft. For example, a broadcast wakeup could be scheduled to happen once after every n individual wakeups, or happen more frequently than an individual wakeup. In Crankshaft, all wakeups fit into a frame, and all frames are identical. Also, BAS-MAC has an inherent means of synchronization–through AS-MAC's hello packets–whereas Crankshaft does not. Crankshaft, however, does include MAC layer acknowledgments, a feature that is not implemented in BAS-MAC.

3.2.5 Verification

Before performing any evaluation of our MAC protocol implementations, we first validated their correctness by reproducing experiments performed by the original authors. We selected experiments directly from the literature in which the protocols were originally presented and recreated them to the best of our abilities (not all parameters were explicitly stated in the research papers, so this involved some amount of inferring on our part). Through this process, we were able to verify that we had faithfully implemented all of our protocols before running our own set of experiments.

The authors of AS-MAC performed a single-hop star topology experiment with two senders and a base station [19]. In this experiment, each sender transmitted 20 50-byte packets to the base station, with 10 seconds in between each transmission. A contention window of size 16 was used, along with a LPL duration of 2.5ms and Hello interval of 60 seconds. Although the authors of AS-MAC used MicaZ motes, they only took energy consumed by the radio into consideration. This means that we could directly compare our reproduction of their experiments, as our TelosB motes also use a CC2420 radio. Figure 15 shows the results obtained by the AS-MAC authors for this experiment. Note the mislabeled y-axis–we can assume that the authors mistakenly used milliwatts (mW) but actually meant joules (J). From this graph, we estimate AS-MAC's energy consumption to be about 0. 22 J. Our reproduction of this experiment yielded a total energy consumption of 0.15J. This decreased energy consumption may be attributed to the fact that we staggered our senders to reduce contention and collisions–it is never specified whether this technique was used in the AS-MAC paper. We feel that this is strong enough evidence to validate our implementation of AS-MAC.

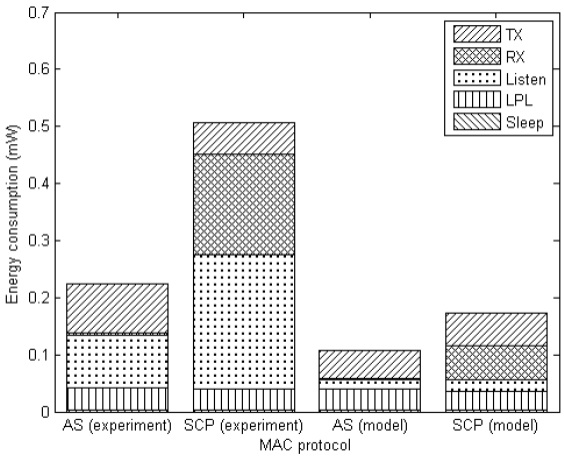

Figure 15: AS-MAC Validation Experiment [19]

The authors of SCP-MAC completed a ten-mote peer topology experiment, in which messages were broad-

cast in 50-second intervals ranging from 50 to 300 seconds. All motes were within hearing distance of each other, and a 1-second wakeup interval was used in conjunction with a 0.3% duty cycle (3ms). Unfortunately, the authors of SCP-MAC used Mica2 motes to perform their experiments. The Mica2 differs significantly from the TelosB, utilizing a CC1000 byte-based radio among other significant hardware differences. Thus our energy readings were not consistent with those presented in SCP-MAC's literature.

Unfortunately we were unable to reproduce any experiments performed by the authors of Crankshaft. This is because their evaluations were performed with the OMNeT++ simulator, using a 96-mote grid topology. Given our limited resources, it was not possible to run this test using our implementation of Crankshaft. However, we are confident that we have faithfully reproduced Crankshaft, having closely inspected its behavior and carefully analyzed its performance in our own experiments.

3.3 Experimental Methodology

The results of our experiments hinge on the ability to accurately measure the amount of energy that a mote uses over the course of an experiment. The amount of energy expended by a mote is dependent on the state of its radio; different radio states consume different amounts of energy per unit time. Our approach to measuring energy was to record the amount of time that a mote spends in each state, and then multiply these time values by a fixed constant that represents the amount of energy used in that state per unit time.

To gather energy data from a mote, we developed a special TinyOS component that hooks into the radio drivers provided by MLA. This component distinguishes the amount of time spent in each radio state. These time values are then multiplied by experimentally measured power constants for each radio state to determine the total energy expended in each state.

We utilize a serial connection between a designated mote and a laptop computer to transfer energy information to a computer on which it can be analyzed and processed. At the conclusion of an experiment, special messages are sent to this mote, and printed over the serial connection. This process is described in greater detail in section 3.4.2.

We decided upon this method of energy recording due to its simplicity and relative accuracy. Connecting motes directly to a device such as an oscilloscope or DAQ card to record energy over the course of an experiment, although potentially the most accurate energy measurement strategy, has a number of key disadvantages. First, in order to retrieve energy readings from all of the motes, each mote must be connected to a separate piece of hardware. If not enough hardware is available, then certain motes must be designated as representatives of the network in terms of energy usage, and a complete picture of energy usage cannot

be gathered. Secondly, a system such as this requires hardware and software on the host machines capable of taking energy samples at a fast enough rate to determine changes in current for the various radio states, as well as recording these samples to a permanent storage location. Changes in the radio states of the motes were observed to occur at the microsecond level, which puts a lower bound on the required sampling rate of accurate recording devices.

Another possible energy measurement strategy would have been to use the Quanto and iCount software and hardware to record energy readings as well as the times in different radio states. Using both Quanto and iCount would allow us to measure when a mote is in a particular state and the energy used during that period of time. Ideally, these technologies could provide an accurate view of overall energy usage over time. However, this strategy was not pursued primarily for two reasons. The first reason was because of software conflicts between Quanto and MLA. Both of these frameworks use modified versions of the CC2420 radio drivers. MLA uses the modified drivers to provide a cleaner interaction with the upper layers of the framework, and to provide a different implementation of clear channel avoidance. Quanto also uses its own modified radio drivers to measure changes in the radio. Therefore, we found it extraordinarily difficult to merge these two frameworks to use a single set of radio drivers. Secondly, the iCount hardware system not only requires Quanto in order to function, but it requires extensive calibration (which was not fully discussed in its conference paper) to accurately measure energy usage.

3.3.1 Radio Recorder

To measure the amount of energy a mote consumes, we developed RadioRecorder, a TinyOS component that keeps track of the amount of time that a mote spends in each radio state. The radio recorder works via slight modifications to the CC2420 drivers which indicate when the radio state has changed. State changes in the radio cause a function to be called in the radio recorder component. This function uses the current time and the last recorded time to determine the time the radio spent in the previous state. It then adds this time to a counter which accumulates the total time in that particular state. Application layer functions are also provided by the radio recorder to clear out radio readings and gather readings for a particular radio state.

3.3.2 Oscilloscope Measurements

In order to determine the amount of energy that a mote consumes in each individual state, we measured the energy usage of a mote running the AS-MAC protocol on an oscilloscope. Because the energy expended in a particular radio state is a function of the hardware and is independent of the MAC protocol being run, it was sufficient to gather all of our oscilloscope readings using just AS-MAC. Before running our experiments, these oscilloscope readings determined the energy expended in each radio state. To measure energy usage, we modified our radio recorder component to output the current radio state on three of the TelosB general-purpose input/output (GPIO) pins. A 10.2 Ω resistor was placed between the positive terminal of the battery of the mote and its positive voltage input line. The three GPIO pins of the mote were connected to digital input lines of the oscilloscope, and the oscilloscope's ground was connected to the mote's digital ground. Two analog probes were also connected to the mote, one on each side of the resistor, with their ground connected to the mote's digital ground. These probes allowed us to measure the voltage drop across the resistor. Through a simple calculation of Ohm's law, we are able to determine the current flowing in the circuit. Since the oscilloscope displayed the digital and analog readings concurrently, current draw was matched to a particular radio state. Figure 16 provides a diagram of this setup, and Figure 17 shows a physical image of the setup.

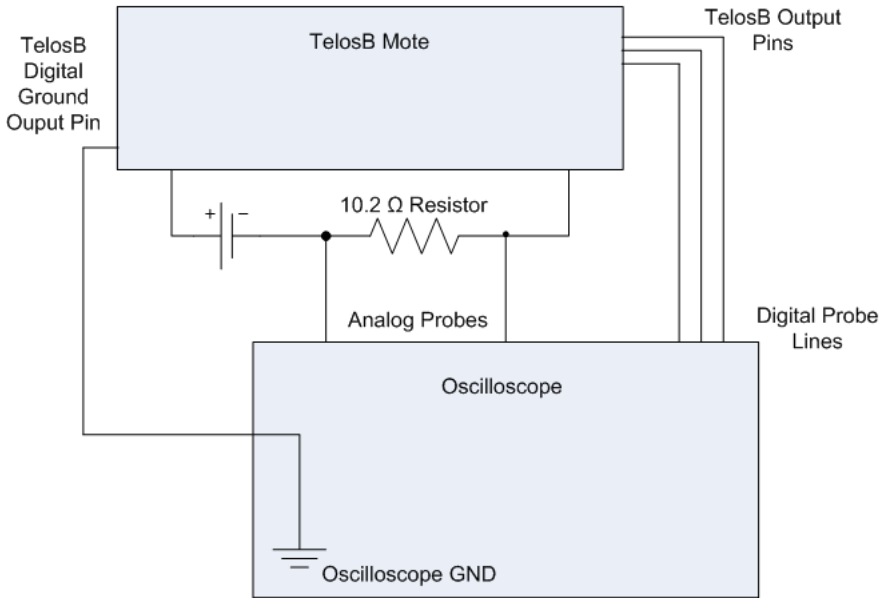

Figure 16: Schematic of the TelosB Mote Connected to Oscilloscope

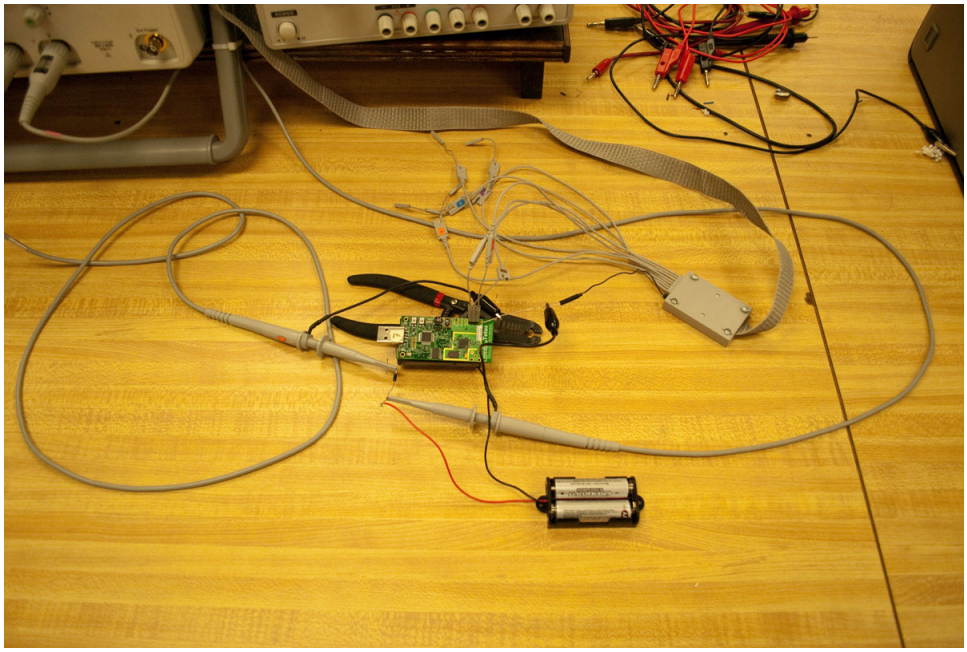

Figure 17: TelosB Mote Connected to Oscilloscope

Figure 18 shows a sample measurement from the oscilloscope, taken during the wakeup tone of AS-MAC. The three lines across the top of the figure, labeled D0, D1, and D2, are digital output signals used to identify a particular radio state. The bottom line represents the voltage difference across the resistor, measured in millivolts (mV). This line is produced by taking the difference between the two analog probes connected to the mote. The horizontal and vertical lines on the figure are used to measure voltage and time, respectively, and their differences can be seen at the bottom of the image. This figure shows that the radio is in the idle state for the wakeup tone for approximately 5.7 milliseconds, with a voltage difference of 187.5 mV.

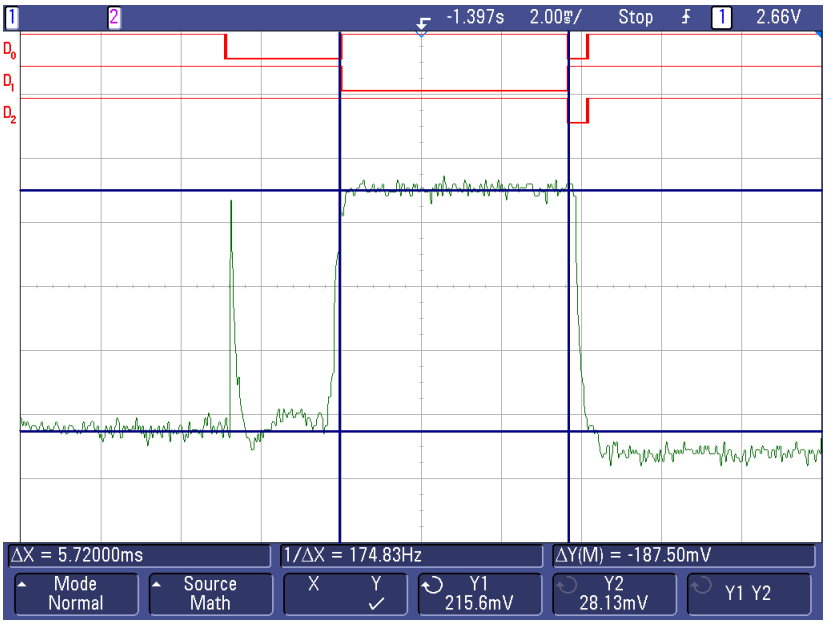

Figure 18: Oscilloscope Snapshot Depicting a Voltage Measurement During an AS-MAC Wakeup Tone.

It is important to note the discrepancy between the voltage differences on the left and right sides of the tone. These differences occur because of current drawn by the processor. On the left side of the tone, as well during the tone, the processor is active, which raises the voltage difference. On the right side of the tone, after the radio has shut down, there is no processing work to be done, which allows the processor to be put to sleep. We chose to include the currents drawn by the microprocessor and the radio in forming our results, because the processor is active during radio operations. The only situation where the radio is on and the processor is idle is during long idle periods, which do not occur in the protocols investigated.

Table 3 summarizes the findings from the oscilloscope readings. Further oscilloscope measurements are shown in Appendix C. Radio usage is divided into six distinct states–Stopped, Starting, Idle, Sending, Receiving, and Stopping. The stopped state refers to times when the radio is inactive. The starting and stopping states occur when the radio is warming up to begin listening or sending, and when the radio is powering down, respectively. In the idle state, the radio is fully powered, but not actively sending or receiving a message. In the receiving state, the radio is actively receiving a message, and in the sending state the radio is actively sending a message. Table 3 also compares our findings with those in the CC2420 datasheet [17]. We found that the idle and receiving states use slightly more energy than shown in the datasheet, which is due to extra energy from the MSP430 microprocessor. Besides confirming the CC2420 datasheet, taking these measurements allowed us to estimate energy spent in the starting and stopping states.

Radio State	Observed Value	CC2420 Datasheet Value [17]
Stopped	0 mA	21 μA
Starting	2.51 mA	N/A
Idle	20.07 mA	18.8 mA
Sending (full transmission power)	17.01 mA	17.4 mA
Receiving	20.22 mA	18.8 mA
Stopping	10.04 mA	N/A

Table 3: TelosB Current and Power Usage in the Various Radio States

3.3.3 Location Selection

In an attempt to find the ideal site for performing our experiments, we compared the signal quality in four different locations on the WPI campus. In each of the four locations, we configured a TelosB mote to broadcast a message every 250ms. A second mote was placed two feet from the broadcasting mote and programmed to receive messages and extract Received Signal Strength Indication (RSSI) readings. On the TelosB, the process of measuring RSSI involves calling the getRssi() command of the CC2420Packet interface. The CC2420 radio chip has built-in RSSI functionality which populates an RSSI register every 8 symbol periods (128 μs). It was empirically found that there exists a difference of -45 between the RSSI register value and the RF input power in dBm [17]. We accounted for this offset in our measurements.

The four locations were:

(1) The third floor of the George C. Gordon Library

(2) The Fossil lab in the basement of Fuller Laboratories

(3) The bleachers of Harrington Auditorium

(4) The grass on the soccer field

As empirically evaluated in [27], RSSI values greater than -80dBm are adequate for achieving 99% packet reception ratios with the CC2420 radio. Our experiments proved that all four locations would be suitable for conducting our experiments, with our outdoor experiment achieving the worst mean RSSI (-67.293dBm). The results of these experiments are displayed in Figure 19. For the sake of convenience, we decided to perform our measurements in the library.

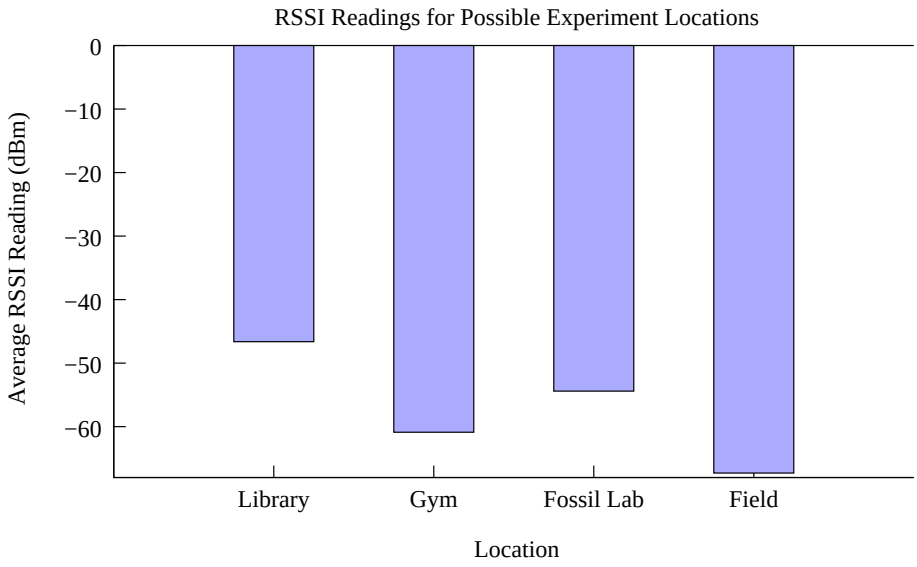

Figure 19: Average RSSI Values at Four Locations on WPI Campus

3.3.4 Initialization Synchronization

An important requirement in all of the protocols tested is the ability to synchronize timing on all of the motes in the network. AS-MAC and BAS-MAC use hello messages and an initialization phase to synchronize motes. However, hello messages are usually sparse, and initialization takes a significant amount of time, especially when a network is built by sequentially introducing motes into it. SCP-MAC also has a mechanism for synchronization, namely falling back on a low-power listening protocol called B-MAC [29] as a bootstrapping phase, as well as the ability to send synchronization packets at given intervals. The Crankshaft protocol has no built-in synchronization, instead relying on an upper layer to ensure that slots and frames line up across motes.

Because of the discrepancies in initialization between protocols, and for simplicity of implementation, we decided to avoid using internal synchronization in any of the protocols. In AS-MAC, hello messages are not sent between motes. Instead, motes are programmed at compile time with a pre-configured neighbor table, which contains the wakeup times and offsets for each mote. A similar approach was taken with BAS-MAC, which includes a pre-configured neighbor table and a pre-configured broadcast offset and interval. Because SCP-MAC and Crankshaft motes do not need to hold any mote-specific data, we simply did not include synchronization packets or a bootstrap phase in our implementations.

In our experiments, we achieve synchronization through a system which starts all motes at the same time. We designed a TinyOS component that wires in to each MAC protocol. This component distinguishes

between two states–the initialization state and the running state. When programmed, motes are initially set to the initialization state. When a mote boots, it checks its current state by reading a value from the onboard flash memory. If its previous state was the initialization state, its new state becomes the running state, and vice versa. It then writes its new state to memory, ensuring that it will persist through a hardware reset. In essence, resetting a mote oscillates it between the initialization state and the running state.

In the initialization state, each mote falls back on a non-existent, or null MAC protocol. This protocol simply keeps the radio on at all times and provides no arbitration of network access. This allows motes to hear any messages sent over the channel, regardless of when they are sent. Additionally, the initialization component listens for button presses on the mote and broadcasts a specific message type when the programmable button is pressed. After sending a message, the sender waits a small, measured amount of time to allow the receivers of the message to process it, before it performs a software reset on itself using the processor's watchdog timer. Receivers of the message also reset themselves upon receipt of the initialization message.

Booting times and flash memory read and write times vary across commodity hardware, and this poses a problem to mote synchronization. For example, it may take one mote longer to read from the flash memory than another, simply because of different electrical characteristics across hardware. To combat this, our system uses a delayed start mechanism. When a mote is reset, its hardware clock is reset very early in the boot process. After the initialization component has finished its reads and writes to flash memory, it sets a timer to wait until a certain, fixed time in the future, normally a few seconds. This fixed time is based off of the hardware clock, and is not affected by discrepancies in hardware. This mechanism allows all motes to begin executing their experiment code simultaneously, regardless of discrepancies in their boot times, or flash reading and writing times. Our initialization system allows us to program all motes in the network, and then begin our experiments via a button press on any one of the motes, all while maintaining clock synchronization within one to two milliseconds.

3.4 Experimental Setup

We performed a series of experiments to thoroughly and fairly evaluate the energy consumption of AS-MAC, BAS-MAC, Crankshaft and SCP-MAC. Each experiment was performed five times to increase the statistical significance, and the results were averaged over the five runs. We varied network size, topology and offered load to ensure the robustness of our assessment.

Each of our experiments followed a similar procedure. First, all motes were connected to a laptop

computer via a ten port USB hub, as shown in Figure 20. A special mote programming script was created that allowed us to simultaneously program each mote with a particular experiment and MAC protocol, which can be seen in Appendix A. The MLA framework allowed us to create one application per experiment, and simply swap in the different MAC protocols.

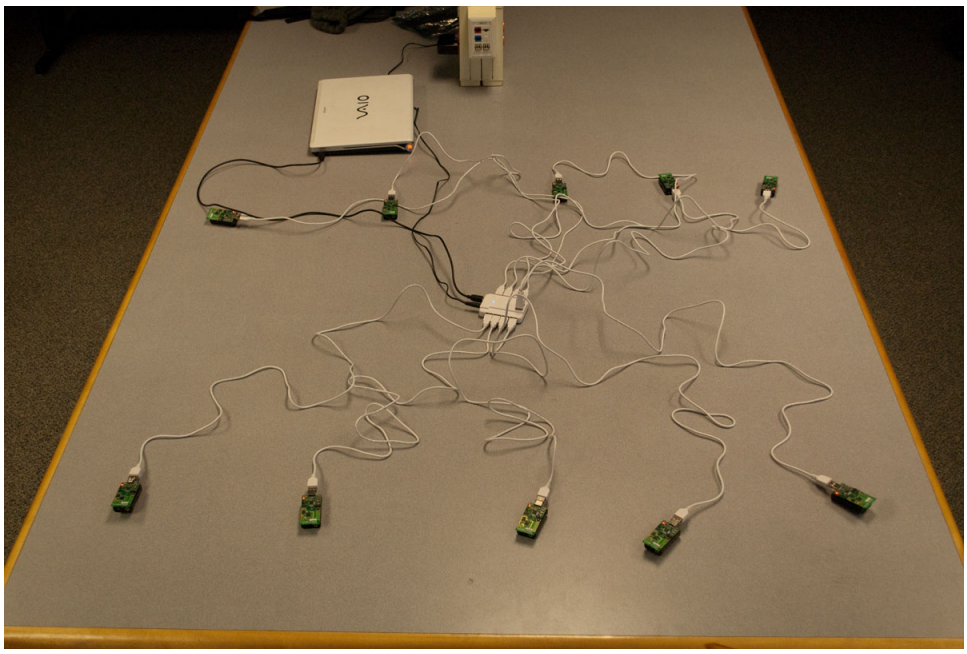

Figure 20: Programming the Motes Before an Experiment

After the motes were programmed, all motes not currently in the initialization state of initialization synchronization were placed into that state via a hardware reset. A serial connection was then established between the laptop and one mote. This mote served as the basestation in our broadcast and convergecast experiments, and the first receiver in our local gossip experiments.

To begin an experiment, the user button was pressed on a single mote, causing them all to reset out of the initialization mode. After a short delay, the radio recorder readings for each mote were reset, and the motes began to send and receive messages according to the desired behavior of the experiment. After the duration of the experiment, all participants sent their radio state information to the mote connected to the laptop computer, as outlined in section 3.4.2. This information contained statistics about the number of messages that node sent and received, including unsuccessful sending attempts.

Three parameters remained constant throughout all experiments–the contention window contained 16 slots, each of 5ms, and the wakeup tone lasted 5ms. Additionally, all experiments lasted for 3 minutes. The remainder of this section discusses the justifications for our choices of MAC parameters, network topologies and the setups of our individual experiments.

3.4.1 Parameter Selection

Each of the four protocols evaluated have unique parameters which affect their behavior, and different parameter settings can yield significantly different energy consumption. With this in mind, we attempted to select parameters which would allow all protocols to be evaluated in the most unbiased way possible. There are several parameters which are common among the protocols. For example, AS-MAC, BAS-MAC, SCP-MAC and Crankshaft all use slotted contention windows. One of the most difficult aspects of comparing four different MAC protocols is configuring their behavior and parameters in order to test them fairly. This is especially difficult because different MAC protocols are designed with different network topologies and tasks in mind. For example, Crankshaft was developed for dense networks, and is able to outperform other protocols, such as SCP-MAC, when aggregate data over the network increases. In order to compare AS-MAC, SCP-MAC, Crankshaft, and BAS-MAC as fairly as possible, we attempted to standardize wakeup intervals, tone and contention check lengths, contention window size and behavior, and initialization.

One concept shared across all of the four protocols is the notion of a wakeup interval. The wakeup interval of a protocol as the amount of time taken to complete one repeating cycle of the protocol. In AS-MAC, the wakeup interval is the amount of time it takes for all motes to wake up one time. In SCP-MAC, the wakeup interval is the time between two scheduled wakeup tones. In Crankshaft, the wakeup interval is a single frame. In BAS-MAC, we defined the wakeup interval to be the length of time for all of the motes to wakeup (as with AS-MAC), in addition to the amount of time required for a broadcast wakeup. To ensure standardization, we fixed the wakeup interval across all protocols, as depicted in Figure 21. This means that in SCP-MAC, there is one scheduled wakeup per interval. For the other protocols, we vary the number of unicast slots depending on the network size. For example, if there are four motes involved in a particular experiment, AS-MAC will have four unique unicast wakeups (one for each mote), while Crankshaft and BAS-MAC will have four unique wakeup slots in addition to one universal broadcast slot. If the experiment size is changed to ten motes, these three protocols will all contain ten unique unicast slots.

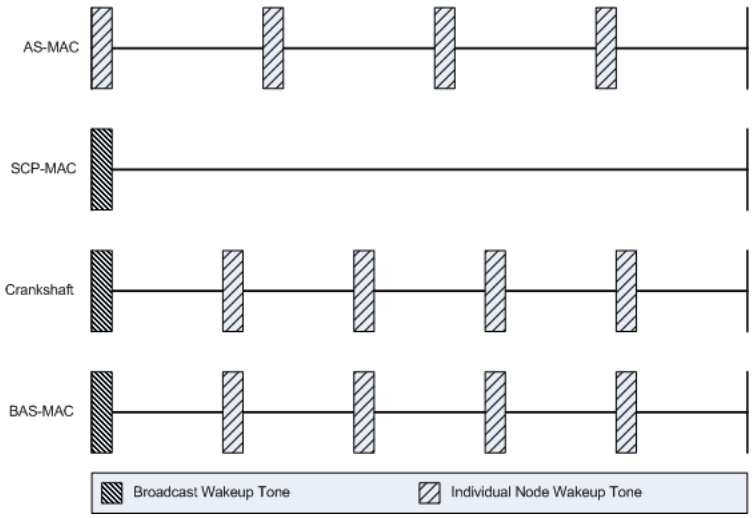

Figure 21: Wakeup Intervals Across AS-MAC, SCP-MAC, Crankshaft, and BAS-MAC Protocols

Because each protocol uses a different method of synchronization, we decided to run each protocol in a manner that ignores the overhead caused by synchronization. By implementing SCP-MAC as a variant of Crankshaft, we avoided the synchronization packets and bootstrapping phase that SCP-MAC normally uses. In AS-MAC and BAS-MAC, hello wakeup times were removed, and these protocols used a predetermined, static neighbor table to control mote wakeup times. Since Crankshaft is the only one of the four protocols to utilize acknowledgment packets, which introduce energy consumption overhead, we disabled these for the experiments.

Because the primary focus of this project is to compare energy usage across protocols, it was important to standardize wakeup tone length and contention slot size. For all protocols, we defined a minimum amount of time needed to sense the channel for activity. Through our oscilloscope readings, clock drift over the length of our experiments, and our implementation for mote synchronization, we decided on a time of 5ms. This time was used in each protocol, for both the wakeup tone length and the contention window slot size.

All of the protocols tested rely on a form of sender contention resolution to determine which sender should be allowed to send a data message to a particular receiver. Thus, we decided to standardize the length and behavior of sender contention. In each protocol, contention slots are picked using a uniform distribution, and all protocols use the same number of contention slots. Although this decision may seem to hurt the SCP-MAC protocol, which uses two-phase contention, our experiments stagger sending motes to reduce contention. Our aim is to measure the energy usage of a mote, which is, in part, a function of the amount of time a mote stays in contention for, and not as much the particular behavior of contention. Given that our largest experiment utilizes 10 motes, it was decided that a reasonable contention window would contain 16 slots, each 5ms in length (creating a total contention window size of 80ms).

These careful considerations ensured that all protocols were evaluated fairly and without bias. However, it should be noted that SCP-MAC is naturally put at a disadvantage through the use of the CC2420 packet-based radio. [39] describes an optimization of SCP-MAC in which the destination field in the header of each incoming packet is inspected upon each reception. If it is determined that the packet is addressed to another recipient, the unintended receiver goes back to sleep, which can significantly reduce overhearing. This optimization can only be utilized with a byte-based radio, such as the CC1000, as the header of a packet can be inspected on the fly, before the entire packet has been received.

3.4.2 Data Collection

After an experiment, each mote holds its own RadioRecorder results in memory. We decided the most practical way to extract this data from each individual mote would be to transmit all results over the radio to a single mote connected to a computer. Since the experiment ends at the same time on each mote, if they all attempted to send their results immediately following the conclusion of the experiment, they would be contenting for the radio channel. To avoid this contention, the transmission of results is staggered based on the identification number of the mote. Namely, each mote waits a fixed amount of time after the experiment to send its results back to the base station. The amount of time to wait is TOS_NODE_ID * 5 seconds, where TOS_NODE_ID ranges from one to ten in the largest experiment. Once all results have been collected by the base station, they are output to a text file for future processing.

3.4.3 Local Gossip

To evaluate the energy consumed by the four MAC protocols in local gossip situations, we considered a large and small pairwise topology, with fast and slow senders. The small pairwise topology consisted of four motes, and the large topology had ten. Half of the motes were designated to be senders, while the other half were designated receivers. In the two-pair experiments, the network wakeup interval was set to 1000ms (each mote woke up to receive once every 1000ms), and in the five-pair experiments, the wakeup interval was 2000ms. The fast senders in both the large and small experiments sent a 50-byte packet every 2000ms, and the slow senders sent one every 10000ms. These parameters are summarized in Table 4.

	Wakeup Interval	Send Frequency
Two Pairs, Slow	1000ms	10000ms
Two Pairs, Fast	1000ms	2000ms
Five Pairs, Slow	2000ms	10000ms
Five Pairs, Fast	2000ms	2000ms

Table 4: Local Gossip Experiment Parameters

3.4.4 Convergecast

We evaluated a large and small convergecast network setup–one consisting of two senders, and the other with nine senders. In both of these experiments, one base station was present, and the other participating motes transmitted exclusively to it. The network wakeup interval was set to 1000ms for all convergecast experiments. In the two-sender setup, fast senders sent once every 2000ms, and slow senders sent once every 10000ms. In the nine-sender setup, fast sender sent once every 10000ms, and slow senders sent once every 20000ms. These parameters are summarized in Table 5.

	Wakeup Interval	Send Frequency
Two Senders, Slow	1000ms	10000ms
Two Senders, Fast	1000ms	2000ms
Nine Senders, Slow	1000ms	20000ms
Nine Senders, Fast	1000ms	10000ms

Table 5: Convergecast Experiment Parameters

3.4.5 Broadcast

To evaluate the performance of the four protocols in broadcasting scenarios, we created experiments in which a single base station broadcasts periodically to multiple receivers. As in the previous experiments, we used one small network setup with two receivers and a large one with nine receivers. The wakeup interval was kept constant at 1000ms for all experiments, and in both the large and small setups, the fast and slow send frequencies were 2000ms and 10000ms, respectively. These parameters are summarized in Table 6.

	Wakeup Interval	Send Frequency
Two Receivers, Slow	1000ms	10000ms
Two Receivers, Fast	1000ms	2000ms
Nine Receivers, Slow	1000ms	10000ms
Nine Receivers, Fast	1000ms	2000ms

Table 6: Broadcast Experiment Parameters

4 Results

The sections below analyze and deconstruct the energy usage results for the local gossip, convergecast, and broadcast experiments. The results of each experiment are partitioned into two types. For the local gossip experiments, results are partitioned according to a mote's role as a sender or receiver. In the convergecast and broadcast experiments, the results are partitioned such that the central node's results (called the basestation) are shown separately from the other nodes in the experiment (called the leaf nodes). Each of the figures below represent the energy consumed (in milli-Joules) averaged across all nodes of the represented type. Total energy consumption is depicted as a stacked bar graph energy consumed in each radio state.

It is important to note that bars are broken purely across radio states and not radio functions. For example, while a mote is sending a message, it spends some initial time in the idle radio state and oscillates between the idle and sending states while sending preambles. This leads to both increased idle state times (and hence energy usage) and sending state times as the mote sends more messages. A similar phenomenon occurs while receiving. Between receptions of preambles, a receiving mote spends a portion of time in the idle state.

In our analysis, we found that the variance between experiments was very low. This is because random numbers generated by a mote, such as when choosing a contention slot, are seeded by the address of the mote, which remains constant across trials. This means that all experimental variation across trials came from how the wireless signals propagated through the air, which was very consistent due to the small distance between motes. The other source of variation in our experiments was across different senders in a single experiment. Because senders choose contention slots randomly, some senders may send longer preambles, on average, than other senders. However, because many messages were sent over a single experiment, and contention slot choices followed a uniform distribution, this variance was also observed to be small.

Our results are divided into sections according to the experiment type (local gossip, convergecast, and broadcast) and presented below. For each experiment type, a few graphs are presented to show the general behavior of each protocol. Supporting graphs are then presented to show and analyze any unique or interesting behavior in a specific situation.

4.1 Local Gossip

Figures 22 and 23 depict the energy usage for the local gossip experiment involving five pairs of motes each sending a staggered messages every 10 seconds. In this scenario, the AS-MAC protocol uses the least energy,

with senders consuming roughly 110 mJ of energy, and receivers consuming roughly 65 mJ. The SCP-MAC protocol consumes roughly 2.5 times as much energy for receivers and 1.75 as much energy for senders, with almost all of the extra consumption due to extra time in the receiving state, both for senders and receivers.

The energy consumption of Crankshaft and BAS-MAC was very similar for both senders and receivers, which is expected since both protocols contain a single broadcast slot and a unicast slot for each receiver. It can observed that the extra energy consumed by Crankshaft and BAS-MAC, as compared to AS-MAC, is time spent in the idle state. This can be easily explained by the extra wakeup tone for broadcasting during each interval. However, it is interesting to see that although these two protocols wakeup twice as many times over the course of this experiment, SCP-MAC still consumes more energy, due to its overhearing of messages.

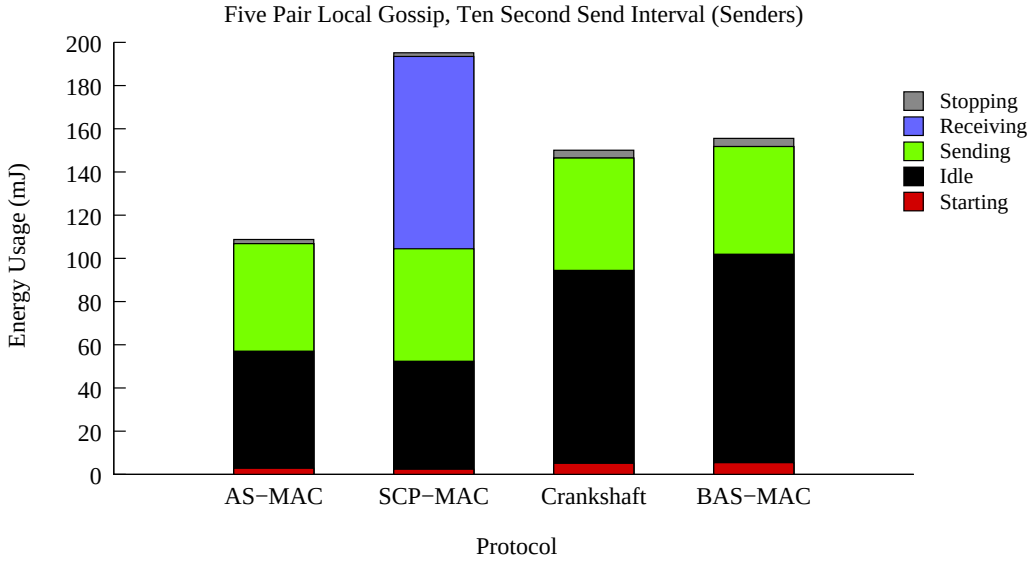

Figure 22: Slow, Large Local Gossip Senders

It is interesting to note that, on average, sending a message consumes more energy than receiving a message. This is because although sending is less costly per unit time, a sender is in the sending state longer than a receiver is in the receiving state due to time spent sending preambles in the contention window.

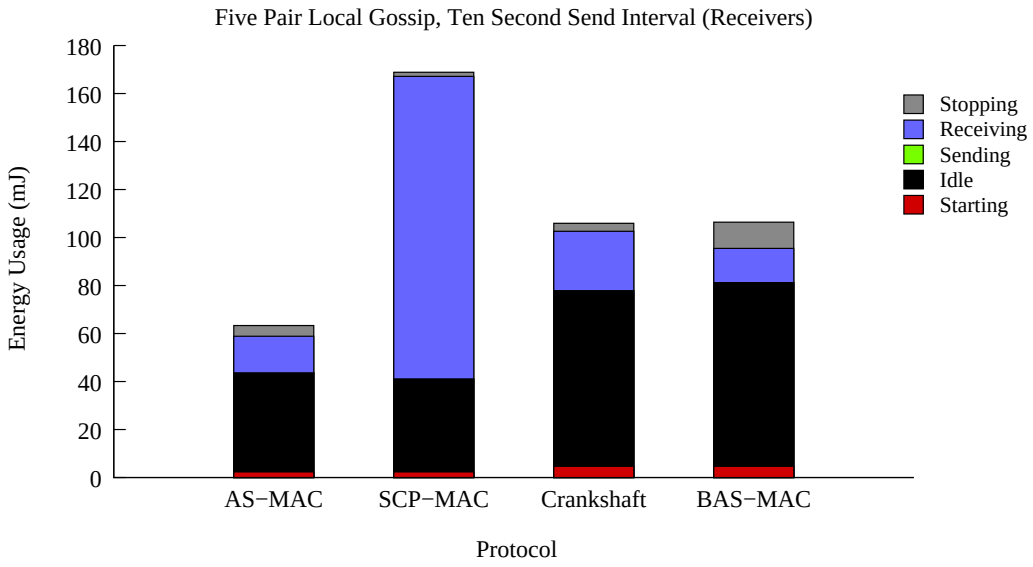

Figure 23: Slow, Large Local Gossip Receivers

Interesting behavior also occurs in the SCP-MAC protocol when the sending frequency for the motes increases. Figures 24 and 25 show energy consumption for senders and receivers with two second send intervals per sending mote, respectively. Upon initial inspection, it would appear SCP-MAC receivers perform better with this faster send interval.

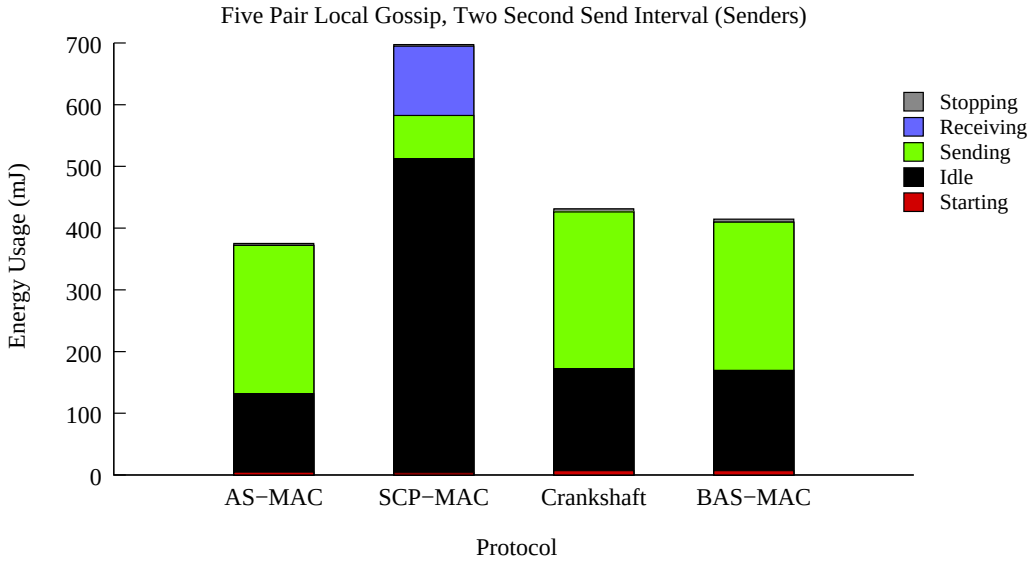

Figure 24: Fast, Large Local Gossip Senders

However, the SCP-MAC protocol only sends about 20% of the total messages requested by the application layer over the course of the experiment. SCP-MAC senders spend much more time in the idle state. This is most likely due to motes waking up to perform contention, but finding the channel busy. Similarly, in

Figure 25 it appears that the energy usage of SCP-MAC is close to that of AS-MAC, when it would be expected that SCP-MAC would use more energy due to overhearing. This occurs because the receivers in the SCP-MAC experiment are receiving less messages than the other protocols, due to a wakeup time that is oversaturated by sender messages. Because some messages are being left unsent, less messages are overheard and the SCP-MAC receivers use less energy overhearing. This oversaturation happens because there are not enough wakeup slots to handle the sending of all the requests made by the application layer, effectively creating an queue of messages still left waiting to be sent when the experiment has completed.

It can also be observed that as the message speed increases, the gap in energy usage between AS-MAC, Crankshaft and BAS-MAC narrows. This is because it takes more energy to actively send or receive a message, compared to idle listening during an empty channel. The unutilized broadcast slots in Crankshaft and BAS-MAC use the same amount of energy regardless of the frequency of unicast messages sent over the network. Therefore, as more messages are sent per unit time, the proportion of energy used for broadcast wakeups shrinks.

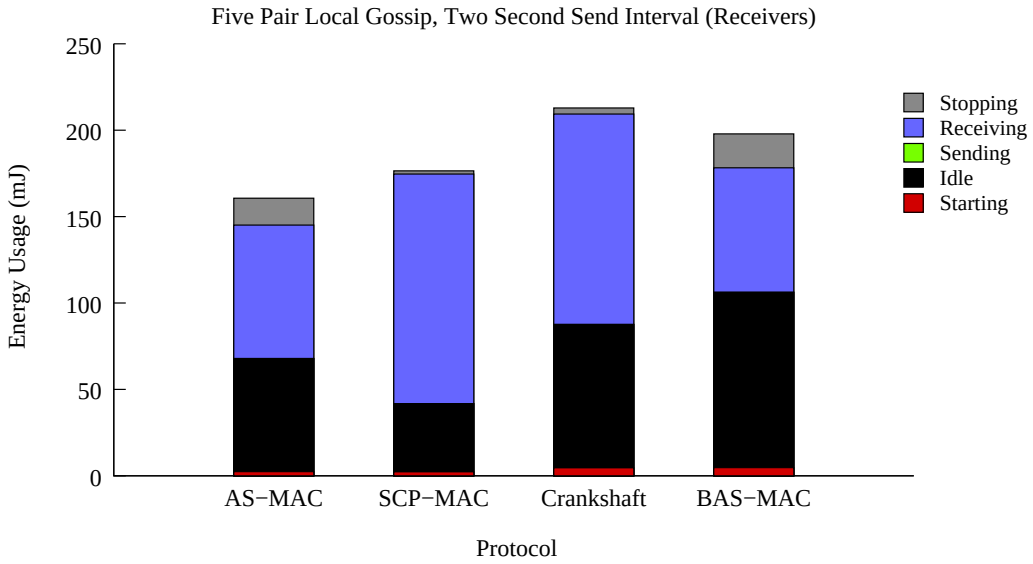

Figure 25: Fast, Large Local Gossip Receivers

4.2 Convergecast

Figures 26 and 27 depict the energy usage for our convergecast experiment involving nine motes each sending staggered messages to a central node every 20 seconds. The slow frequency of sending, in addition to staggering sending eliminate collisions in this set of experiments. In this scenario, AS-MAC exhibits the best performance overall. In a similar fashion to the local gossip scenario, nearly all of the difference in energy usage between AS-MAC and SCP-MAC is due to overhearing, with roughly as much energy consumed by a

sender due to overhearing as through all other radio activity. However, even with the significant amount of energy consumed due to overhearing, the performance of SCP-MAC is comparable to that of Crankshaft and BAS-MAC for senders, and better for the central node. This is because of the two wakeup times per wakeup interval in these two protocols, and the absence of overhearing in the central node due to all messages being intended for it.

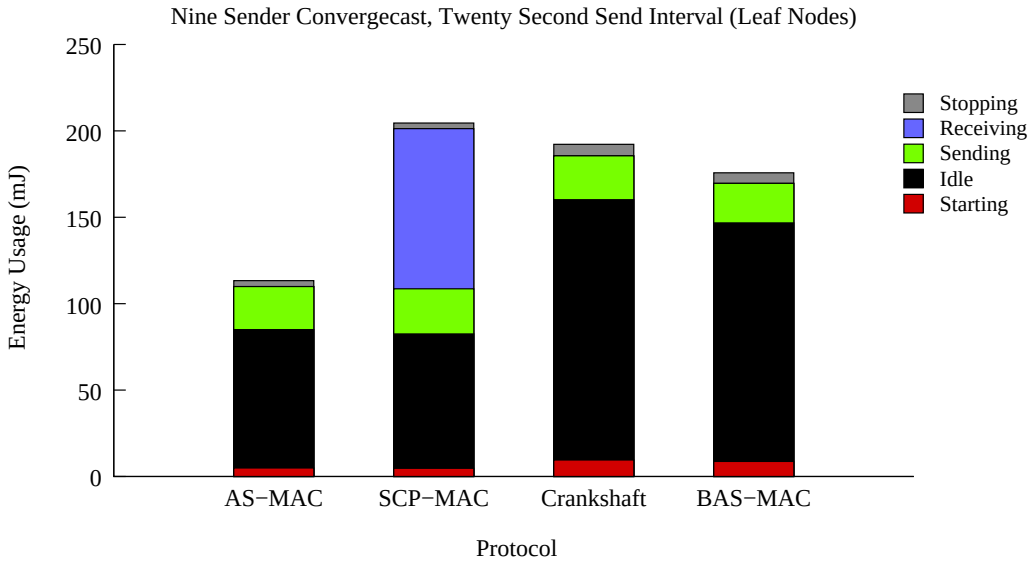

Figure 26: Slow, Large Convergecast Senders

In analyzing the overhearing energy usage for SCP-MAC over the different topology sizes and sending rates tested, there is a clear relationship between the network density (both in terms of the number of nodes in the network and the frequency of messages) and the relative energy efficiency of SCP-MAC. As the network becomes more dense, the energy efficiency of SCP-MAC suffers, a phenomenon not seen with the other protocols. This effect can be seen in Figure 28. This figure shows the average energy consumed by the leaf nodes with a send interval of ten seconds. As the size of the network increases from two to nine senders, the amount of energy expended overhearing increases as well. In fact, in the nine sender experiment, more energy is expended overhearing than by all of the other radio operations combined.

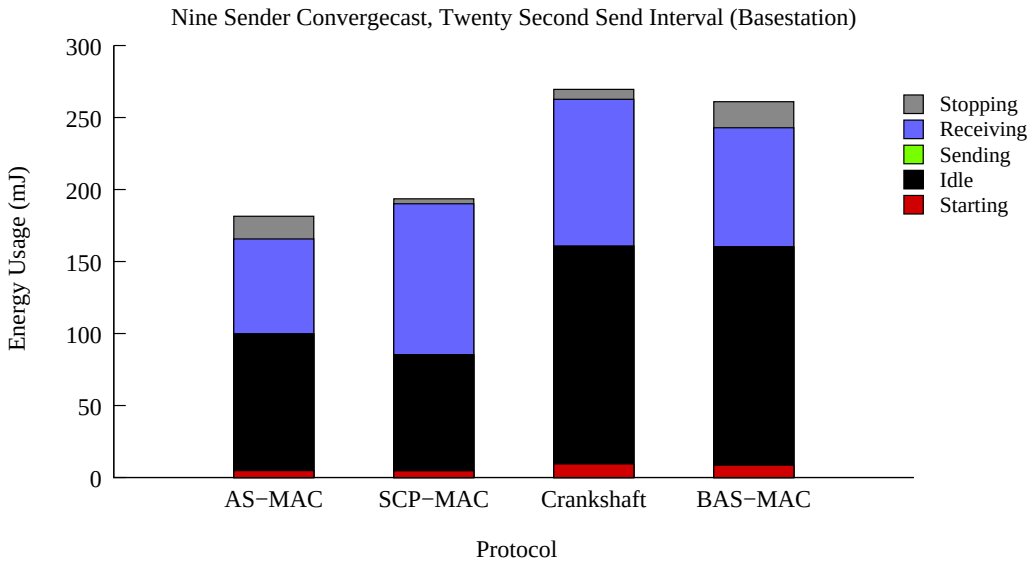

Figure 27: Slow, Large Convergecast Receiver

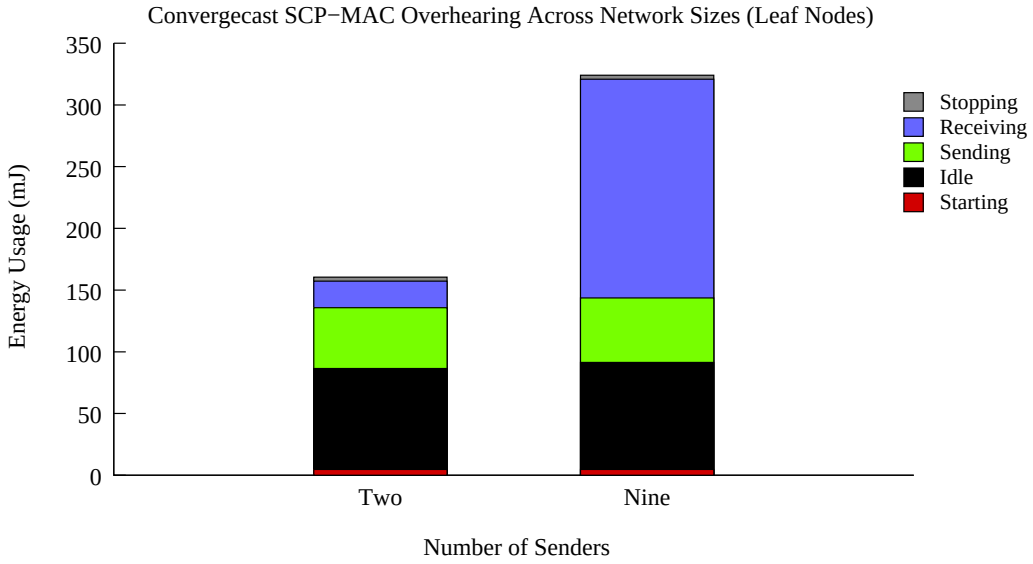

Figure 28: SCP-MAC Convergecast Overhearing with a Ten Second Send Interval

4.3 Broadcast

Energy consumption for broadcast traffic is displayed in Figures 29 and 30, for sending and receiving. These experiments utilize a star topology in which a single base station broadcasts to nine leaf nodes. The graphs displayed are for the slow sending interval (10 seconds), and the fast interval graphs can be viewed in Appendix B. Figure 29 highlights AS-MAC's inefficient broadcasting mechanism and portrays how by simply adding a single broadcast slot into each wakeup interval of AS-MAC makes a significant difference, reflected by BAS-MAC's performance.

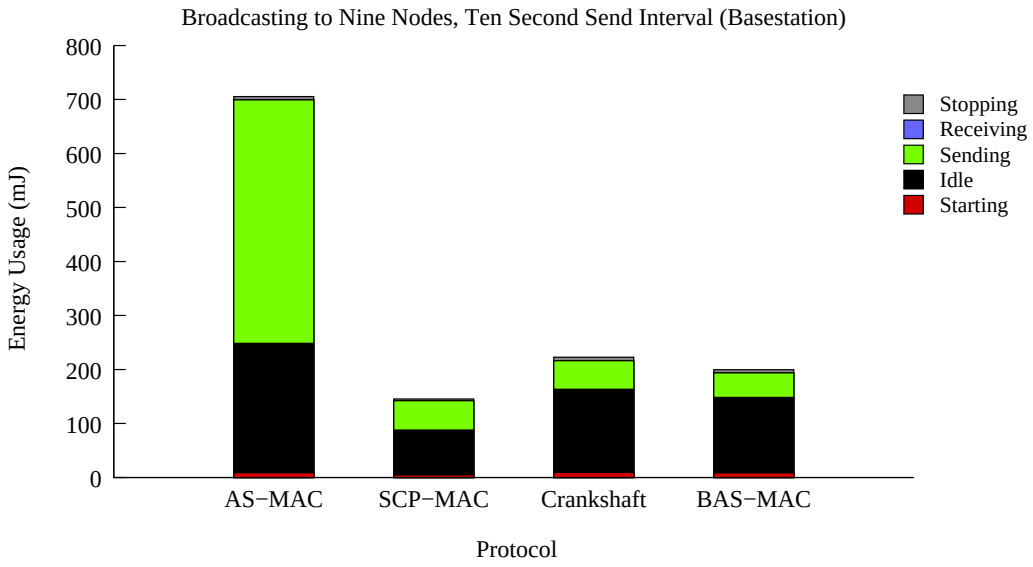

Figure 29: Slow, Large Broadcast Sender

The broadcasting scenario is where SCP-MAC is the most effective. Since all nodes wake up simultaneously in SCP-MAC, it naturally performs very well with broadcast traffic. Receivers of broadcast traffic perform similarly in AS-MAC and SCP-MAC. This is because each of these protocols have a single wakeup slot per interval. However, this slot is a broadcast slot in SCP-MAC but a unicast slot in AS-MAC. Because AS-MAC only has unicast slots, this puts a greater burden on the broadcasting mote, causing the greatly increased energy consumption seen in figure 29. Furthermore, there is only one wakeup slot per wakeup interval in SCP-MAC, as opposed to Crankshaft and BAS-MAC, which each have both a broadcast and a unicast slot. This causes SCP-MAC to outperform these two protocols in both sending and receiving. Additionally, this is a case where overhearing is nonexistent, since all messages are intended for all receivers. This means the main disadvantage of SCP-MAC – overhearing – is not present in this scenario.

From these results, it is important to distinguish between a basestation that is connected to a centralized power source, and one that is not. If all broadcasting is done by a mote whose energy usage is not a concern, then AS-MAC would be a good protocol choice, since the extra energy used by AS-MAC occurs from the mote sending the broadcast messages. However, there are other situations in which a non-central mote may broadcast messages, especially in a multi-tiered network topology. For example, queries may be made to all motes using a broadcast flood. In this scenario, each node must broadcast some number of messages. Also, in more complex topologies, motes could be partitioned into clusters, and one mote in each cluster is chosen as a cluster-head. This mote would then have added responsibilities, which could include broadcasting messages to other nodes in its cluster.

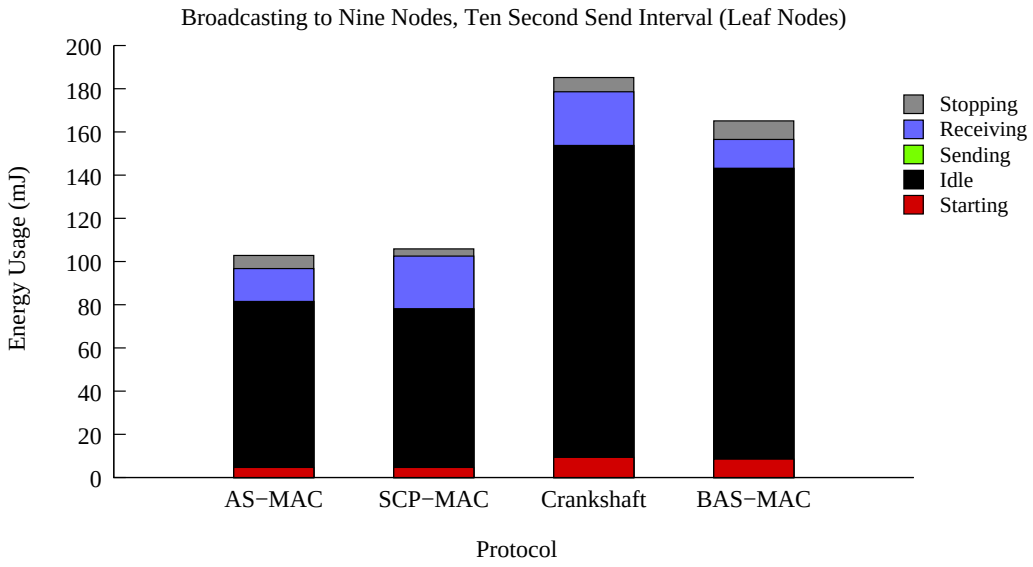

Figure 30: Slow, Large Broadcast Receivers

4.4 Discussion

Overall, we found that no protocol worked best in all scenarios. AS-MAC performed the most effectively in our local gossip and convergecast scenarios, due to a minimal amount of wakeup slots and its ability for overhearing avoidance. However, AS-MAC performed badly in the broadcast scenario, with a large amount of energy required to broadcast a single message. SCP-MAC performed best in the broadcasting scenario, because overhearing was not an issue, and all nodes wake up at the same time. We found that Crankshaft and BAS-MAC had similar performance across scenarios, showing no distinct disadvantage in any particular scenario.

5 Conclusions

The goal of this project was to perform a thorough, balanced analysis of the energy consumption of AS-MAC, Crankshaft and SCP-MAC, three low-power medium access protocols designed for use in wireless sensor networks. During this investigation, we discovered a relatively easily-implemented alteration to AS-MAC to make it more robust in scenarios which utilize broadcast traffic. The result of this was the BAS-MAC protocol.

We began our project by first selecting the physical mote which we would use in our experiments and concluded that the TelosB mote was best suited for our experiments, due to its newer radio and an easy to program USB interface. Next, each of the four MAC protocols was implemented in TinyOS using the MLA framework. The MLA framework was utilized to provide a base of reusable code, as well as the ability to swap different MAC protocols into a single experiment. We then created an energy profile for the TelosB motes through the use of modified radio drivers and an oscilloscope. A software component was created to measure the time a mote spent in each radio state, which combined with our energy profile, yielded an estimate for total energy usage. In order to find a suitable location for our experiments, a number of sites were tested, both indoors and outdoors. Ultimately, the George C. Gordon library was chosen due to it having the best signal strength, as well as the convenience of its location.

Experiments were run to simulate local gossip, convergecast, and broadcast scenarios. In order to maintain consistency across the four MAC protocols, wakeup intervals, wakeup slot lengths, synchronization methods, contention window size, and contention resolution behavior were all standardized. Experimental synchronization was achieved through the use of a special component that allowed all motes to reset simultaneously from a button press on a single mote. Data from our experiments was gathered though a serial connection between a laptop and a designated mote. Experiments were divided into two phases, where the first phase sent messages between nodes and measured their energy, and the second phase allowed all nodes to send their energy readings to the mote with the persistent serial connection. Our evaluations show that in single-hop networks with relatively large send intervals and staggered message sending, AS-MAC is best in the local gossip and convergecast scenarios, while SCP-MAC is best overall in the broadcast scenario. We conjecture that Crankshaft would perform best in extremely dense hybrid (unicast and broadcast) network topologies, especially those which broadcast frequently. Finally, BAS-MAC would be optimal in networks which utilize hybrid traffic with infrequent broadcasts, and where broadcasting is performed by motes that do not have an unlimited power source.

6 Future Work

Our empirical energy analysis of four WSN MAC protocols is a unique contribution to the research community. However, this vast and rapidly-evolving sector of computer science research has many additional areas to be explored. This section discusses ideas for further research in energy-efficient medium access control protocols for wireless sensor networks.

6.1 Multi Hop

As our experiments only investigated single-hop topologies, one area of further study would be to extend our energy measurement techniques to more complex topologies in which not all motes have a direct line of communication. This includes variants of the line and grid topologies discussed earlier, as well as a random topology in which motes are randomly scattered over a large area. We were able to partition motes in the network into two distinct types and average their energy results because all motes were within radio distance of each other and exhibited very similar behavior. However, in a multi hop topology, aggregating and analyzing energy usage measurements and their causes becomes more complex.

One must be very careful to include enough information to distinguish cause and effect when dealing with more complex topologies. One of the major benefits of investigating single-hop topologies is the ability to more easily determine the interaction between motes in the network. As network topologies become more complex, the experiment designer needs to address issues such as the hidden terminal problem, as well as decide on a suitable routing strategy. Also, in some topologies certain motes may act as "choke points" where many sending motes compete for a chance to send to these particular motes. This can be seen in a more complex topology where motes need to route information to a central node. Motes closer to the center of this topology must relay more messages than those at the periphery. This means that simply averaging the network's energy usage does not accurately model the underlying energy behavior in the system.

6.2 Parameter Adjustments

There are numerous parameters in the four MAC protocols studied, each of which has the ability to effect the energy consumption of the protocol. This project used a very specific set of parameters, and held many parameters constant, such as the contention window size, the wakeup interval, and the wakeup time. Although we found this necessary in order to put the different protocols on a level playing field, as well as give

our project a reasonable scope, further work could be designed to locate optimal values for these parameters in certain circumstances. Furthermore, the AS-MAC and BAS-MAC protocols, since they hold neighbor information tables, have the ability to schedule motes with different wakeup intervals. One example of this would be to schedule the basestation node to wake up more often than the sender nodes in a convergecast scenario. Another example would be to make the broadcast slot in BAS-MAC occur more or less often than the unicast slots. In Crankshaft, the number of broadcast and unicast slots can be modified in an effort to selectively control the amount of nodes that share a slot, which would indirectly control the amount of overhearing in the network.

One interesting parameter to investigate is the size of the contention window, in particular, the investigation of the relationship between the number of single-hop neighbors and how the number of contention window slots affects energy usage and the proportion of successfully sent messages. The goal of this study would be to determine if there is an optimal number of contention slots, given a particular number of single-hop neighbors. In the protocols investigated, the majority of energy used for sending is due to preambles sent to occupy contention slots. Reducing the number of contention slots reduces energy usage, but increases the number of failed messages as traffic density increases.

To see why this occurs, consider a network with many senders and few contention slots. A message failure occurs when a message is garbled due to multiple senders transmitting symbols over the shared medium (in this case, the wireless channel) at the same time. With only a few contention slots, if a number of motes wish to send a message at the same time (and to the same mote in a protocol such as AS-MAC) there is a greater probability that two or more motes will wake up on for the first unused contention slot. These motes will sense the channel and all assume that it is free, causing them to simultaneously send their message. This behavior creates a trade-off between packet reception (or retransmissions) and the number of contention window slots used.

6.3 Latency and Throughput

Although energy is often the most important concern in wireless sensor networks, it is usually not the only one. Certain applications require sufficient levels of latency and throughput, which we did not investigate in our research. For example, a WSN-based security system requires extremely low latency to be effective. A study which evaluates latency and throughput of multiple MAC protocols, similar to our energy consumption evaluation, would provide a more robust profile of the trade-offs which exist among the various WSN MAC protocols. Although the original pieces of literature in which these protocols are presented visit the issues of

latency and throughput, they are evaluated with varying parameters and network topologies, so it is difficult to compare them.

6.4 Transmission Power Control

Transmission power control is an often-ignored area of wireless sensor network research. The propagation of wireless transmissions is extremely complicated, and by lowering transmission power, we add more uncertainty to already difficult problems. Additionally, only 8 of the CC2420's 31 discrete power levels are documented [17]. For these reasons, among others, the research community has primarily focused upon other methods of conserving energy, such as energy-efficient MAC and routing layers. ATPC [27] empirically evaluated the relationship between Received Signal Strength Indicator (RSSI) and packet reception rate, discovering it to be approximately linear. ART [7] concluded that RSSI and Link Quality Indicator (LQI) are not robust enough metrics for some indoor WSN environments, so they developed their own metric, which takes into consideration the impact of topology control via dynamic transmission power control on channel contention. It is clear that significant energy saving potential exists in this area.

References

[1] The network simulator - ns-2. http://www.isi.edu/nsnam/ns/.

[2] Chatterjea, van Hoesel, and Havinga. Ai-lmac: an adaptive, information-centric and lightweight mac protocol for wireless sensor networks. In *Intelligent Sensors, Sensor Networks and Information Processing Conference, 2004. Proceedings of the 2004*, pages 381–388, Dec. 2004.

[3] Dutta, Feldmeier, Paradiso, and Culler. Energy metering for free: Augmenting switching regulators for real-time monitoring. In *Information Processing in Sensor Networks, 2008. IPSN '08. International Conference on*, pages 283–294, St. Louis, Missouri, 2008. IEEE.

[4] El-Hoiydi, Decotignie, Enz, and Le Roux. Poster abstract: wisemac, an ultra low power mac protocol for the wisenet wireless sensor network. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 302–303, New York, NY, USA, 2003. ACM.

[5] Fonseca, Dutta, Levis, and Stoica. Quanto: Tracking energy in networked embedded systems. *OSDI*, pages 323–338, 2008.

[6] Gay, Levis, von Behren, Welsh, Brewer, and Culler. The nesc language: A holistic approach to networked embedded systems. *SIGPLAN Not.*, 38(5):1–11, 2003.

[7] Hackmann, Chipara, and Lu. Robust topology control for indoor wireless sensor networks. In *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 57–70, New York, NY, USA, 2008. ACM.

[8] Halkes and Langendoen. Crankshaft: An energy-efficient mac-protocol for dense wireless sensor networks. *Lecture Notes in Computer Science*, 4373/2007:228–244, 2007.

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

[10] Hill and Culler. Mica: a wireless platform for deeply embedded networks. *IEEE Micro*, 22(6):12–24, November 2002.

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

[12] Crossbow Technology Inc. Analog, linear, and mixed-signal devices from maxim/dallas semiconductor. http://www.maxim-ic.com/.

[13] Crossbow Technology Inc. Imote2 data sheet. http://www.xbow.com/.

[14] Crossbow Technology Inc. Mica2 data sheet. http://www.xbow.com/.

[15] Crossbow Technology Inc. Micaz data sheet. http://www.xbow.com/.

[16] Crossbow Technology Inc. Telosb data sheet. http://www.xbow.com/.

[17] Texas Instruments Inc. Cc2420 data sheet. http://www.ti.com.

[18] Jamieson, Balakrishnan, and Tay. Sift: A mac protocol for event-driven wireless sensor networks. In *ESWN '06*, pages 260–275, 2006.

[19] Jang, Lim, and Sichitiu. As-mac: An asynchronous scheduled mac protocol for wireless sensor networks. In *Mobile Ad Hoc and Sensor Systems, 2008. MASS 2008. 5th IEEE International Conference on*, pages 434–441, Atlanta, Georgia, USA, 2008. IEEE.

[20] Jiang, Dutta, Culler, and Stoica. Micro power meter for energy monitoring of wireless sensor networks at scale. In *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*, pages 186–195, New York, NY, USA, 2007. ACM.

[21] Holger Karl and Andreas Willig. *Protocols and Architectures for Wireless Sensor Networks*. John Wiley & Sons Ltd, 2007.

[22] Keshav. Real 5.0 overview. http://www.cs.cornell.edu/skeshav/real/overview.html.

[23] Klues, Hackmann, Chipara, and Lu. A component-based architecture for power-efficient media access control in wireless sensor networks. In *SenSys '07: Proceedings of the 5th international conference on Embedded networked sensor systems*, pages 59–72, New York, NY, USA, 2007. ACM.

[24] Köpke, Swigulski, Wessel, Willkomm, Haneveld, Parker, Visser, Lichte, and Valentin. Simulating wireless and mobile networks in omnet++ the mixim vision. In *Simutools '08: Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, pages 1–8, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[25] Levis, Lee, Welsh, and Culler. Tossim: accurate and scalable simulation of entire tinyos applications. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 126–137, New York, NY, USA, 2003. ACM.

[26] Levis, Madden, Polastre, Szewczyk, Whitehouse, Woo, Gay, Hill, Welsh, Brewer, and Culler. Tinyos: An operating system for sensor networks. In *Ambient Intelligence*, pages 115–148. Springer Berlin Heidelberg, 2005.

[27] Lin, Zhang, Zhou, Gu, Stankovic, and He. Atpc: adaptive transmission power control for wireless sensor networks. In *SenSys '06: Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 223–236, New York, NY, USA, 2006. ACM.

[28] Milenkovic, Milenkovic, Jovanov, Hite, and Raskovic. An environment for runtime power monitoring of wireless sensor network platforms. In *System Theory, 2005. SSST '05. Proceedings of the Thirty-Seventh Southeastern Symposium on*, pages 406–410, March 2005.

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

[30] Pottie and Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43:51–58, 2000.

[31] Rhee, Warrier, Aia, Min, and Sichitiu. Z-mac: A hybrid mac for wireless sensor networks. *Networking, IEEE/ACM Transactions on*, 16(3):511–524, June 2008.

[32] Shnayder, Hempstead, Chen, Allen, and Welsh. Simulating the power consumption of large-scale sensor network applications. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 188–200, Baltimore, Maryland, USA, 2004. ACM.

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

[34] Tao, Radhakrishnan, and Sarangan. Pmac: An adaptive energy-efficient mac protocol for wireless sensor networks. In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 12*, page 237.1, Washington, DC, USA, 2005. IEEE Computer Society.

[35] van Dam and Langendoen. An adaptive energy-efficient mac protocol for wireless sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 171–180, New York, NY, USA, 2003. ACM.

[36] Varga. The omnet++ discrete event simulation system. In *European Simulation Multiconference (ESM'2001)*, June 2001.

[37] Werner-Allen, Dawson-Haggerty, and Welsh. Lance: optimizing high-resolution signal collection in wireless sensor networks. In *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 169–182, New York, NY, USA, 2008. ACM.

[38] Ye, Heidemann, and Estrin. Medium access control with coordinated adaptive sleeping for wireless sensor networks. *IEEE/ACM Trans. Netw.*, 12(3):493–506, 2004.

[39] Ye, Silva, and Heidemann. Ultra-low duty cycle mac with scheduled channel polling. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 321–334, Boulder, Colorado, USA, 2006. ACM.

A Mote Programming Script

```
# This script is used to build an application on a given number of motes
# Author: Brian Bates
# Arguments:
# 1 - The number of motes to program
# 2 - The MAC layer to use
# 3 - File containing defines
# 4 - USB port number to start at (optional)
# 5 - Address number to start at (optional)

CURR_ARG="0"
#Open up the file specified to read the flags
while read line
do
        if [ -n "$line" ]
        then
                echo "-D$line"
                CFLAGS="$CFLAGS -D$line"
        fi
done < $3

#Set the static init number of nodes #CFLAGS="$CFLAGS -DSTATIC_INIT_NUM_NODES=$1"

export CFLAGS=${CFLAGS}

export UPMA_MAC=$2

#Loop through all of the motes CURR_USB="0"
if [ "$#" -ge "4" ]
then
        CURR_USB="$4"
fi

CURR_MOTE="0"
CURR_ADDR="1"
if [ "$#" -ge "5" ]
then
        CURR_ADDR="$5"
fi

while [ "$CURR_MOTE" -lt $1 ]
do
        echo "make telosb install.${CURR_ADDR} bsl,/dev/ttyUSB${CURR_USB}"
        make telosb install.${CURR_ADDR} bsl,/dev/ttyUSB${CURR_USB}
        CURR_MOTE=`expr $CURR_MOTE + 1`
        CURR_ADDR=`expr $CURR_ADDR + 1`
        CURR_USB=`expr $CURR_USB + 1`
done
```

B Energy Usage Graphs

This appendix contains energy usage graphs for all of our experiments. Each graph depicts the energy usage for the AS-MAC, SCP-MAC, Crankshaft, and BAS-MAC protocols as stacked bars broken down by radio states.

B.1 Local Gossip

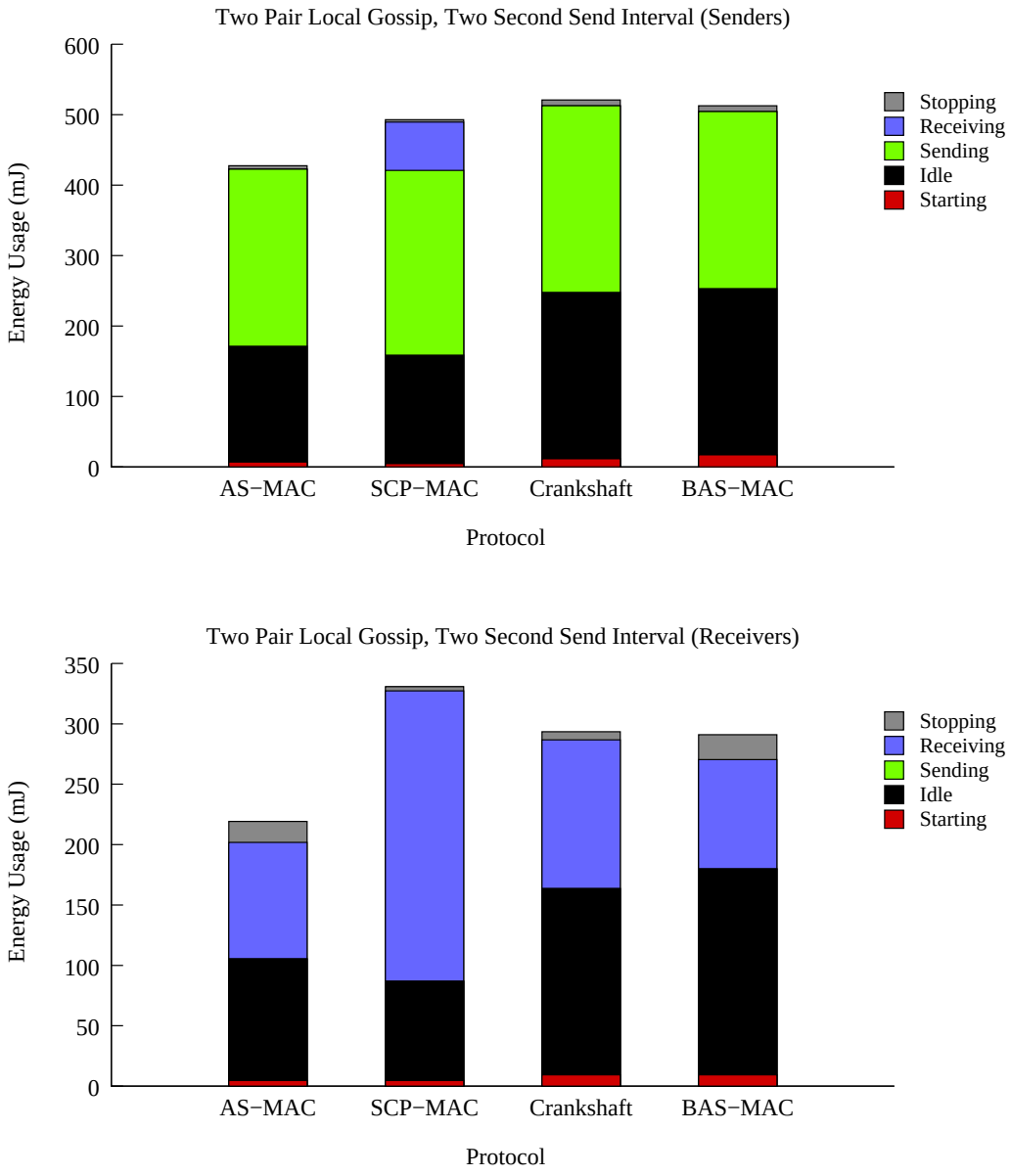

Figure 31: Two Pair Local Gossip - Fast Sending

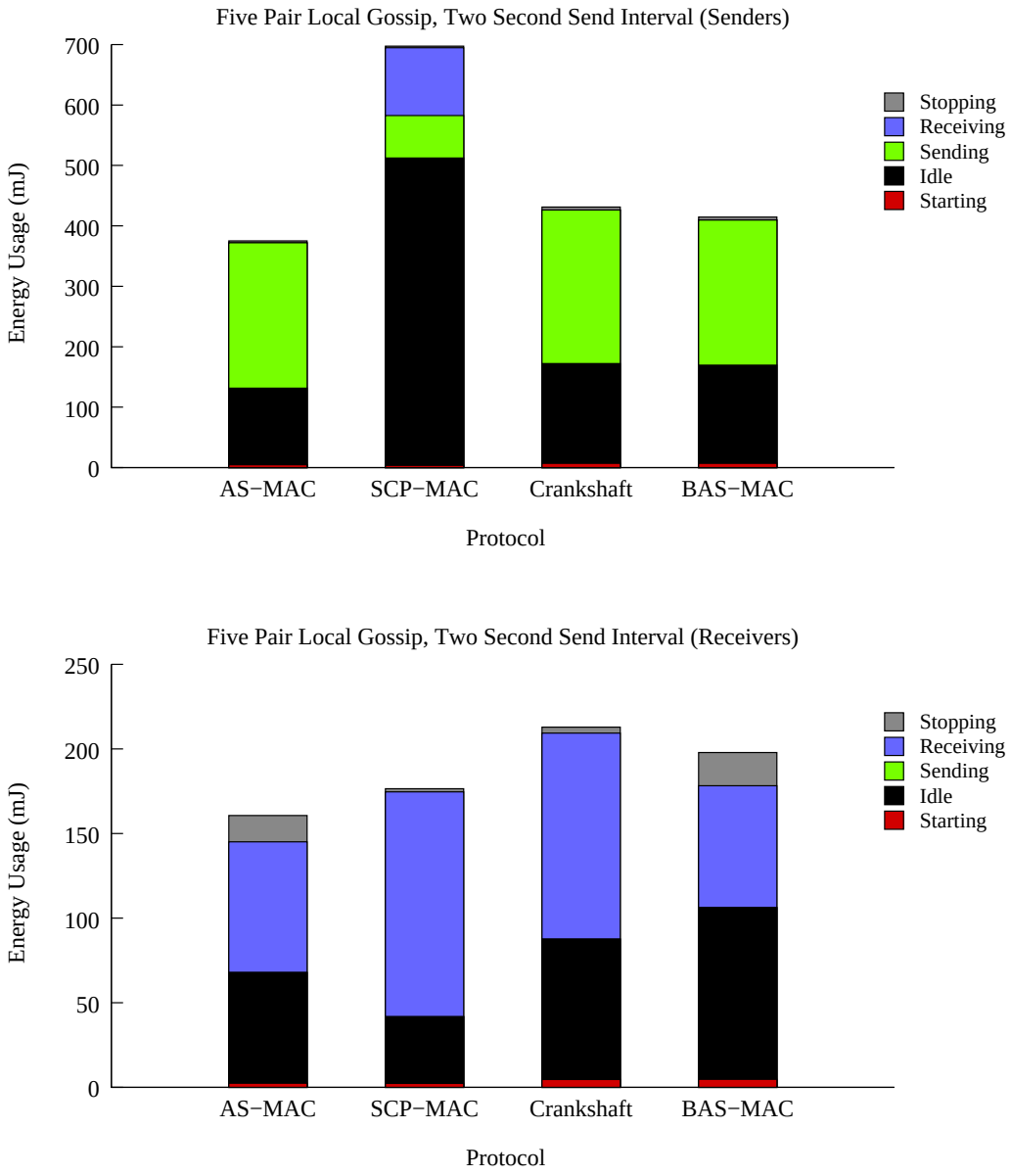

Figure 32: Five Pair Local Gossip - Fast Sending

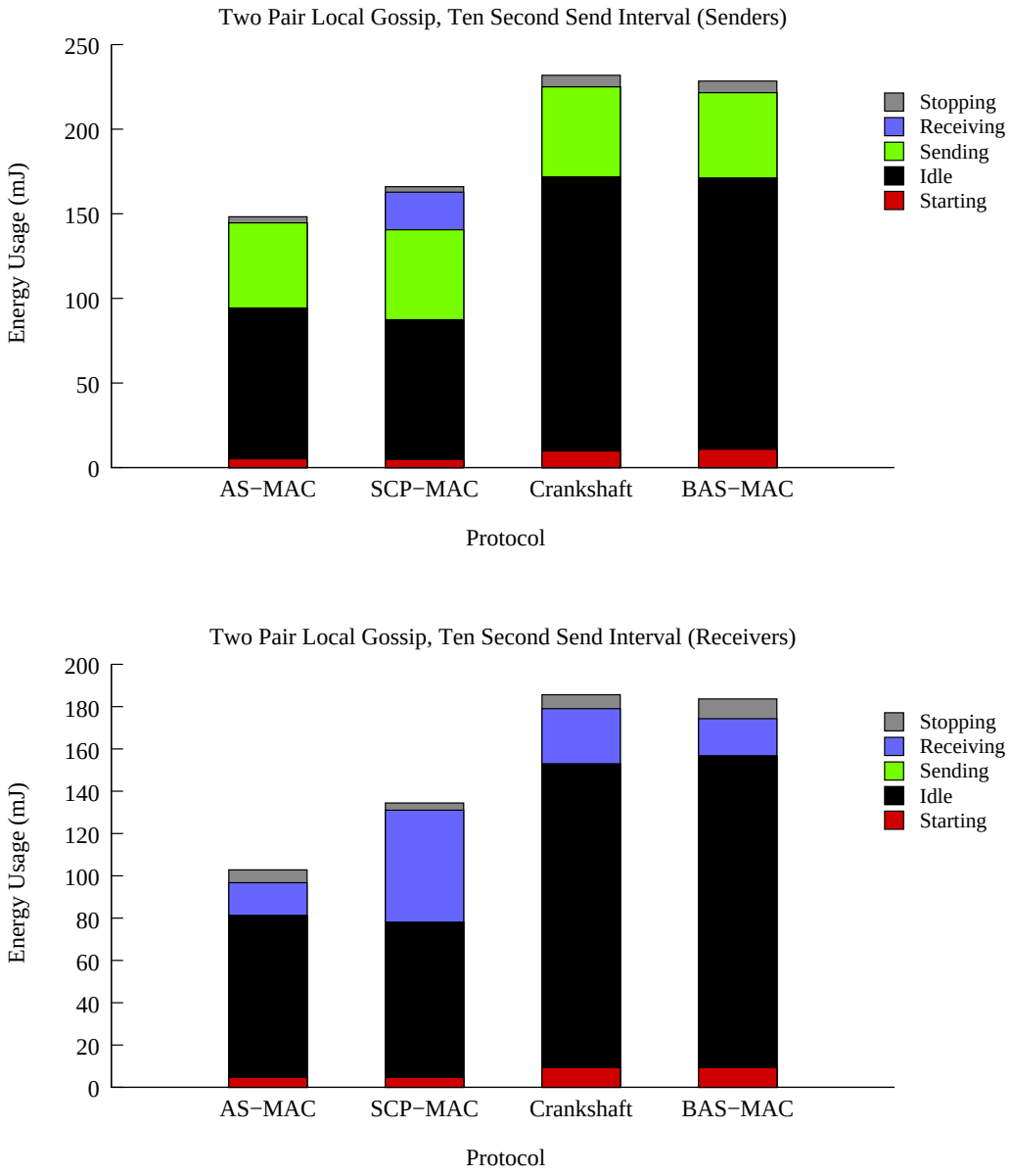

Figure 33: Two Pair Local Gossip - Slow Sending

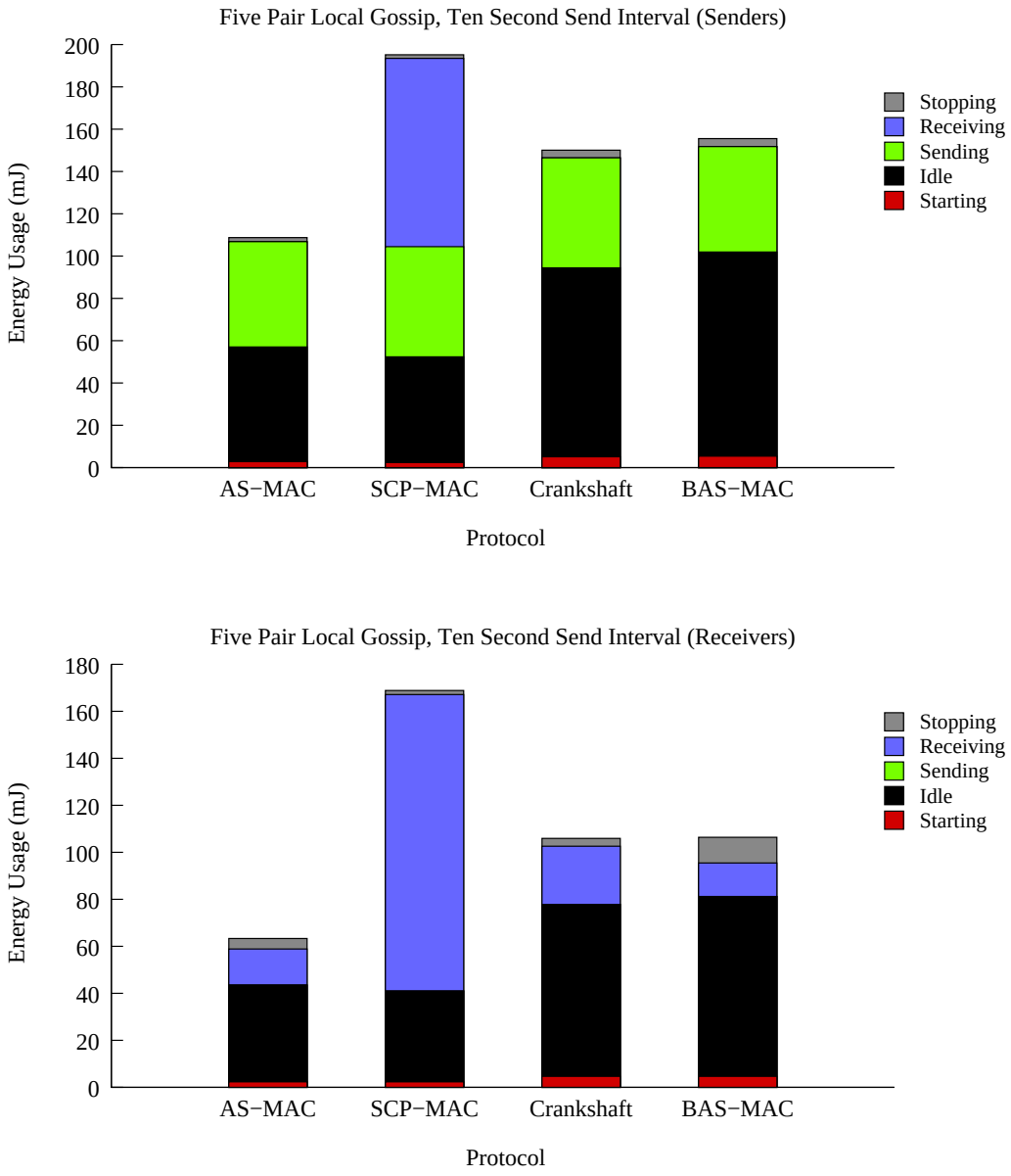

Figure 34: Five Pair Local Gossip - Slow Sending

B.2 Convergecast

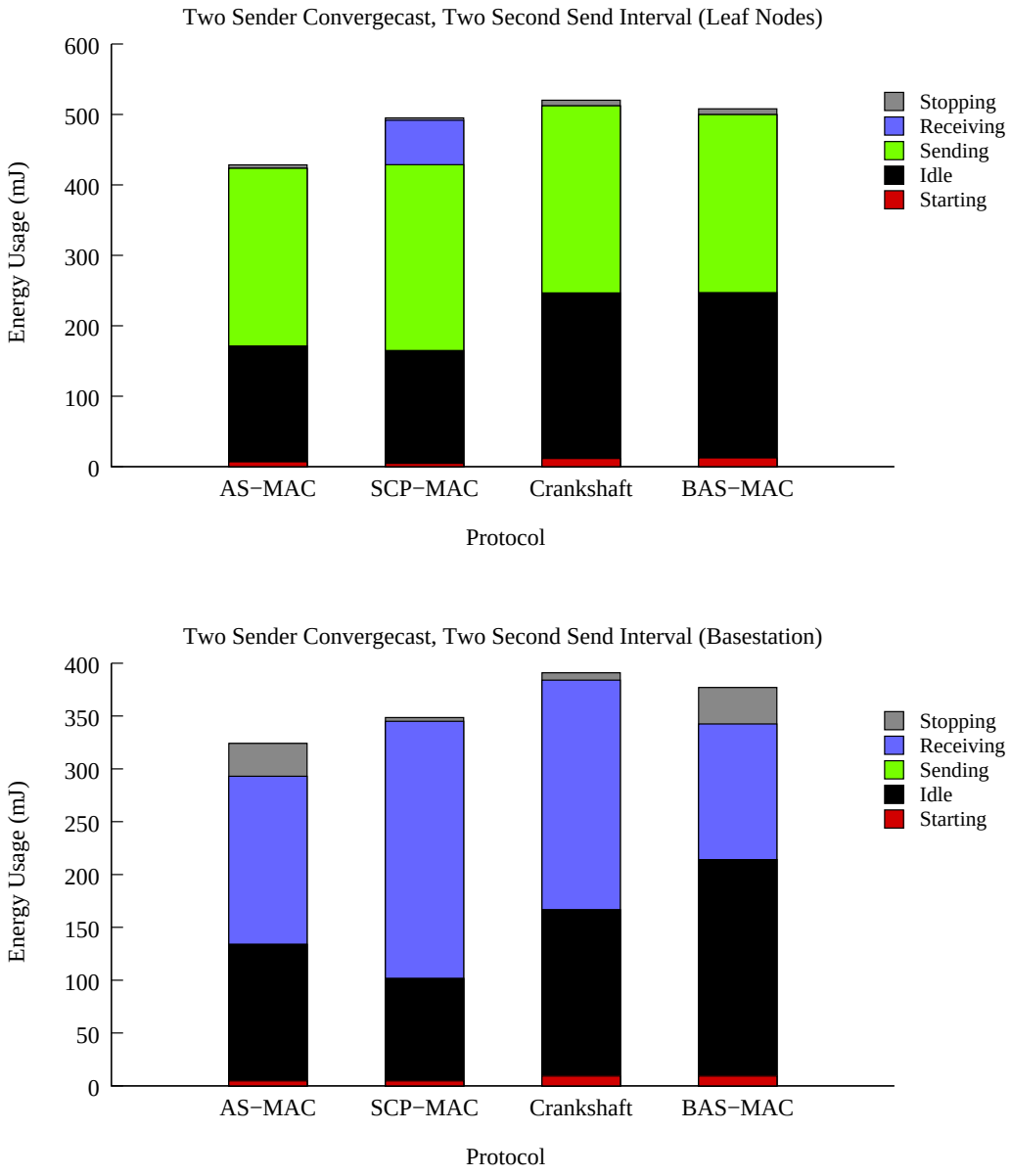

Figure 35: Two Sender Convergecast - Fast Sending

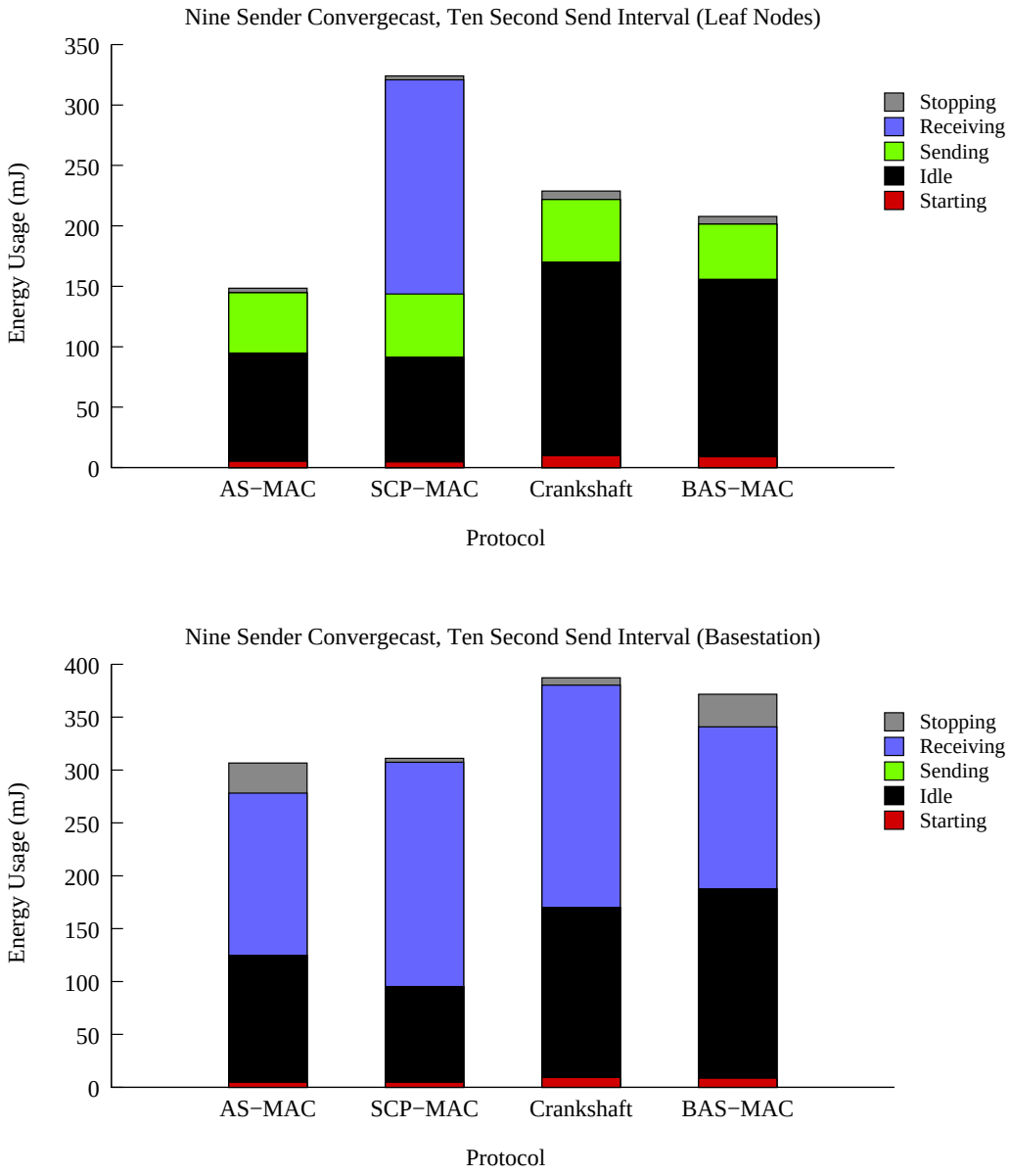

Figure 36: Nine Sender Convergecast - Fast Sending

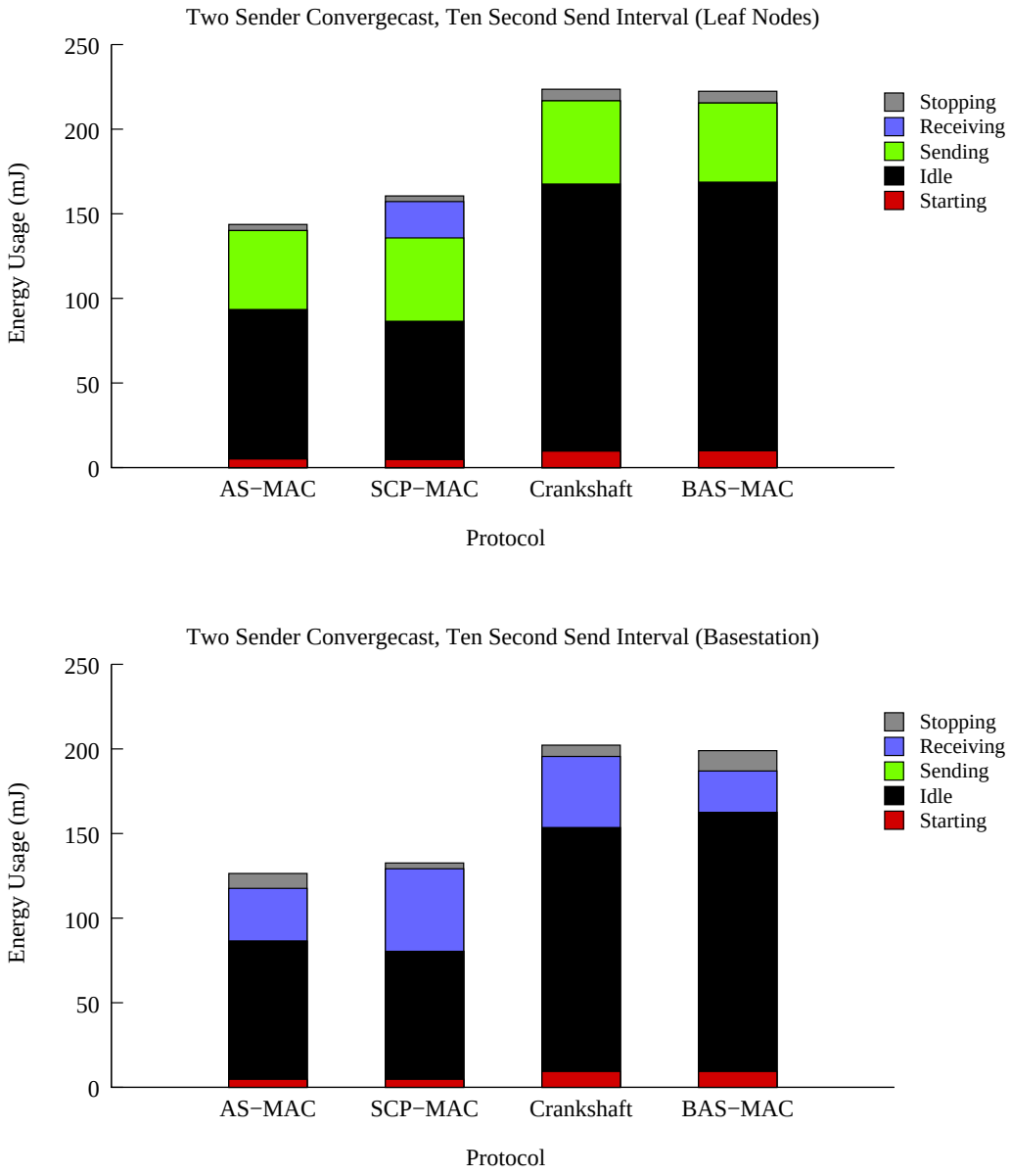

Figure 37: Two Sender Convergecast - Slow Sending

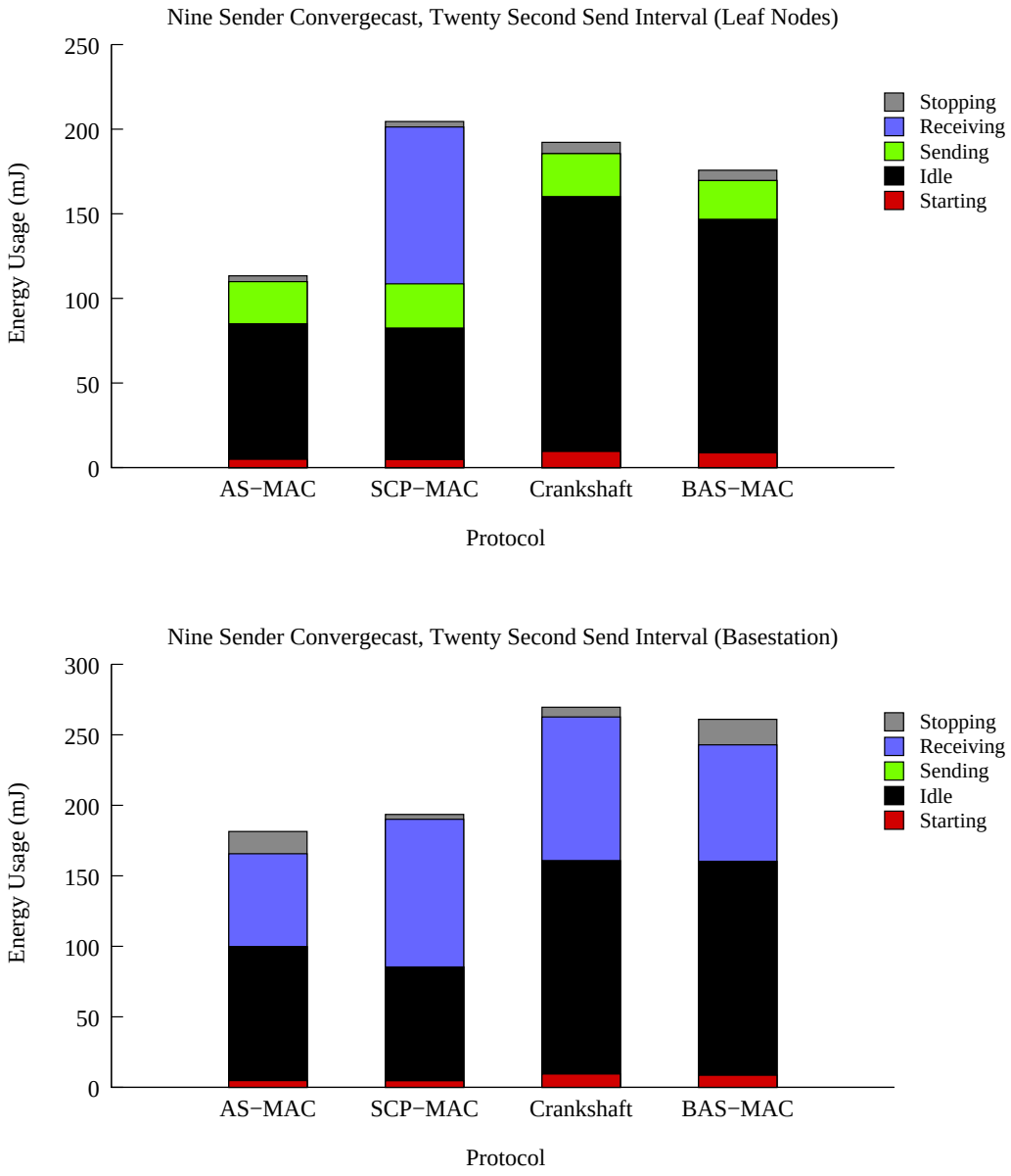

Figure 38: Nine Sender Convergecast - Slow Sending

B.3 Broadcast

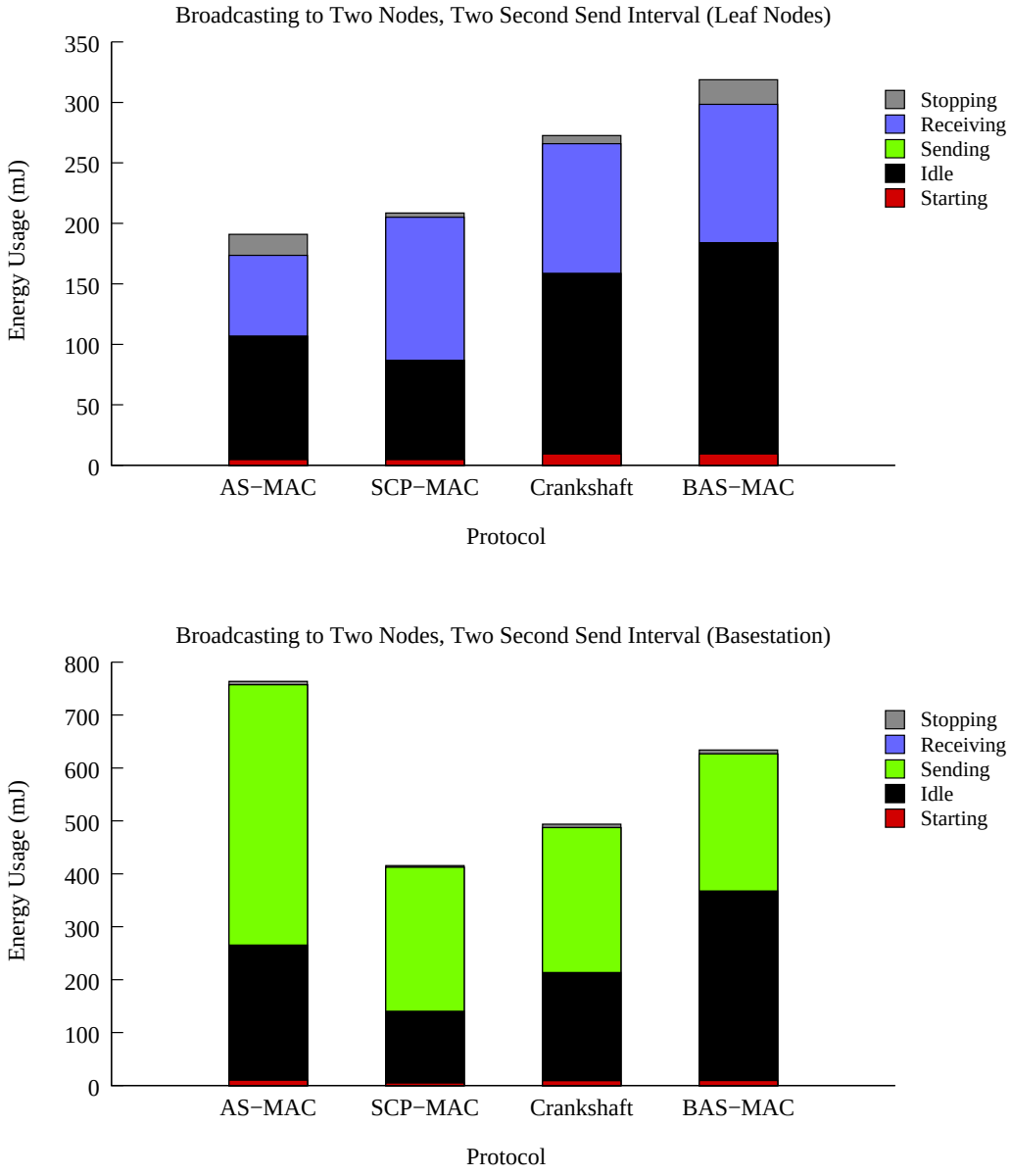

Figure 39: Two Receiver Broadcast - Fast Sending

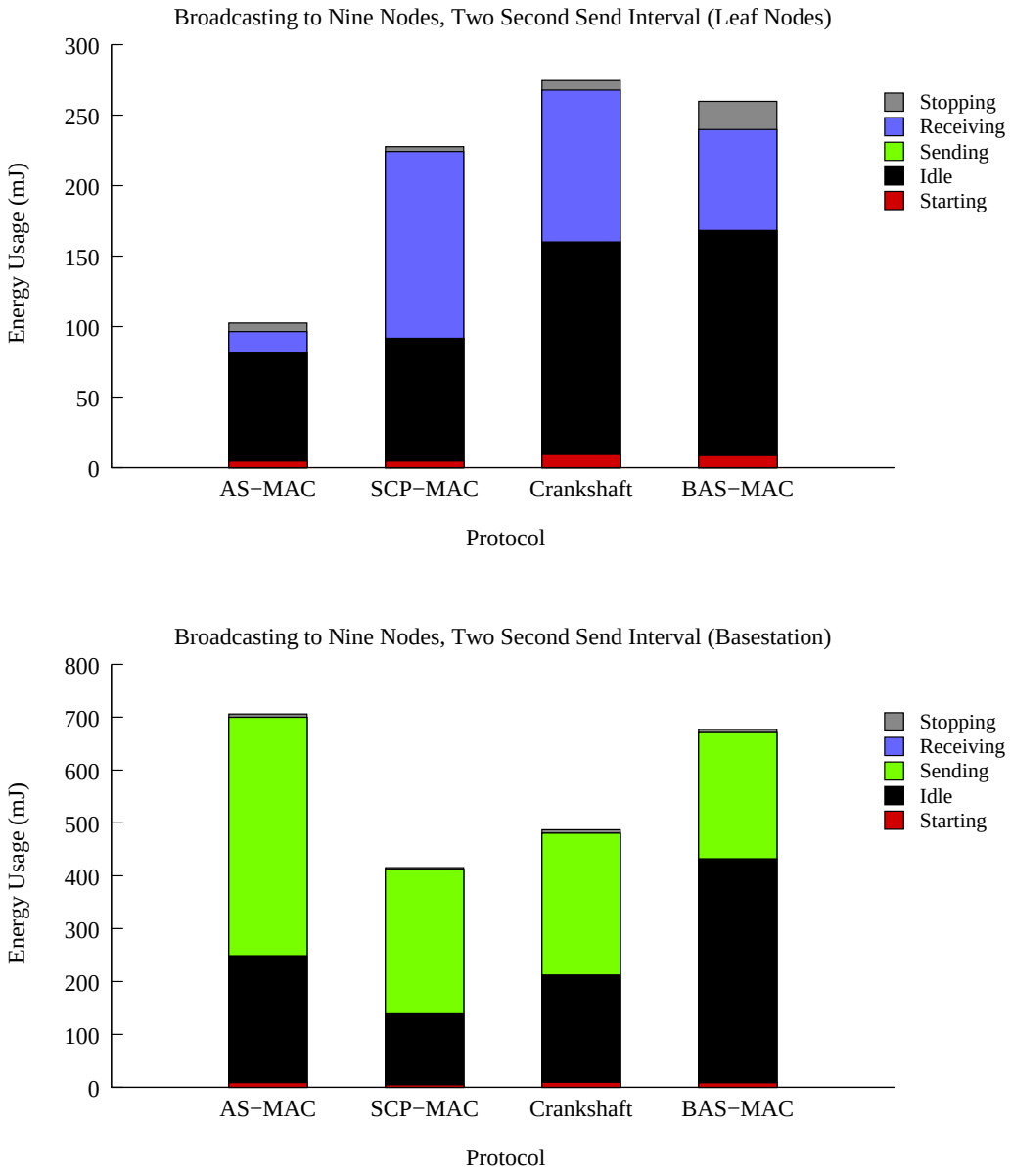

Figure 40: Nine Receiver Broadcast - Fast Sending

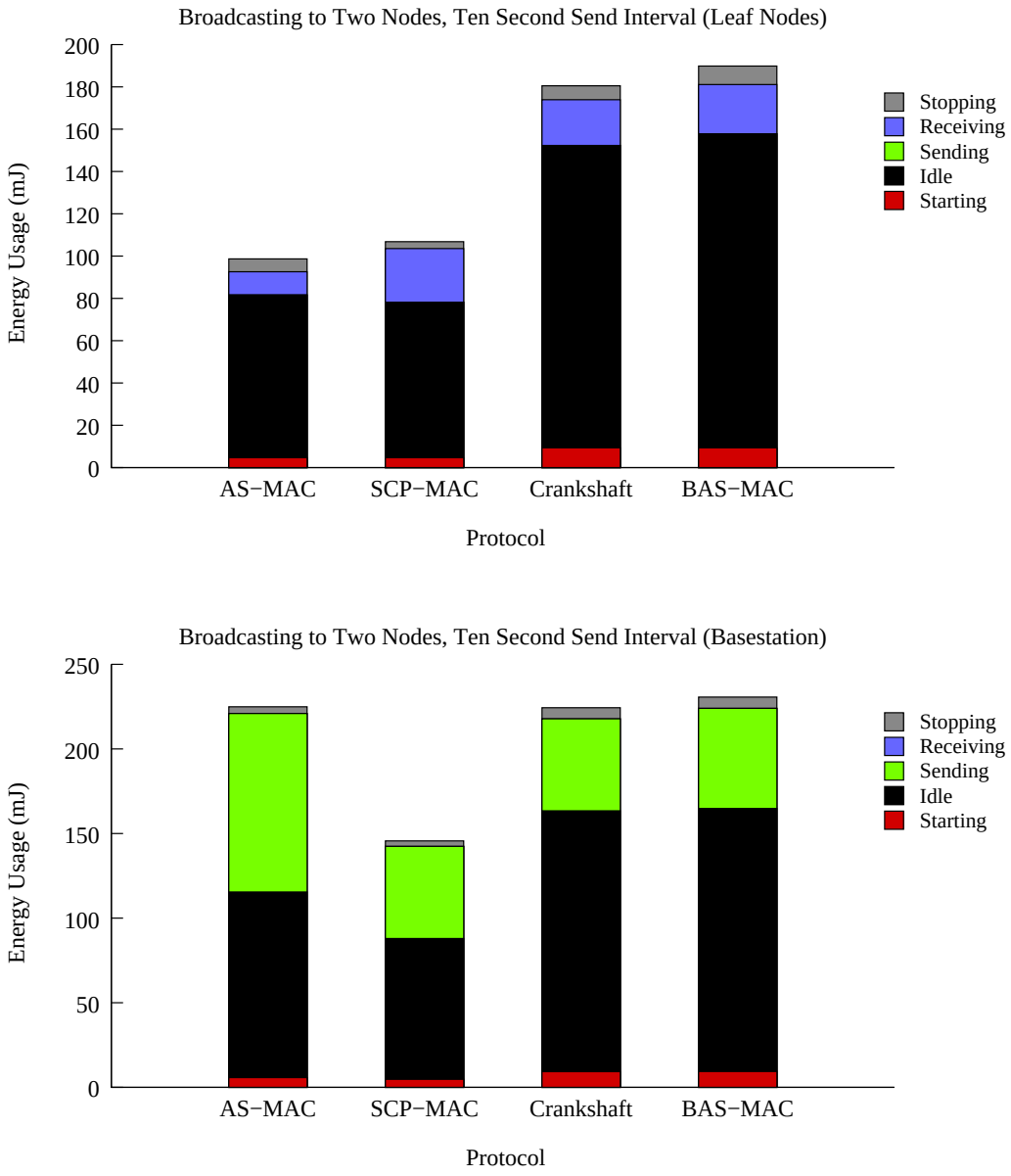

Figure 41: Two Receiver Broadcast - Slow Sending

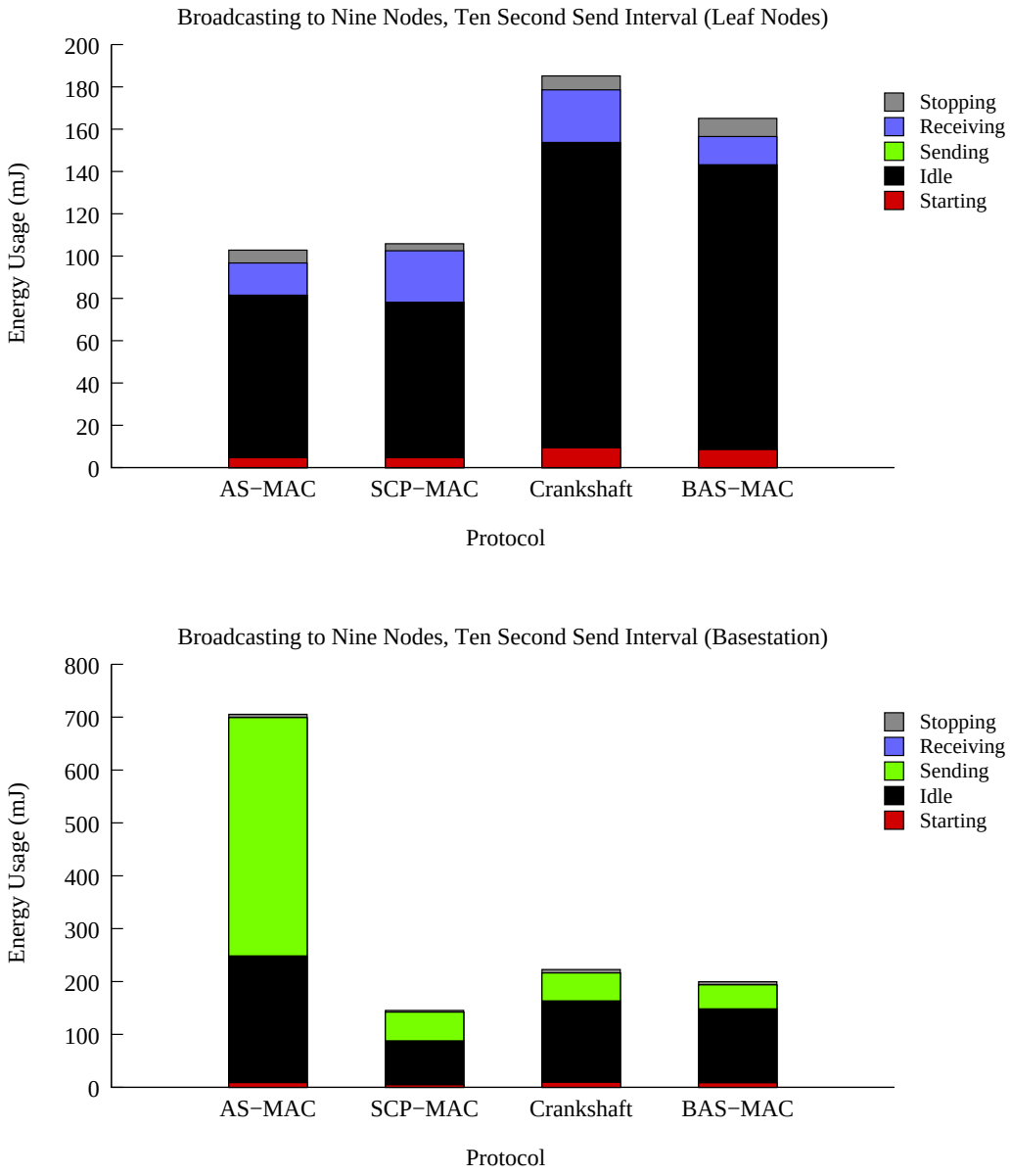

Figure 42: Nine Receiver Broadcast - Slow Sending

C Oscilloscope Measurements

This appendix contains oscilloscope images taken when creating the energy profile used for the TelosB motes. These images show the general behavior of the radio and its states when sending, receiving, and performing wakeups.

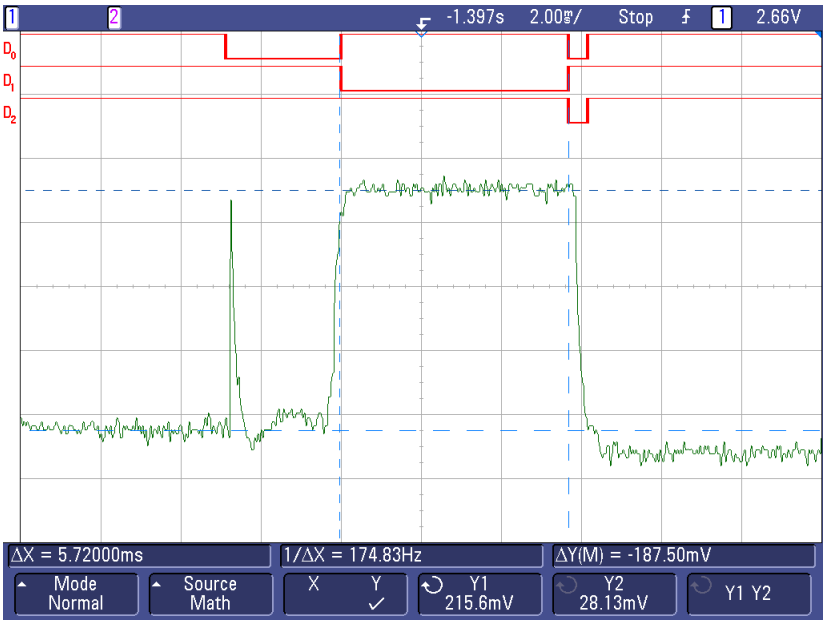

Figure 43: Oscilloscope Image - Radio Wakeup

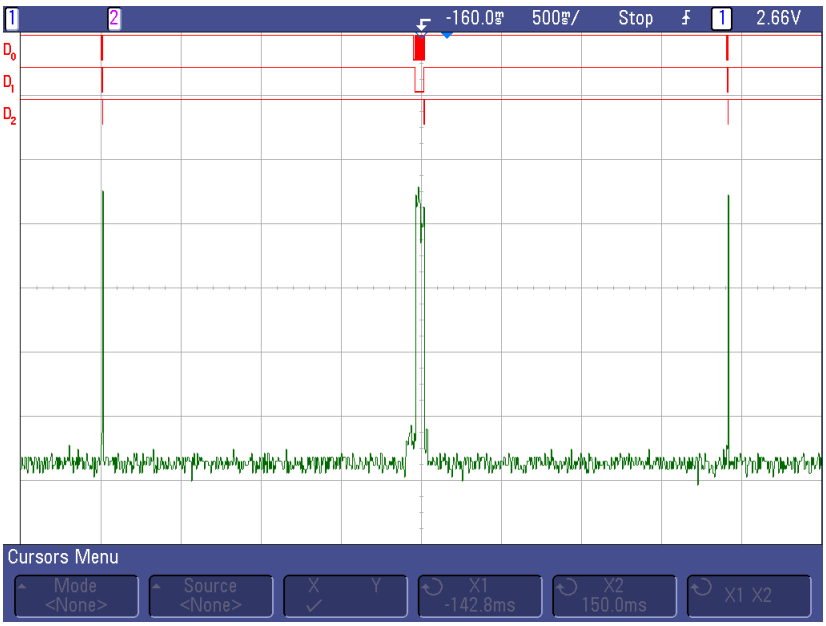

Figure 44: Oscilloscope Image - Idle and Receiving Wakeups

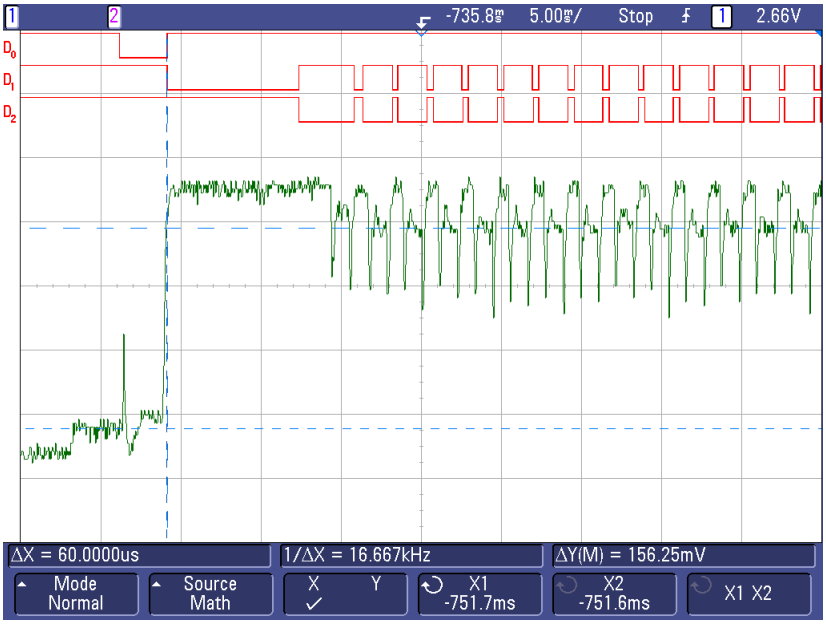

Figure 45: Oscilloscope Image - Sending Preambles

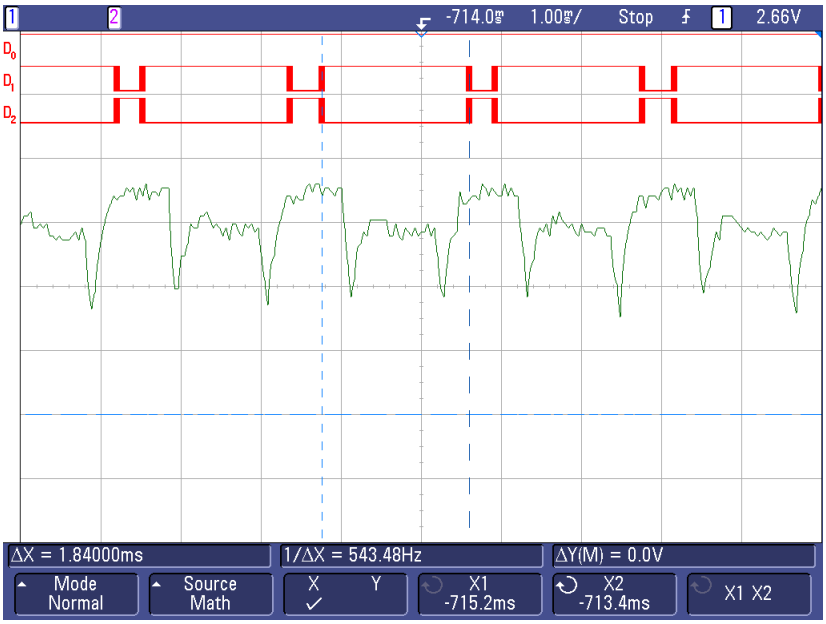

Figure 46: Oscilloscope Image - Preamble Close-up

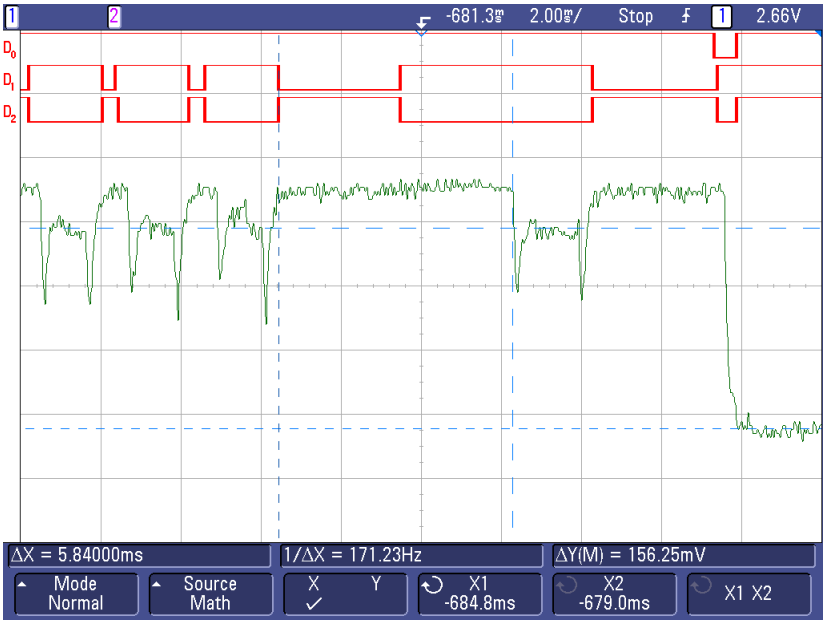

Figure 47: Oscilloscope Image - Preamble and Data Packet Sending

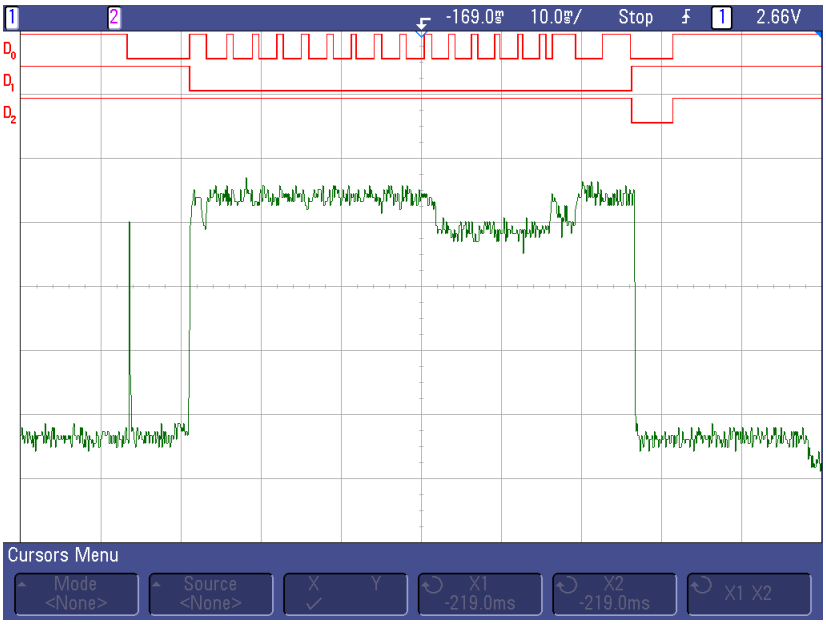

Figure 48: Oscilloscope Image - Receiving a Message