Field-deployable Concurrent-Transmitter Networks for Long Term Environmental Monitoring

by
Douglas Carlson

A dissertation submitted to the Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy.

Baltimore, Maryland
January 2014

Abstract

In this work, I draw upon my experience with field deployments of low-power data collection systems to come up with new approaches to long-term environmental monitoring (LTEM). Our goal is to develop long-lived, reliable systems that can be deployed by non-experts.

The specific requirements of LTEM systems lead us to pursue techniques which can take advantage of the natural heirarchies that exist in sensor deployments while avoiding the difficulties inherent in selecting efficient data transmission routes in the face of unreliable hardware and difficult-to-measure link dynamics.

The approach we advocate eschews traditional single-path routing and transmission methods in favor of approaches that leverage non-destructive simultaneous packet transmissions over subsets of the network. We apply this principle to develop a medium access protocol suitable for dense networks (Flip-MAC) as well as a method for identifying the set of potentially-useful forwarders between a data source and its destination (CX).

This document not only characterizes and evaluates the low-level behavior of these protocols, but also describes the design of a larger multi-tiered data collection system based on CX, a suite of hardware which is well-suited to both CX and common deployment patterns, and the design of a "dirt-to-database" system which gives domain scientists the tools they need to deploy and manage networks on their own.

Dr. Andreas Terzis *Advisor*	Associate Professor Department of Computer Science The Johns Hopkins University
Dr. Alexander Szalay *Primary reader*	Professor Department of Computer Science, Department of Physics and Astronomy The Johns Hopkins University
Dr. Omprakash Gnawali *Secondary Reader*	Assistant Professor Department of Computer Science University of Houston

For all of my teachers.

Acknowledgements

Research is an inherently collaborative process, and I'd like to thank my personal and professional collaborators that made this work possible.

I'd like to thank my family for their constant support and encouragement. They gave me the opportunities, motivation, and skills necessary to pursue a meaningful and purposeful education. They provided me with an idyllic childhood. Words can't express my gratitude. This isn't limited to my parents, Chuck and Edith: my sister Christina has been a shining example of positive thinking and hard work my entire life. I'm very fortunate to come from an extended family that has valued education and individual accomplishment, and I would not be in the place that I am without them. I'd like to particularly thank my grandparents Wayland and Myrtle Carlson and Maurice and Gwen Whitlock. Their hard work and dedication made my family's accomplishments possible, and the pride they take in their descendants' accomplishments is well-deserved.

I'd like to thank Kelly Boyd: everything that I do, I do for our team. Your support and encouragement have gone so far beyond the bounds of what is reasonable to expect that it boggles the mind. I'd also like to thank Kelly's family and long-time friends. They've welcomed me with open arms and made me feel like this is really my home.

My HINRG labmates, past and present, have been integral to this process. Dr.'s Razvan Musaloiu-Elefteri and Jayant Gupchup: your guidance and collaboration at the start of my time at JHU got me on my feet and gave me the tools I needed to be successful. Plus, it was a lot of fun writing papers with you guys. Dr. Chieh-Jan Mike Liang was my TA and frequent teacher in all matters of embedded programming, and he never failed to brighten my mood. Dr. Jeonggil Ko: your dedication to work was amazing and sometimes intimidating, but you were always happy to lend a hand and a joke. Dr. Jonghyun Lim, you were continuously

upbeat and a joy to be around. Andong Zhan, Zainan Victor Zhou, and Da Zheng: you guys brought fresh perspectives and excitement to our lab, and I'm sure that the future will bring you many great successes. Dr. Yin Chen, I am consistently blown away by the depth and breadth of your knowledge. Combined with your quiet and cheerful attitude, you are a real force for Getting Things Done. Finally, I must thank Dr. Marcus Chang. I don't think I've had a single research idea in the last five years that hasn't been in some way improved by our discussions.

A big thank-you is due to my collaborators outside of the lab. Our colleagues in the Earth and Planetary Sciences department have given this thesis a reason to exist. Without their input and without their hard work, none of this would be possible. Thank you, Dr. Katalin Szlavecz, Scott Pitz, Chih-Han Chang, Lijun Xia, and Michael Bernard.

Thanks go to Dr. Alex Szalay for his guidance in my PhD. From day one, his wide and deep pool of knowledge has made our work possible. His feedback in my Graduate Board Orals helped to shape the work appearing in this thesis, and his performance on my defense committee have made it the document that it is.

Thank you to Dr. Omprakash Gnawali for our many collaborations over the years. Your perspective on the sensor network field and your attention to detail greatly improved all of our joint work. Thank you for your feedback on my thesis. Additionally, I must thank you for volunteering the resources of your research lab to extending and improving the Breakfast platform software and providing a sense of continuity for our projects.

I have many things for which to thank my advisor, Dr. Andreas Terzis. First, thank you for having the confidence in me to accept me into the program. Thank you for smoothly guiding me from a novice to an expert in an exciting and fascinating field. Thank you for giving me the tools, flexibility, and trust to work independently. Thank you for the many hours of discussion that shaped my vision of sensor networks. It is an honor to have been

your advisee.

In more general terms, I'd like to thank all teachers (but especially mine). Teaching is one of the most noble professions in the world, and a long line of dedicated teachers got me to where I am today. I have to give a special thanks to my first grade teacher, Mrs. Clark, who made an extra effort to get me into gifted and talented programs at an early age. Recently, as I have looked for careers, I have frequently thought to myself "would it make Mrs. Clark happy to see me doing this?" It's important to be mindful of how the work that you do affects the world around you. We all owe a debt of gratitude to the education professionals that make the successes of our scholars and scientists possible.

In addition, I'd like to thank the various funding sources that made the work in this thesis possible: the National Science Foundation through grants 0546648, 0754782, and 1111507, NSF-MIRTHE, and the Moore Foundation.

The work in Chapter 3 originally appeared in DCOSS 2011 as "Flip-MAC: A Density-Adaptive Contention-Reduction Protocol for Efficient Any-to-One Communication." [9] Chapter 4 originally appeared in DCOSS 2013 as "Forwarder Selection for Multi-Transmitter Networks," [8]. The material in Section 4.7.4 and Section 4.1 is new in this thesis. Elements of Chapter 2 were originally reported in "K2: A System for Campaign Deployments of Wireless Sensor Networks" [10] and in my PhD qualifying project "Challenges of Faults in Long-Term Sensor Network Deployments." The latter was a collaborative effort with Dr. Jayant Gupchup, Dr. Omprakash Gnawali, and Dr. Andreas Terzis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Many of our planet's physical phenomena can only be understood through direct observation. Deploying arrays of sensors in a natural environment can allow environmental scientists and others to discover previously unobserved processes, build models of complex systems, and validate or invalidate previously-held beliefs. Wireless Sensor Networks (WSNs) have been proposed as tools which can enable such observations at previously-impractical spatial and temporal scales. It is these Long-Term Environmental Monitoring (LTEM) systems which we wish to build.

At a high level, we claim that WSNs for environmental monitoring must:

- Be robust to failures of individual devices or links: devices and links are unreliable in the wild.
- Support hierarchical groupings of analog sensors and wireless nodes: this is how domain scientists design experiments.
- Automatically capture as much deployment metadata as possible: misconfigured or misidentified devices cannot deliver usable data.

In this thesis, we will present hardware and software solutions that address each of these requirements.

In Chapter 2, we describe our motivations behind these claims and lay the groundwork for the rest of this thesis.

In Chapter 3, we present a medium access protocol, Flip-MAC, suitable for dense groups of wireless nodes, such as those seen in our deployments. This work uses the principle of non-destructive radio collisions to operate. If multiple devices wirelessly send the same data at the same time, the result can still be properly decoded. This property is in stark contrast to destructive collisions, where the combined signal of differing or unaligned transmissions cannot be decoded reliably. By taking advantage of this effect, a potential data recipient can engage in a series of communication rounds with the *set* of devices wishing to send them data, rather than each individual sender. This insight lets us reduce the level of contention in logarithmic time to the number of competing transmitters.

In Chapter 4, we apply the same principle to multi-hop communication. We describe a method, which we call *CX*, to efficiently identify the set of nodes which lie between a source and its destination using multi-transmitter floods. In a multi-transmitter flood, a node sends a packet, and each recipient rebroadcasts it at exactly the same time (resulting in non-destructive collisions). We can use this networking primitive to quickly ascertain which nodes are "useful" forwarders for a source-destination pair, and use this to both save energy (by turning off radios at unused nodes) and increase data throughput (by limiting the number of "hops" that each packet travels to the distance between the source and destination). This approach aims to reduce the impact of individual link or transmitter failures on reliability, and avoids the need for costly link-estimation.

In Chapter 5, we extend CX to operate in a multi-tiered network, such as those we see in LTEM deployments. We also take steps to make it revert to a low-power idle mode when elements of the network infrastructure are absent.

We tie all of this together into a package which can be deployed by non-experts in Chap-

ter 6. In order for LTEM systems to succeed, they must be usable by somebody other than the system designer. Here, we present a suite of hardware that matches our deployment needs and supporting software to automate many of the common tasks necessary to ensuring data provenance and quality. We end this chapter with a qualitative discussion of system usability drawn from a preliminary deployment.

We close with general remarks and discussion in Chapter 7.

Chapter 2

Motivation and Background

In this chapter, we describe the Environmental Monitoring application space in some detail. We present our observations of the major requirements of this application, explain the shortcomings that we see in the current approaches, and lay the groundwork for the solutions presented in later chapters.

2.1 Application Requirements

In order to understand the application space for this research, we should start with our high level system goals and present a few illustrative deployments.

Our overall task is to instrument some study area with analog sensors, sample them periodically, and wirelessly collect their data. We place a number of *motes* in the field to carry out this task. A mote is a low-power device consisting of a microcontroller, radio, sensor inputs, and some persistent data storage. In this work, we may also refer to these as *nodes*.

Sensor data must be collected reliably and must be assigned meaningful physical values in time and space. The system itself should be deployable and maintainable by non-specialists. It should be able to exist in the field on battery power for the duration of the study period.

Table 2.1 summarizes several major deployments we have conducted over the years in support of the Life Under Your Feet project [59], which studies soil ecology with sensor net-

Name	Duration	#Nodes	Coverage	Data Rate	Internet
Cub Hill	08/08 - 07/12	50	Uniform	2.6 B/min	Permanent
SERC	03/09 - 06/11	36	Patches	2.6 B/min	Permanent
USDA	07/09 - 08/10	22	Patches	2.6 B/min	Permanent
Brazil	11/13/09 - 12/18/09	50	Uniform	52 B/min	Transient
Ecuador	05/22/10 - 06/07/10	20	Patches	52 B/min	Transient

Table 2.1: Summary of Koala [44] and K2 [10] Deployments.

works. They can be roughly grouped into campaign-style deployments (where data is uploaded to a laptop and reasearchers visit the site frequently), and semi-permanent deployments. The overall trend has been towards patchy deployments, where a few widely-spread study areas are densely instrumented.

Requirements may vary from deployment to deployment, but there are a few items that we have found to be common to studies of soil conditions and lower atmosphere dynamics, the areas which we have directly helped our colleagues to explore.

Data rates are relatively low. Measurements germane to soil respiration (temperature, volumetric water content, photosynthetically-active radiation, etc) can be taken at the resolution of minutes or tens of minutes and provide meaningful information– these phenomena just don't change very rapidly. At the other extreme, we've performed measurements of soil CO_2 concentrations, air temperature, and relative humidity at a 30-second sampling interval. These deployments saw a per-node data rate of 2.6 to 52 B/minute.

Data latency is unimportant almost to the point of irrelevance. It's important that scientists can identify fault conditions within a few days or weeks of their occurrence (e.g. faulty sensor measurements, low batteries, moisture infiltration), but they don't need up-to-the-second measurements of the environment. Much of the required analyses are conducted over the course of seasons.

On the other hand, we do have very high data delivery requirements. Many environmen-

tal models require gap-free datasets to be applied. While it's inevitable that some measurements will be missed, every effort should be taken to minimize this. In practice, we have used linear interpolation to fill short gaps in a single sensor's data stream and more complex mechanisms using cross-sensor correlations to handle more difficult cases [27].

We need to support a range of deployment topologies, but most importantly "patchy" deployments. The analog sensors in use may be quite expensive. Soil moisture probes cost tens of dollars, while CO_2 sensors may run into hundreds of dollars apiece. For this reason and for ease of deployment, it's much more practical to densely instrument a few discrete and qualitatively different sites within a study area than it is to uniformly blanket the whole study area. We have supported deployments, for example, that attempt to characterize the differences between old-growth and secondary-growth forests, between forested and grassy areas, or between intentionally manipulated experimental sites. Such deployments may or may not align well with the communication range of wireless sensor nodes. Typical distances between individual analog sensors may be quite small (centimeters), while the distance between study sites may be hundreds of meters or more. On the other hand, smaller deployments may be uniformly instrumented, so an ideal system should handle both types of topologies.

Sensor measurements in themselves are not very useful. Without correct sensor metadata (types of sensors, locations, calibration records, etc.) to translate numbers into physical measurements in space and time, a sensor network can't deliver meaningful data. Manually collecting this information is tedious and error-prone. Anything that we can do to automate its collection will prevent misconfigurations and lost data.

Finally, network protocols that can reliably transfer data are necessary but not sufficient to the success of a WSN in the field. We must reduce maintenance and deployment effort as much as possible: replacing batteries or failed hardware is time-consuming, and delayed maintenance can lead to extensive data loss. Generally speaking, the biggest power consumer

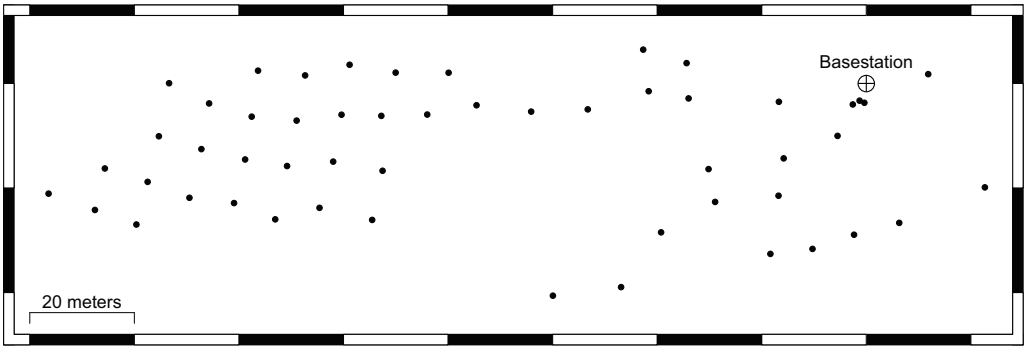


Figure 2.1: A map of the Cub Hill deployment. The communication range of nodes was typically greater than 20 meters and the network was very well connected.

is radio communication, so WSN systems strive to extend battery lifetime by turning their radios off as much as possible (called "duty-cycling"). We can't control all of the environmental factors which may damage hardware, but we can certainly make every effort possible to reduce battery usage.

In the rest of this chapter, we'll look at each of these factors in detail.

2.2 Deployment Patterns

The scientific missions of WSN deployments frequently lead to challenging networking conditions. Cub Hill (Figure 2.1) is the kind of deployment pattern that was originally envisioned for the TelosB mote platform [50] and the Koala collection protocol [44] (described in 2.4.1). Devices are more or less uniformly spaced, and there is a permanently powered, permanently Internet-connected base station. None of our subsequent deployments have followed this model.

Our deployment at the Smithsonian Environmental Research Center (SERC), for example, looks very different. This is pictured in Figure 2.2. The dark points represent 2-4 nodes, while the light points are single nodes serving primarily as relays, whose sole purpose was to keep the network connected. They were not connected to external analog sensors, and

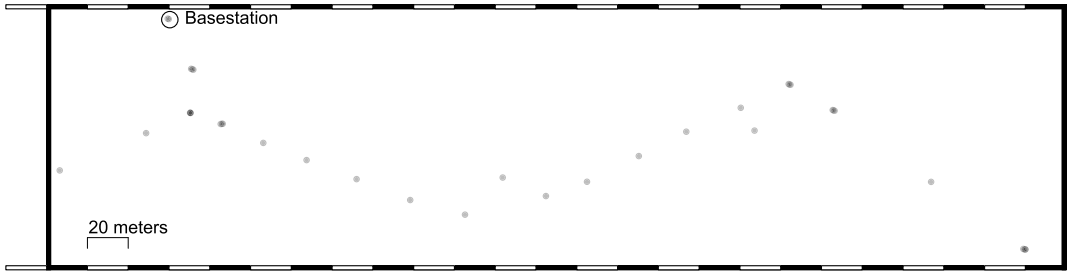


Figure 2.2: SERC deployment map. Due to scale, many node locations overlap: darker points represent multiple nodes, while light points represent single nodes. Note the change in scale from Figure 2.1

they were attached to trees a few meters off the ground. One third of the deployed nodes were relays. The left side of the map is an old-growth forest, while the right side of the map is secondary growth. Each of the dark points corresponds to a set of experimental plots. An experimental plot was roughly 1 m^2, and each plot was broken into two sections, each containing a different type of leaf litter. These plots were spaced a few meters apart. This deployment had to be visited on multiple occasions to replace relay nodes that had fallen from their locations and disconnected the network. Unlike Cub Hill, battery and enclosure failures at relays had to be resolved in order to maintain network connectivity.

Our Yasuni National Forest (Ecuador) deployment was even more patchy. Due to the logistics of setting up relays, the separate study sites were not in radio range of each other and had to be visited individually.

At Yasuni, this separation had the benefit of isolating the data forwarding load of each node from the other patches since only one patch could be woken up at a time. Since we had a fixed collection point at SERC, we could not take advantage of the spatial patchiness. All nodes were active at the same time, so nodes at the young forest site were essentially doing nothing useful while nodes at the old forest site were transmitting their data.

Given that we can expect future deployments to be patchy, there are energy savings available by intelligently segmenting the network. In Chapter 5, we will quantify the energy

savings possible by taking patches into account in our collection protocol.

We can also designate a subset of nodes as "routers" and equip them with longer-range transceivers. This reduces the amount of hardware that must be deployed and maintained, which is good for the domain scientists who have to change fewer batteries and is good for the computer scientists who want their networks to stay connected. In Section 6.2, we'll describe the "Bacon" wireless mote, which is well-suited to the node placement needs of environmental scientists.

The SERC deployment is composed not only of patches of nodes, but also of patches of sensors. Each experimental plot had one or two sets of soil temperature and moisture probes at 10 cm and 20 cm depths. Our USDA deployment was even more densely instrumented, with roughly 40 sensed locations in each of two small (several meter) patches.

The hardware which we used for these deployments allowed us to connect only 4 analog sensors to each node. This makes logistics and maintenance more difficult for dense deployments: why should we have to drain batteries in ten different boxes that are inches away from each other? Additionally, this prevents us from effectively amortizing connection costs. There's a certain amount of overhead necessary to setting up routes to each wireless node, storing sensor measurements, and putting them into packets. Reducing this overhead will reduce the amount of data which a network has to transmit, and this will reduce total energy usage. The "Toast" analog multiplexer board which we present in Section 6.2 gives users flexibility in how many analog sensors they can connect to a single sensor node.

Related Work These experiences are not unique to our research. One of the earliest WSN deployments, the Great Duck Island project [58] followed this "patchy" model. More recently, Tenet [49] advocated a tiered network architecture as well, though it is more oriented toward processing data in the network than collecting a full historical record.

The concept of separating sensing, storage, and routing concerns has previously been

proposed in the literature [52]. The Luster [53] system is representative of this approach, with dedicated sensing, storage, and communication nodes organized into clusters. However, the analog sensor interface they describe is subject to the same problems we aim to deal with using the Toast board, and their cluster-head devices [13] are more akin to expensive full-fledged PCs than motes.

Other researchers have pointed out the benefits of transceiver heterogeneity [72] in WSNs, showing that a relatively small number of "backhaul" links can greatly improve overall network energy efficiency. Others have developed hardware that supports multiple radios, with the goal of increasing reliable reception range [35] by exploiting channel diversity. The approach that we espouse maintains interoperability between the backhaul links and the links used by end-devices. This uniformity keeps hardware and software design simple.

2.3 Supporting Systems

In WSN literature, a deployment is "successful" if samples taken at nodes get to a database. In our experience, however, this is but one of the victory conditions. While some of this work may be considered "just engineering," we believe that our experiences should reframe what it means for a deployment to succeed, and the solutions which we develop can serve as a valuable example to future researchers in this area.

Our first "campaign deployments" [10] were intended to allow domain scientists to instrument a site and collect as much useful data as possible in a short (several week) time period. As such, they needed to get quick access to preliminary measurements and move sensors from point to point periodically. To some extent, the need for flexible data processing and agile hardware placement are necessary for a well-performing WSN. For instance, Ramanathan et al. envision tight interaction between researchers and their hardware in Suelo [51], incorporating expert knowledge from domain scientists in detecting and responding to sensor

faults.

The original Life Under Your Feet design philosophy was to make the network as "dumb" as possible, dealing only with opaque blocks of data and doing all data processing in a series of off-site databases. This is unsuitable for the normally-disconnected use case, where a researcher may need to have access to processed data in the field. In Section 6.3.3, we describe an improved, more modular data processing approach.

Tracking hardware as it's tested, assembled, deployed, and moved is a critical part of any successful deployment. Domain scientists receive streams of raw data from sensors that must at some point be interpreted as physical measurements taken at some point in space and time. Deploying a network involves painstaking notes of, at a minimum, what types of sensors are deployed at which locations. When this information is inaccurate or missing, we end up with useless data. If the ultimate goal of a WSN is to deliver scientifically useful data, then this is as bad an outcome as complete packet delivery failure. In our Brazil deployment, we had even stricter requirements, as the sensors in use were individually calibrated. Not only did we need to track what types of sensors were deployed where, but we also needed to provide a unique ID for each analog sensor to be used to apply its calibration.

In Section 6.3, we describe the mechanisms we have developed to record and automatically track sensor types, identifiers, and connections from assembly to deployment time.

2.4 Low Power Networking

Many instances of WSN deployments exist in the literature, and here we attempt to summarize the main elements of a "normal" data collection system. We also describe the key issue of maintaining good performance in the face of unreliable hardware and communication links, which will shape our approach to this application.

2.4.1 Existing WSN Systems

Most WSN systems for environmental monitoring move data from source to sink using fundamentally similar methods. A series of nodes determine their best choice of "next hop" through a combination of local link quality measurements and multi-hop distance metrics, and then strive to send data to that next hop without interfering with other transmissions and with a minimum of energy waste. This problem is generally decomposed into media access (relating to interference-prevention and coordinating between each sender/receiver pair) and routing (relating to the selection of paths in the network). While a wide variety of approaches have been taken to carry out the task of data collection, they rely on identifying useful individual links and transmitting data over a series of them.

Our research group's sensor deployments have, until this point, been based on the Koala [44] data collection system originally developed by Musăloiu-E. et al., implemented in TinyOS [23, 29] and running on the commonly-used TelosB mote [50]. Each deployment consists of a single base station node (a mote attached via USB to a laptop/PC) and a network of battery-powered nodes (mote + analog sensor assembly) in the field. Each node generates data periodically (e.g., by sampling external analog sensors) and buffers it in an external flash memory chip.

Nodes running Koala keep their radios off most of the time. Periodically, the gateway wakes up the entire network and downloads data from each node individually. Details of this process can be found in [44], but we'll point out a few of the most important characteristics here.

First, Koala uses centralized source-routing (where the base station dictates over which routes data should be forwarded) rather than maintaining persistent data collection routes (as in the commonly-used Collection Tree Protocol [24], MintRoute [71], and many others).

This is criticial to maintaining low power operation without reliable infrastructure. If the base station is absent, the network stays in a low-power state, as opposed to searching for a nonexistent route to the data sink.

Second, Koala avoids the need for a Media Access Control (MAC) protocol by centrally coordinating the in-network communication. A MAC protocol typically has to both avoid conflicts between users of the medium and ensure that a transmitter and its intended receiver can rendezvous (e.g., agree on a common radio channel and communication time). In Koala, the entire network stays active for the duration of a download, and the route provided by the base station dictates an inter-packet spacing time that aims to avoid intra-path interference.

Within the sensor network community, there are a few other notable networking solutions.

CTP [24] is the preeminent data collection protocol in the low power WSN community. In CTP, a routing tree is constructed in a distributed manner. Each node maintains an internal notion of its current parent, and they send their data and the data of their descendants up towards the root of the tree as it is collected. The largest "real-world" CTP deployment of of which we are aware is the GreenOrbs project [28, 40]. While the original CTP work demonstrated impressive agility and tolerance to node failures, the GreenOrbs authors describe networking issues that they see in large-scale networks: they specifically point to high levels of packet loss in nodes far away from the data sink.

6LowPAN [31] brings IPv6 to the low power wireless networking domain. 6LowPAN provides general-purpose data delivery service, not just data collection. RPL [70] describes the underlying routing method used for this task. Nodes arrange themselves into a Destination Oriented Directed Acyclic Graph (DODAG). A message is routed upward towards a root node until it reaches a node that knows the path to the destination. At this point, it is sent down over a series of links to the end point. RPL determines the graph structure on the basis of one or more objective functions, but the general goal of finding reliable links between devices

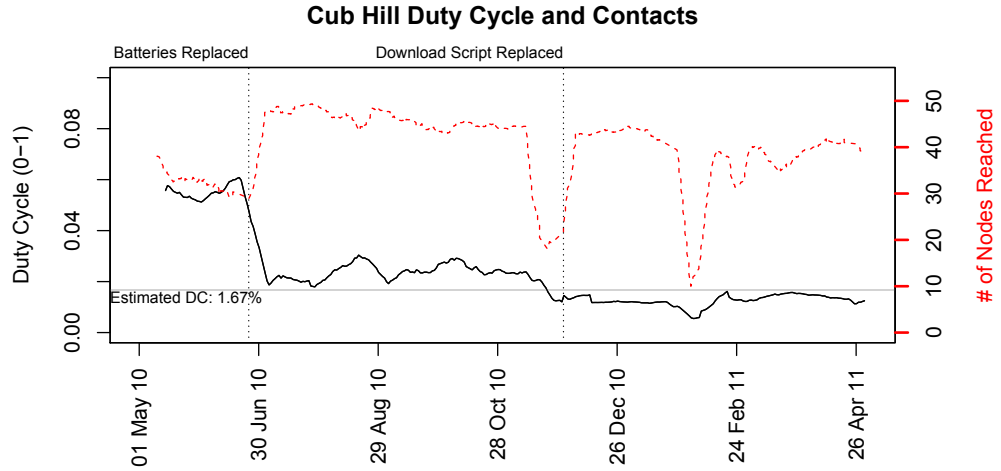


Figure 2.3: 14-day moving average of duty cycle and daily contact rates for an illustrative period of the Cub Hill deployment.

and maintaining some notion of a directed graph structure is similar to that of tree-based routing protocols.

The Backpressure Collection Protocol [42] implicitly defines the routing paths by using the depth of each node's packet queue as a decision criterion. As a node offloads its data, its queue length drops: this, along with estimates of per-link ETX (expected transmissions, the number of packets that must be sent before the first is successfully received) is used to set up a routing gradient towards the data sink. While the route-selection that this protocol realizes is less explicitly focused on the concepts of links and shortest paths than other methods, data still travels from one device to another over single links to a sink.

2.4.2 Stability, Routing, and Energy Usage

In order to select a good route, we have to pick a set of stable nodes between the source and destination, connected by reliable links. This can be remarkably difficult to do consistently due to the realities of long-term outdoor deployments and the limitations of single-path rout-

ing.

Figure 2.3 shows the high-level performance of Koala at our Cub Hill deployment for a period where the network size and mote software remained the same. For reference, the horizontal line shows the analytically-derived expected duty cycle based on data generation rates, estimates of network connectivity, and a detailed model of the protocol [11]. This site consisted of 50 nodes, more-or-less uniformly spaced apart: see Figure 2.1. The communication range of these nodes was generally greater than 20 meters, so the network was densely connected.

Prior to a network-wide battery replacement in June 2010 (the first vertical line in Figure 2.3), the actual radio duty cycle exceeded the expected duty cycle. During this period, the network was connected well enough for the base station to detect many nodes in the network, but it was unable to reliably find good routes to them. This results in failed and re-attempted connections, costing radio on-time and energy. While diminished connectivity certainly contributes to this problem, unreliable hardware is a major factor. Once a node's battery has been depleted to a certain level, it can no longer sustain the continuous current load of the active radio. However, it can send wakeup probes and operate for short periods of time, which leads it to be detected and considered for use in data forwarding. Figure 2.5 shows the voltage of partially depleted batteries under different levels of load, while Figure 2.4 shows the impact that brownouts have on route selection: connection attempts fail at higher rates when routes are chosen that contain unstable nodes.

While batteries will run down over time, it's also important to design protocols that default to low power states when failures occur. Figure 2.5 shows two incidents at our Cub Hill deployment where a bug in the download script kept the network awake for an extended period of time. After event A, Box 24 starts showing the signs of a depleted battery (large voltage drop under load), and it dies entirely during event B. Similar instances of basestation errors

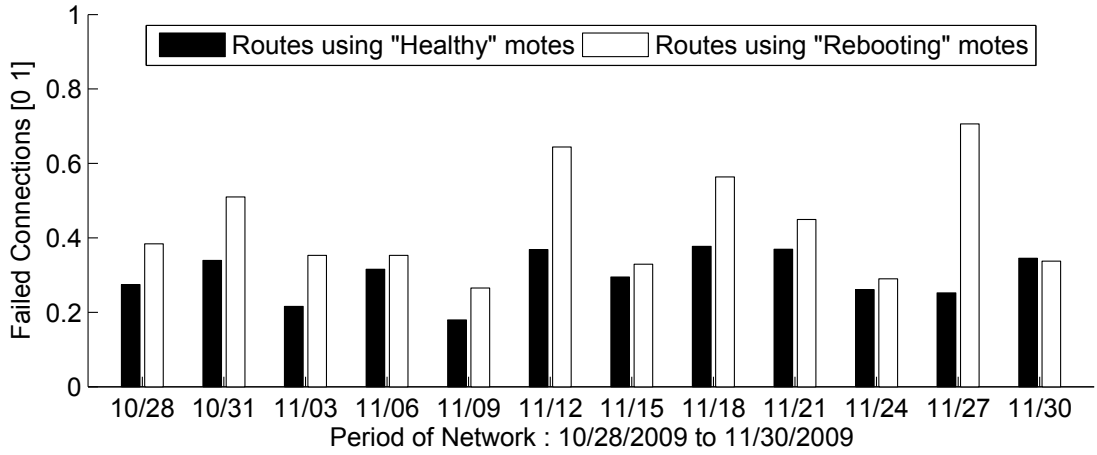


Figure 2.4: A comparison between the fraction of failed connections when using routes that include motes with high reboot rates ("Rebooting") in comparison to those using only motes with low reboot rates ("Healthy"). N.B. this data is from before the route selection method was improved in 2.3.

triggering cascading faults in the network were reported by the authors of SensorScope [32], where intermittent failures at a cellular uplink led to extensive packet queuing and data drops. Langendoen et al. [36] also describe a situation where failures at PC-class backbone nodes caused data loss and expensive (and useless) routing churn.

Figure 2.6 demonstrates the interplay of high reboot rates with duty cycle in detail, for an earlier period in the deployment. The top figure shows the overall duty cycle, normalized to the data collected (to account for changes in network size over time). We can see both a distinct increase in the normalized duty cycle over time and an increase in the variability of duty cycle as time goes on. Not only did the network become less efficient, it also became less predictable.

The most proximal cause for the increased duty cycle is the number of connection failures. Such failures increase as the network degrades, and this corresponds with the increasing reboot rate in Figure 2.6. We note that the daily reboots will peak and then go back down over time, as marginal nodes take longer to recover to operational voltages (rather than oscillating

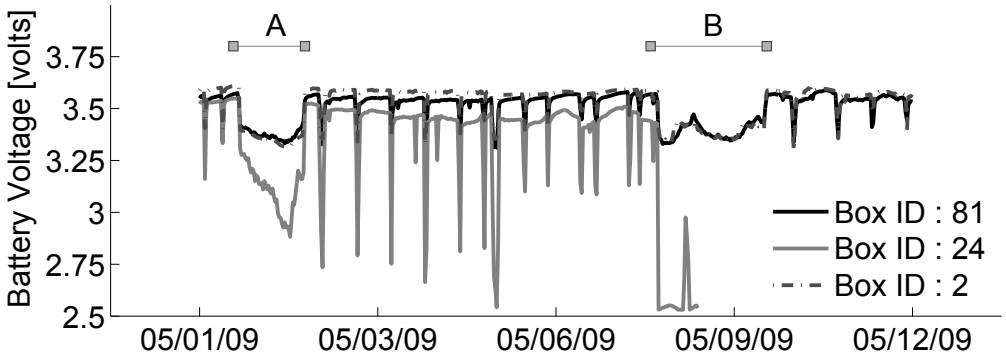


Figure 2.5: Impact of base station failure on battery voltage. Motes kept radios on during periods A and B. Drops in voltage correspond to download events.

around the brown-out zone) and eventually stop running entirely.

These sorts of problems are inherent to single-path routing: each route you select is an all-or-nothing bet, and the price you pay for failure ranges from time/energy wasted on retries to packet loss and routing failures (loops, etc.).

In November 2010 (the second vertical line in Figure 2.3), we modified the download process in an attempt to both improve route selection and mitigate the cost of individual connection failures, details of this are in [10]. We began using breadth-first downloads which favor recently-acquired link quality information over older information, and resulted in fewer attempts before each node could be reached. Since nodes could generally be reached in fewer attempts, we reduced the number of retries before "giving up" on a node until the next download. Finally, we weighted against nodes which were on a previously-failed download path in an attempt to weed out misleading link quality measurements and unstable nodes. Being more aggressive at pruning out potentially-bad information let us maintain roughly the same contact rate but at a much lower duty cycle.

It is somewhat unsatisfying to conclude that the best that we can do with a centralized and reasonably up-to-date view of the network is to give up quickly on nodes that are hard to

reach with the best information we have available, and hope that we can do a better job next time. So how does one improve from here?

The available options we see are to either do better at route selection or to move from all-or-nothing routes to something that is more resilient to failures.

Improving route selection would involve gathering better link quality information, explicitly incorporating non-link sources of route failures into the selection logic (battery, moisture, etc), or both. Link quality information is not free: you need some radio traffic to measure in order to assess a link. Adding more considerations to route selection increases logic and seems like a band-aid approach. Changing temperature affects battery voltage, so should that be included too? How does a node actually detect that it has an unstable neighbor? What if none of a node's neighbors are consistently stable, but there's generally at least one that's stable on a short term basis?

Rather than chase down every source of error in route selection or link quality measurements, we will seek a method for adding redundancy to routes.

Recent work has explored softer definitions of routes in low power networks. In opportunistic routing (for example, [16]) packets are allowed to travel over less-reliable but longer-range links. Even more radically, systems such as Flash Flooding [41] and the Low-Power Wireless Bus [21] do away with routing entirely by communicating via efficient network floods. We see these approaches as taking important steps to addressing the problems arising from fragile links and forwarders. In the next section and the following chapters, we will apply these concepts to build a disconnection-tolerant collection protocol that uses reliable sets of forwarding nodes rather than potentially-shaky single-path routes.

2.5 A New Direction

So where do these observations leave us?

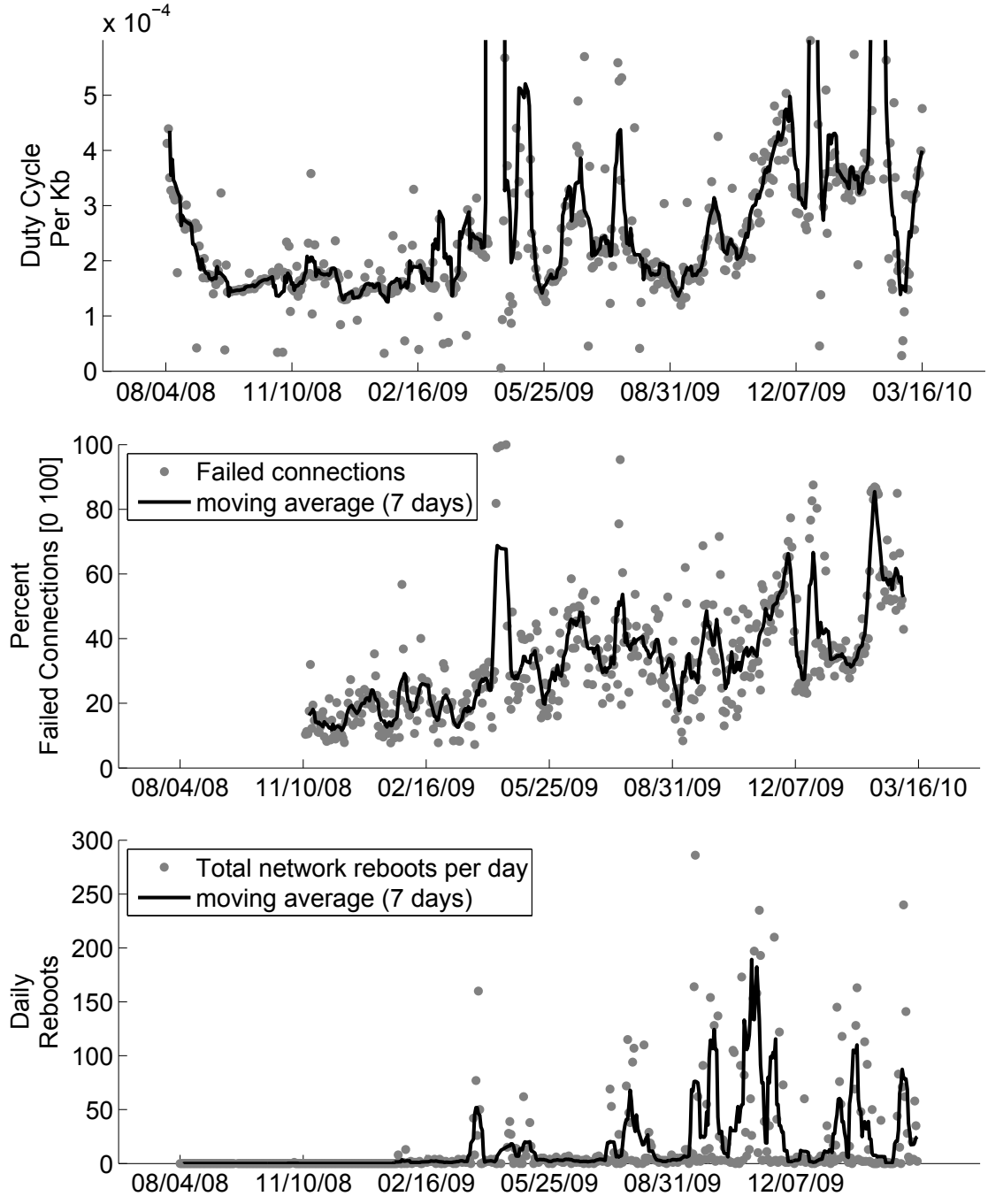


Figure 2.6: Correlation between duty cycle (per unit data), reboots, and retransmissions in the Cub Hill deployment. A series of base station faults explains the high duty cycles in 4/09, where almost no data was retrieved.

We want a system that matches sensor deployment needs more closely than our current approach. Organizing nodes into patches and attaching many analog sensors to a single node will improve both energy efficiency and manageability. Automating the discovery and identification of analog sensors will improve the yield of scientifically useful data and lead to more dynamic deployments. This is described in the details of our hardware design and supporting software systems in Chapter 6.

We want a collection protocol that operates in a low power state as much as possible. Koala starts down this road, by leaving nodes idle until a download is required. We want to take this a step further by decoupling each patch from the rest of the network: nodes should be able to enter a low power state whenever there is not an active download in their immediate vicinity. This is reflected in the design of our multi-tiered collection system described in Chapter 5.

Finally, we want a system that is robust to the common networking problems arising from low-power, single-path routing. We will realize redundant simultaneous delivery paths by leveraging non-destructive interference. Under this approach, we aim to identify a subset of the network that roughly lies between the source and destination and use them all for forwarding. By carefully scheduling transmissions to occur simultaneously, we can send a data packet over this set of nodes without fear of losses due to interference.

2.5.1 Taking advantage of non-destructive interference

The principle of non-destructive radio interference is well-suited to both the spatially patchy nature of our deployments and our desire for robust delivery methods. Here we briefly explain non-destructive intereference.

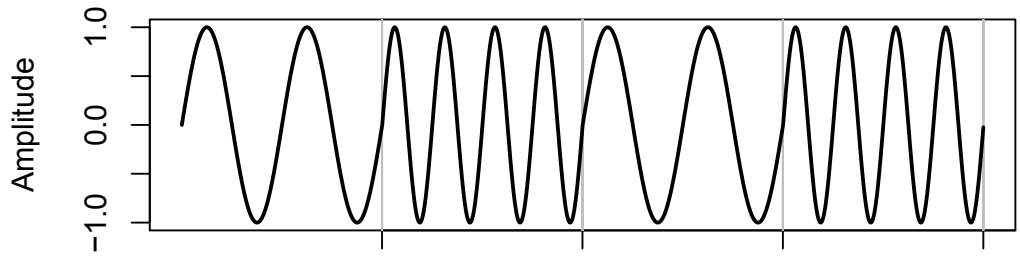
Drs. Musăloiu, Dutta, et al. noted an unexpected benefit in the original Koala system which leads us down the road to such a method [19]. During the Koala wake-up process,

nodes send probes periodically and interpret an acknowledgement to one of these probes to mean "stay awake." However, once multiple nodes are awake, one would expect that their acknowledgements would interfere with each other (and so the prober would not wake up). This does not occur, however, because the packet acknowledgements collide non-destructively. They are sent so close in time to one another that their combination results in a waveform that can still be demodulated successfully.

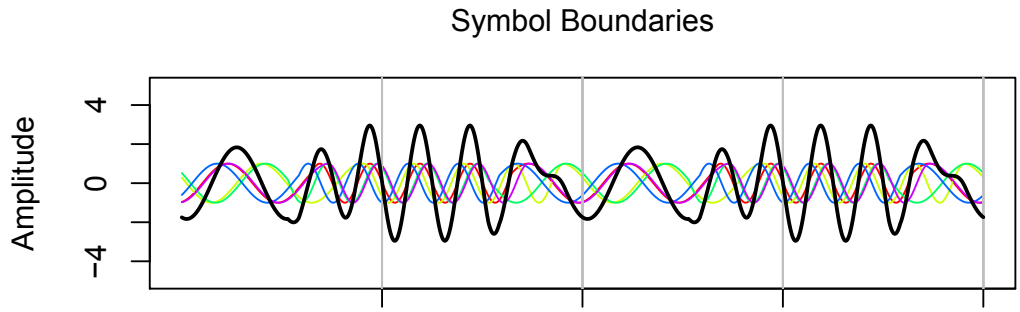
Figure 2.7 shows an idealized view of the effect of multiple non-destructively interfering transmitters. When transmissions are tightly aligned in time (to within a fraction of the underlying signals' chip frequency [22]), nodes can decode the combination of these transmissions. This process is aided by the presence of radio capture effect, in which the gain control of a receiver effectively drops the weaker of two simultaneous and interfering signals below the noise floor relative to the stronger signal.

This behavior was used to build a MAC protocol in [18], and we apply this to use in extremely dense network conditions in chapter 3.

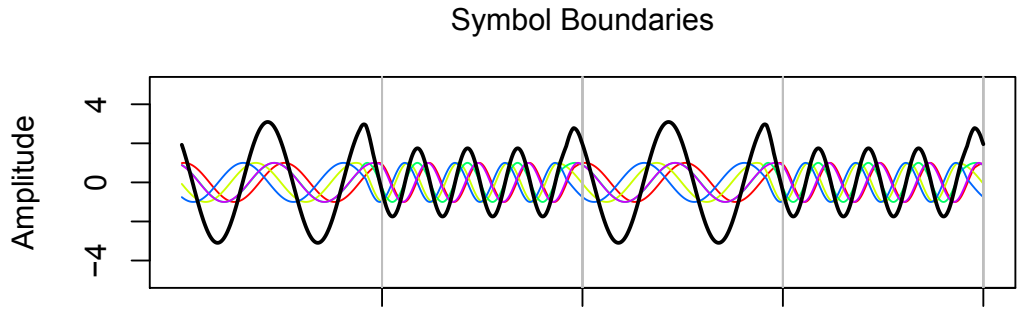
It was not until recently [21,22,54] that this technique was applied to *data* transmissions in low power WSNs in industrial and academic research. In Chapter 4 we will apply these findings to build a system which identifies a useful subset of the network and uses efficient floods over just this portion of the network to deliver data.



(a) Single FM transmission.



(b) Five unaligned transmissions. Note distortions at symbol boundaries.



(c) Five concurrent transmissions with random phase differences.

Figure 2.7: Idealized results of multiple concurrent transmitters interfering with each other. Vertical lines indicate symbol boundaries and each original signal has an amplitude of 1. When the FM signals shift from the 0-frequency to the 1-frequency at the same time, the result is a wave with the same frequency as a single transmitter and varying amplitude.

Chapter 3

FlipMAC: Building a MAC Protocol on Non-Destructive Packet Collisions

Our first effort into using concurrent transmissions to solve real problems in sensor networks was Flip-MAC. We noted that the region surrounding a router node was likely to be quite dense relative to the rest of the network. We wished to explore methods for efficiently resolving contention between a large set of nodes attempting to reach a single destination. The method described in this chapter uses a series of request-acknowledgement rounds to winnow down a large set of possible senders to a small set of competing nodes. It makes use of concurrent transmissions to carry out this contention-reduction process in logarithmic time relative to the initial number of senders.

While it is not directly applicable to our particular target application (in which we require long-term, low temporal resolution measurements from *all* nodes), it may be well-suited to similar network topologies having different data delivery needs. It merits inclusion in this thesis not only as a record of the scientific process leading us to our final system design, but also as a potentially valuable tool for similar classes of sensor networks.

This work followed in the footsteps of A-MAC and BackCast [17, 19] , and was indirectly inspired by the observations of non-destructive ACK collisions in the Koala LPP wakeup process [44].

3.1 Introduction

Many wireless sensor networks (WSNs) are deployed in dense patterns, where a single node can have dozens of radio neighbors [39]. The advent of long-range radios is likely to create even denser deployments, with hundreds of nodes within radio range of each other.

In this chapter, we consider the case of a dense network of sensing nodes that continuously record high-resolution data, most of which is "uninteresting." Periodically, some event occurs which is detected by multiple nodes. When this happens, an infrastructure node wants to quickly become aware of it and retrieve the high-resolution data from *any* of the detectors which experienced the event. Namboodiri et al. described a similar use case in the context of a home security monitoring application, while we add the assumption that it may not be necessary to service *all* of the nodes with data to send [45]. Due to the relative rarity of events, we would prefer to use a non-scheduled MAC protocol to handle the channel negotiation process, which generally will have lower maintenance costs than a scheduled protocol.

Rather than proposing a complete MAC protocol to fit this use case, we propose dividing the problem into two steps: the first quickly reduces contention to a manageable level, and the second uses an existing non-scheduled MAC protocol to select a single sender.

In Flip-MAC, our goal is to quickly reduce the level of contention from many senders to a handful. The process we follow is analogous to all nodes performing a series of coin-flips, where the senders whose coins fail to match the receiver's simultaneously exit the competition. The competition ends when no senders match the receiver (as this indicates that relatively few senders remain).

This process is realized through a series of probe-acknowledgement cycles. During the main negotiation process, the receiver sends a probe to one of two possible 802.15.4 addresses, while each eligible sender temporarily sets their ID to one of these two addresses. These

choices are made randomly at each node. The eligible senders which "guessed correctly" (by setting their ID to the destination of the probe) send simultaneous acknowledgements and make their selection for the next cycle. When one of these probes goes unacknowledged, the receiver assumes that none of the remaining senders made a matching selection, which indicates that only a few senders remain in negotiation. At this point, we revert to a simple CSMA/backoff scheme and initiate the main data transfer with *any* one of the remaining nodes.

Most Medium Access Control (MAC) protocols make the assumption that collisions between transmitters are solely destructive. It has been demonstrated that this is not always the case [17, 22]. In Flip-MAC, the concurrent transmissions are limited to hardware-generated acknowledgements provided by the CC2420 [62] transceiver. These are implicitly synchronized and are produced in a highly deterministic fashion, so they interfere constructively in most cases. While Flip-MAC is designed with the CC2420 in mind, it makes use of principles which should apply to any 802.15.4-compliant transceiver.

Flip-MAC has three main benefits. First, negotiation takes logarithmic time with respect to density, allowing the protocol to scale to extremely high levels of initial contention. Second, final contention levels are low and largely independent of the initial contention level. Finally, the incorporation of several two-way communications in negotiation helps to bias sender selection in favor of good links: in our intended use case, any sender is equally valid, so this is a positive outcome.

Our experiments yield promising results. A testbed evaluation demonstrates that Flip-MAC works effectively with up to 44 contending senders, completing negotiation successfully more than 95% of the time. These experiments also show an increase in median negotiation time from one round to five rounds as density increases from one sender to 44, which agrees well with the expected ideal logarithmic performance. Simulation results show that Flip-

MAC should perform well in large networks, effectively reducing the median final contention by an order of magnitude even when low-PRR links are present.

The rest of this chapter is organized as follows. In Section 3.2, we give a brief survey of related systems and MAC protocols, highlighting the important differences of Flip-MAC from them. In Section 3.3, we provide a detailed description of the operation and expected behavior of Flip-MAC. We evaluate Flip-MAC's performance in Section 3.4. In Section 3.5 we propose some extensions to this research, and we offer concluding remarks in Section 3.6.

3.2 Related Work

A staggering number of papers have been written about MAC protocols. Rather than attempt to describe all of them, we wish to point out several protocols and systems which are evocative of Flip-MAC: for a comprehensive overview, see [6].

Receiver-initiated MAC protocols were perhaps most famously explored in RI-MAC [57]. Nodes with data to send wait until the intended recipient sends a beacon frame. All waiting senders respond with data, and if a collision occurs, the receiver sends another beacon with a "backoff-window" specified. Senders attempt to retransmit their data at some random point in this window.

A-MAC [17] is another receiver-initiated MAC protocol, but it takes advantage of the fact that hardware-generated 802.15.4 acknowledgement frames can be sent and processed much more quickly than full 802.15.4 data frames to reduce wasted idle-listening time. A-MAC also takes advantage of the fact that hardware acknowledgments collide constructively to "robustly distinguish the case of zero replies (indicating no pending traffic) from one or more replies (indicating pending traffic)." The authors demonstrate impressive gains in current consumption over RI-MAC and Low-Power Listening (LPL) [7], but point out that limitations remain, particularly in the realm of dense networks. Additionally, their method is only eval-

uated up to densities of four contending senders (though they show that acknowledgements are still readily decodable with 94 simultaneous senders). Like any protocol which relies on detecting collisions and backing off, as contention increases, backoff times suffer. We see A-MAC and Flip-MAC coexisting well together, as they leverage the same communication primitives.

The StrawMAN [47] system is reminiscent of Flip-MAC, in that they both allow multiple transmitters to coexist and make use of randomized negotiation. In StrawMAN, all pending senders transmit a packet of variable length, while the receiver reads the Received Signal Strength Indicator (RSSI) to determine the length of the longest transmission. The receiver then broadcasts a message containing the length of the longest sender-request packet, and that sender responds by transmitting its data. However, the authors of [17] point out that pollcast [14], implemented with a similar technique, can be prone to false positives when multiple nodes perform this activity or external interference is present.

The authors of Alert [45] attempt to solve a similar problem to ours (selecting a single sender from a pool of many eligible senders), but they approach it through a randomized channel/time slot assignment that is optimized to minimize the delay of the first message and the overall delay to collect all messages. Our approach does not require any synchronization between nodes, has channel utilization dictated primarily by the receiver's latency requirements, and is simpler to analyze.

Finally, while not a MAC protocol, Glossy [22] was recently introduced to simultaneously solve the problems of efficient network flooding and time synchronization. Like Flip-MAC, Glossy relies on non-destructive concurrent packet transmissions to achieve high performance. The authors of Glossy take considerable care to transmit data packets (not acknowledgements) with very precise timing, while we use the basically "free" precision of hardware-generated acknowledgement. Their work is indicative of the nascent trend in WSN research

of exploiting concurrent radio transmissions.

3.3 Protocol Description

Most non-scheduled MAC protocols rely on collision detection and backoffs at some point: in both A-MAC and RI-MAC, senders back off in response to collision-detection messages from the receiver, for example. When contention is very high, these backoffs can hurt the MAC protocol's performance. In the extreme case, hard-coded maximum backoff limits will cause receivers to abort when contention is too high. Flip-MAC is used to quickly reduce contention to low levels, at which point traditional MAC protocols can work effectively.

The high-level operation of Flip-MAC uses a series of probes to randomly select nodes from a pool of eligible senders. Each probe is sent to a random ID from a small range of choices, and any senders[1] which correctly guess the ID selected (by setting their ID to it temporarily) remain in negotiation. The selected senders respond with acknowledgements, which collide non-destructively and allow the receiver to tell whether nodes are still participating in negotiation. This removes some fraction of the remaining senders on each negotiation round. In this manner, we quickly reduce the number of eligible senders to the point where a basic MAC protocol will perform well. We reserve a few bits from the 802.15.4 address to allow the negotiation to take place solely through transmissions from the recipient and fast, non-interfering hardware acknowledgements from the senders.

In this section, we describe the protocol in detail and characterize its behavior under loss-free conditions.

[1]Unless otherwise noted "sender" refers to a node which ultimately wishes to send data to the "receiver" (not to be confused with the node which is transmitting a Flip-MAC control packet).

Abbreviation	0	1	2	Meaning
Not Used	0	0	0	Reserved for normal data traffic
DP	0	0	1	Is Data Pending?
NC_0	0	1	0	Negotiation choice: 0 selected
NC_1	0	1	1	Negotiation choice: 1 selected
RC_0	1	0	0	Resolution confirmation: 0 selected
RC_1	1	0	1	Resolution confirmation: 1 selected
RC_x	1	1	0	Resolution confirmation: none selected

Table 3.1: Predicate Address prefixes. These are prepended to the unique 13-bit ID of a node to form a predicate address.

3.3.1 Operation and Implementation

In our scheme, nodes may only use the lower 13 bits of their 16-bit 802.15.4 short ID for unique identification. The three highest-order bits are reserved for encoding "predicate addresses" (PA's) used to control Flip-MAC and select from eligible senders. Table 3.1 describes the format used.

Figure 3.1 demonstrates the negotiation process in detail. Nodes with data to send to a specific recipient change their 802.15.4 ID to match the "data-pending" (DP) PA of the recipient. Nodes periodically probe to their own DP PA. If a sender receives a probe on this address, it acknowledges it and randomly selects one of the "negotiation-choice" (NC) PA's for the recipient and sets its 802.15.4 address to this. Likewise, if a receiver gets an acknowledgement to a DP probe, it randomly selects one of its NC PA's and sends a probe to it. This process continues: every time that a sender receives a probe, it assigns itself to a new NC PA. Every time that the receiver gets acknowledgements, it probes to a new NC PA. In the basic case, we consider a set of two NC PA's (requiring a single bit), but a larger number could be used at the expense of consuming more IDs.

If a sender doesn't get a probe within the alloted time, it assumes that it picked incorrectly, and changes its address to the "resolution-confirmation" (RC) PA corresponding to its last correct choice. The sender "hopes" that it was in the last batch of "winners" and waits for

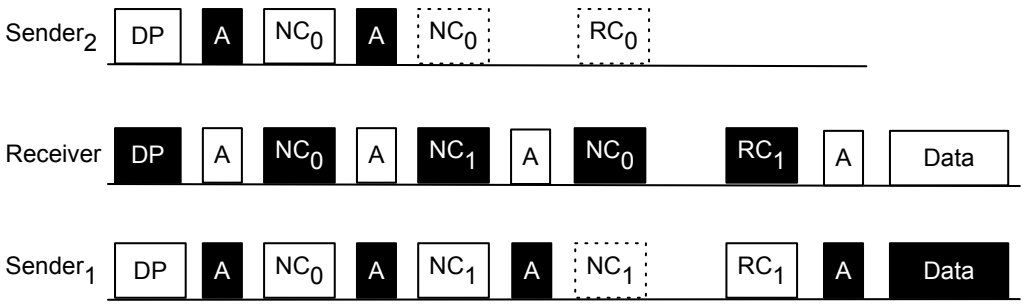


Figure 3.1: Negotiation sequence. Black boxes indicate transmissions, with text indicating the PA to which a probe was sent. White boxes indicate receptions, the text within indicates the PA at which the receiver was listening. Broken lines indicate that nothing was received due to a PA mismatch.

immediate confirmation. If this confirmation doesn't arrive, it concludes that some other node has been selected and stops participating[2].

If the receiver gets no acknowledgements, it assumes that no nodes matched its last selection (implying that relatively few senders remain). Once this state is reached, the receiver sends a probe to the RC PA corresponding to the *last acknowledged* NC probe that it sent.[3] When this is acknowledged, the behavior reverts to the low-contention MAC protocol in use.

Flip-MAC was implemented in nesC [23] for TinyOS on the TelosB platform [43]. Since it's ultimately intended to coexist with another MAC protocol, great care was taken to make minimal component wiring changes: the only major logical change in the CC2420 radio stack is the replacement of the `CC2420CsmaC` component with a `CC2420FlipMacC` component.

We used a simple CSMA + exponential backoff scheme for the low-contention MAC protocol. This code, including full instrumentation for debugging and data collection, consumes approximately 24 KB of ROM and 3 KB of RAM.

[2]We consider the problem of notifying senders that their data is no longer desired to be orthogonal to this work. One 3-bit PA prefix is not in use, so this could be added as a "cancel pending transmission" message in the future.

[3]In the case where the DP probe was acknowledged, but not the first NC probe, a special RC_x PA is used.

3.3.2 Convergence Behavior: Loss-Free Setting

In the absence of packet loss, Flip-MAC can be characterized by answering two key questions. First, how many rounds of negotiation are required before the receiver ends the process? Second, how many senders will remain at the end of the negotiation process?

Negotiation Length

We start investigating negotiation length by analyzing how many senders will remain after k rounds.

For generality, we let p_{sel} equal the probability that a sender selects the same NC PA as the receiver, and we assume that the selection of NC's is independent for each node. The probability, then, of a sender selecting the same NC as the receiver k times is simply p_{sel}^k. We can therefore define the number of nodes remaining at the k^{th} round, given an initial number of senders n, with a simple binomial distribution:

$$R(n, k, p_{\text{sel}}) = B(n, p_{\text{sel}}^k)$$

The negotiation process ends when a probe is sent, but no acknowledgements are received. If m senders remain, this is equal to the probability that all m senders fail to pick the same NC PA as the receiver, and given by $P_{\text{na}}(m)$ (the probability of no-acknowledgements received from m actively-negotiating nodes):

$$P_{\text{na}}(m) = (1 - p_{\text{sel}})^m$$

Combining these, we get $P_{\text{s}}(k, n)$, the probability of stopping on the k^{th} round, when starting with an initial density of n:

$$P_{\mathrm{s}}(k,n) = \sum_{j=0}^{n} r(j,n,k-1,p_{\mathrm{sel}}) P_{\mathrm{na}}(j)$$

where $r(j,n,k,p)$ is the probability mass function of $R(n,k,p)$. In plain English, the probability of stopping at the k^{th} round when starting from an initial density of n nodes is given by summing the probability of having exactly j nodes at the preceding round, and having all j nodes pick incorrectly, for all values of j up to the original density.

Figure 3.2 shows the cumulative density function that this produces for several initial contention levels, as well as the results from simulating this process directly with $p_{sel} = 0.5$. There is very good agreement between the analytical result and the simulation. Note that, as one would expect, doubling the sender density adds a single round to the negotiation length.

Sender Density Post-Negotiation

The negotiation process ends when none of the eligible senders select the same NC PA as the receiver. Aside from the time required to reach this point, we are also concerned with the final contention level.

The expected value of the number of senders participating in a no-acknowledgement round can be taken from our definition of $P_{\mathrm{na}}(m)$, with initial sender density n.

$$E(n_{\mathrm{na}}) = \sum_{i=1}^{n} i \cdot P_{\mathrm{na}}(i)$$

This examines the probability of stopping with i nodes in contention for every i up to the maximum possible, n. The expected value converges to two as n increases with $p_{\mathrm{sel}} = 0.5$. Simulation results bear this out.

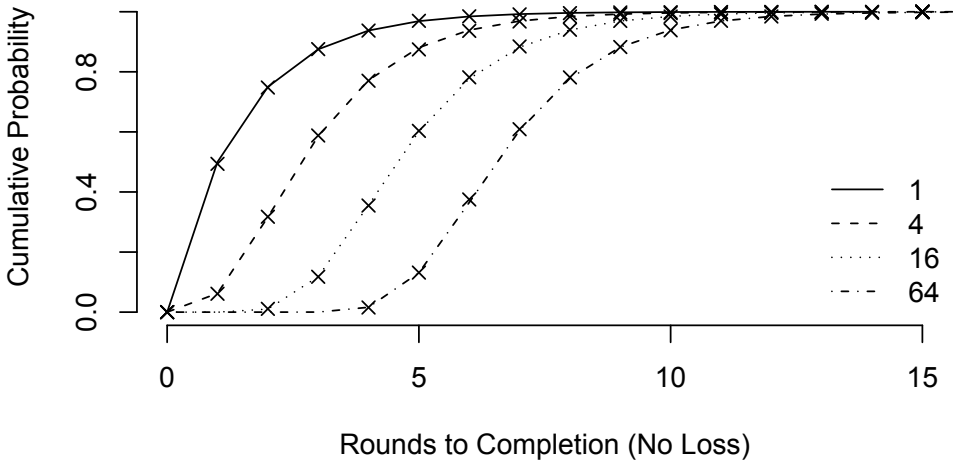


Figure 3.2: CDF of rounds to completion for several initial sender densities, from simulation and definition of $P_s(k, n)$ The curves are from the analytical results, the points are from the corresponding results of 10,000 simulation trials.

3.4 Evaluation

In this section, we seek to characterize Flip-MAC's performance under a variety of conditions. First, we evaluate Flip-MAC's behavior in absolute terms on a moderately-sized testbed. We then turn to a Python simulation in order to evaluate how correlated and uncorrelated packet losses affect Flip-MAC. Finally, we explore the beneficial positive bias which Flip-MAC exhibits in link selection at large scales.

Unless otherwise noted, all figures referred to in this section show the 10^{th}, 25^{th}, 50^{th}, 75^{th}, and 90^{th} percentiles of measurements.

3.4.1 Testbed Evaluation

Procedure

In order to evaluate Flip-MAC, we ran a series of testbed experiments over the course of 40 hours. One node was selected to be the receiver, and a variable number of nodes were selected

to act as senders. Every two seconds, the receiver sent a DP probe, and all senders attempted to send a data packet to it. When each round was finished, every node logged the results to the serial port: their final status, timestamps from each packet reception/transmission, and the RSSI/LQI of the last packet which they received in the round.

The receiver was set to a centrally-located node[4] on a two-floor testbed inside of a university academic building, consisting of 60 nodes total. We ran multiple 15-minute experiments with progressively larger sections of the testbed: only the closest node, the nodes in the same room, the nodes on the same hallway, the nodes on the same floor, and finally the nodes on the entire testbed. The maximum possible contention was 59. However, since not all nodes were in communication range, we will refer to the sets of senders by the median initial contention level observed over the experiments rather than the number of nodes which were designated as senders. The median initial contention level was 44 for the full testbed. Approximately 28,000 individual probes were sent over approximately 60 test batches.

Proof-of-Concept

The first question we need to answer is "Does Flip-MAC work in practice?" Figure 3.3 gives a high-level answer. When we aggregate results based on the portion of the testbed in use, we can see that while the initial contention rises from one to 44, we see the negotiation failure rate go from close to zero to almost 5%. These failures can be categorized as "RC Failures," where the final RC probe was not acknowledged, or "DP Failures," where the initial DP is not acknowledged.[5] We suspect that the loss increases because as we expand the pool of senders, we add progressively worse and worse links (farther away from the receiver). We note that

[4]We had intended to repeat this with different receivers, but the other nodes we tried experienced serial communication issues which made the data unusable. We discarded test results where we detected that the receiver had experienced poor serial communication.

[5]In cases where negotiations ended successfully, the simplistic CSMA protocol we used achieved a 97% delivery rate. A more sophisticated protocol, or one that included retransmissions, would likely do better.

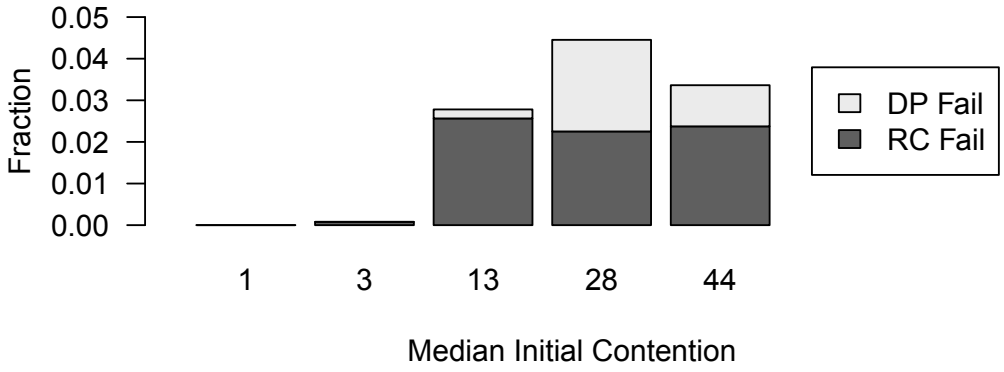


Figure 3.3: As the initial contention increases, we see an increase in negotiation failures. However, non-acknowledged DPs (which are most impacted by initial contention) account for only 2.2% of outcomes in the worst case.

prior work has shown that ACKs can be decoded robustly under good link conditions for up to 94 nodes [17]. By using no routing protocol and attempting to use every possible link on the testbed, we are working in a worst-case environment, but we still see more than 95% of negotiations end successfully. In practice, a good routing protocol would avoid including poor contenders in the initial pool, mitigating this problem somewhat.

Figure 3.4 further suggests that link quality variations contribute to negotiation failures. The negotiation failure rates of the multi-room experiments (13, 28, and 44) trace roughly similar shapes, though higher contention levels are impacted more heavily. The worst performance occurred during daytime hours on the second day of testing.

Contention Reduction

Flip-MAC is primarily intended to reduce contention from high levels to manageable ones. Figure 3.5 demonstrates its effectiveness in this task. The median final contention remains

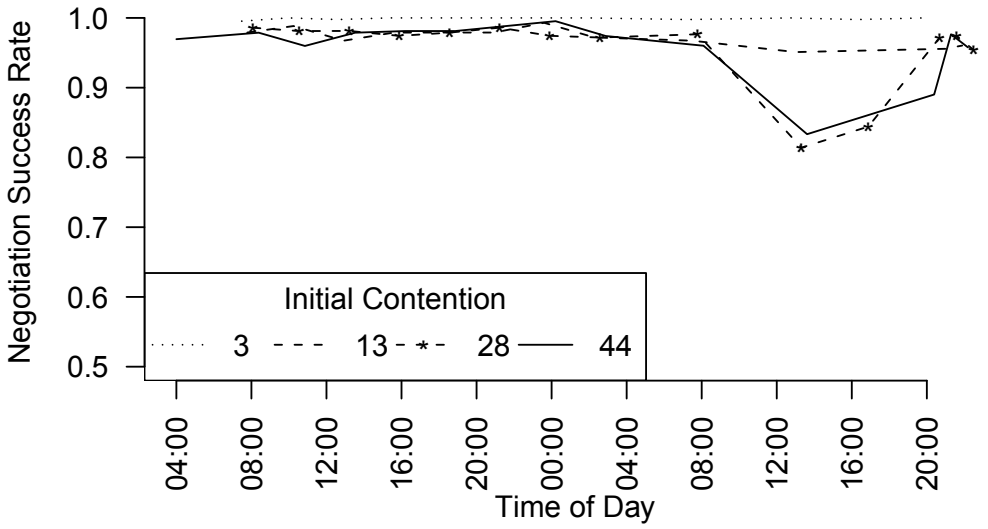


Figure 3.4: Testbed delivery rate as a function of time, for the initial contention levels tested. Note Y-axis scale.

at or below two. While we see fairly long tails at the highest contention levels, we note that the 75thpercentile remains below seven, which is easily manageable by most MAC protocols.

Flip-MAC's expected logarithmic running time is one of its primary benefits, and Figure 3.6 shows rough agreement with the analytical results. There aren't measurements at very large scales to definitively bear out the analysis, but we can see that the median number of rounds required only rises from one to five as we increase the contention from 1 to 44 nodes.

Time Overhead

In order to determine the overhead required in negotiation, we measured the difference between when a receiver sent their first DP probe to the point where negotiation completes and nodes begin to send their data with the 32KHz on-board clock. We tuned the inter-round interval to 16 milliseconds, which allows enough time for senders to consistently receive a probe, send their acknowledgement, and switch to a new predicate address in time to receive

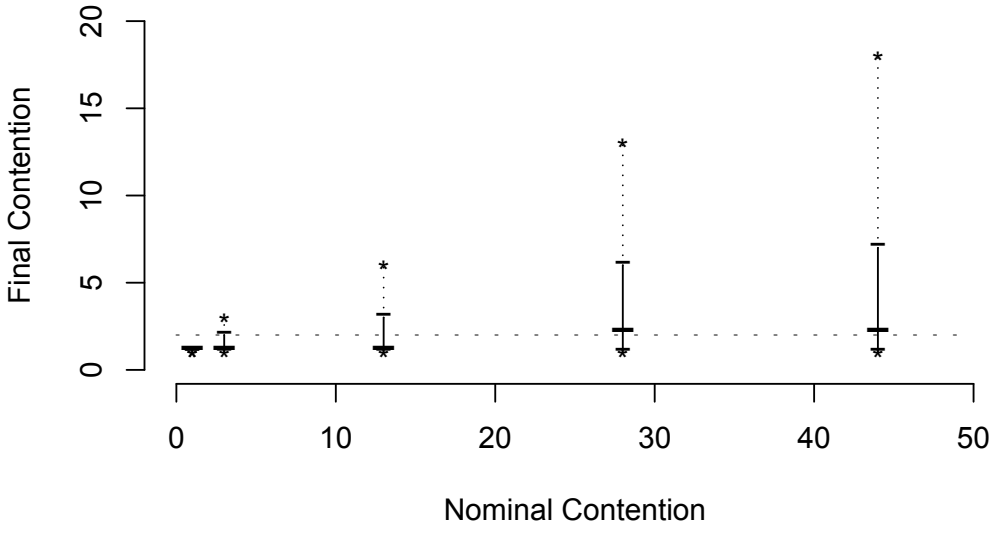


Figure 3.5: Results from testbed experiments demonstrating Flip-MAC's ability to reduce contention. The median final contention remains at or below two (marked with horizontal line).

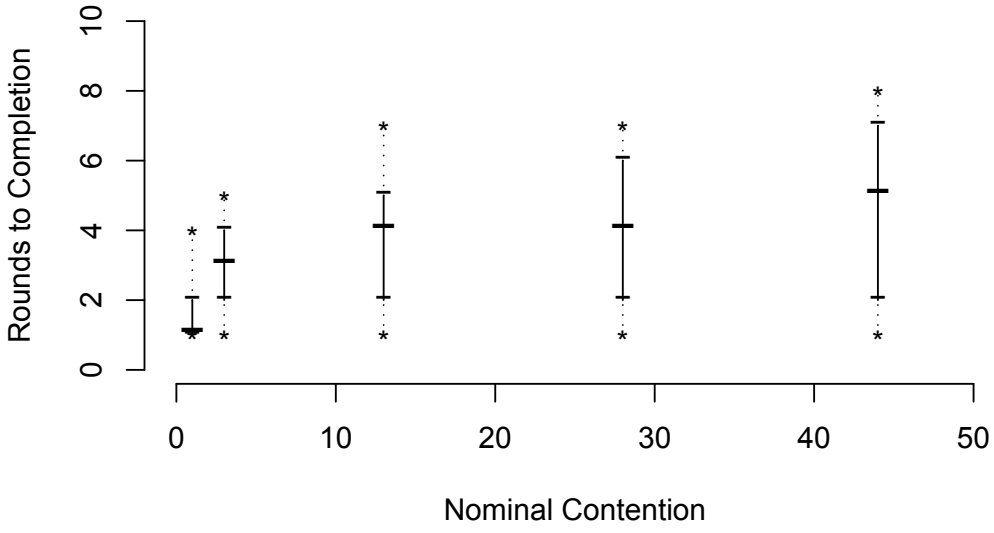


Figure 3.6: Results from testbed experiments showing relationship between rounds-to-completion and initial sender density. The median rounds-to-completion only rose from 1 to 5 as we increased contention from 1 to 44 nodes.

the next probe. In addition to this, approximately 1.6 milliseconds of overhead are involved in sending the initial DP probe, and the final RC probe consumes another one-round interval. From the results in Figure 3.6, we can expect less than 66 milliseconds of overhead in 90% of the cases where no contention is present. Our implementation attempted to keep the CC2420 radio stack as clean as possible. It's certainly possible that if the lower layers of the stack were to be re-written with Flip-MAC in mind, it would be possible to reduce the period between probes. However, this optimization runs counter to the idea that Flip-MAC could supplement low-contention MAC protocols.

3.4.2 Simulation

We wrote a simulator in Python to help us assess Flip-MAC's performance in a more controlled environment than the testbed provides. In the remainder of this section, we present descriptions of how the simulation indicates that Flip-MAC will behave when subjected to different types of packet loss and how this impacts sender selection bias. Unless otherwise noted, each experiment simulated 128 senders, and each set of error-bars depicted is derived from the results of 1,000 trials.

Correlated Loss

The first type of loss we are concerned with is correlated loss, where a probe is lost to all senders or an entire set of acknowledgements is lost to the receiver. This could occur, for example, if a burst of WiFi traffic or some other interference is present during negotiation.

If a probe is lost to all senders, no acknowledgements will be sent in response. The remaining senders from the previous step and the receiver will therefore both move on to the resolution-confirmation phase. This has the effect of shortening the negotiation process, but

potentially leaves many nodes in contention when it's complete.

If all acknowledgements are lost, the situation is slightly more complicated. When this occurs, some senders receive a probe, send their acknowledgements, and move on to the next round of negotiation. However, the receiver assumes that no nodes responded to its last probe, and moves on to the resolution-confirmation stage. At this point, it's possible that nodes which *failed* at the last negotiation round are currently expecting an RC message, and nodes which succeeded are waiting for an NC message. It's also possible that no nodes are waiting for an RC message at this point. We accept that such failures can occur and will simply retry the negotiation when they do.

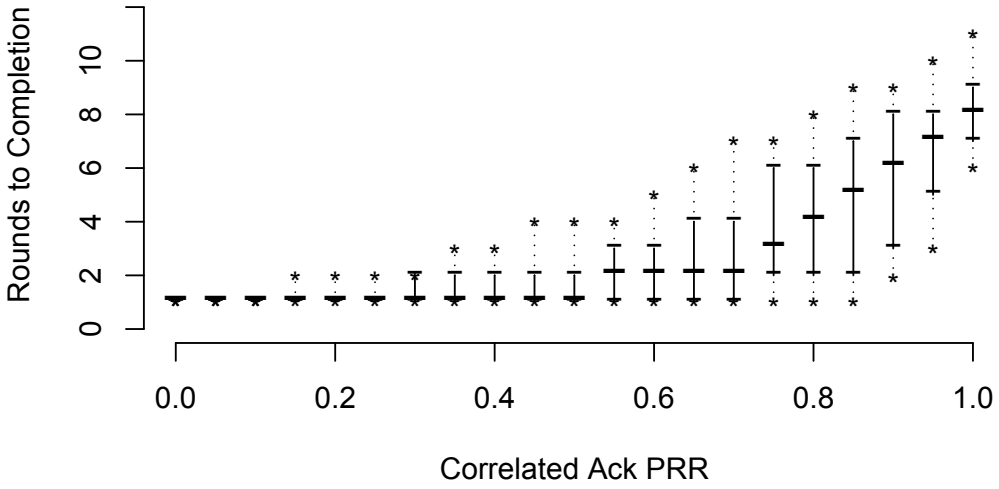
We investigated the impact of correlated loss by simulating the negotiation process with 128 initial senders and a correlated acknowledgement PRR which varied from 0 to 100%. Varying the correlated probe loss rate elicits a similar response, so we omit those results for brevity.

Figure 3.7 confirms our expectations: a low acknowledgement PRR leads to very short negotiations, but can potentially leave very high levels of contention. We leave it to future work to make Flip-MAC more resilient to correlated losses. This could be achieved in practice by incorporating retries into the negotiation process, or allowing the low-contention MAC protocol to coordinate quickly restarting negotiations with Flip-MAC if contention is too high to work effectively.

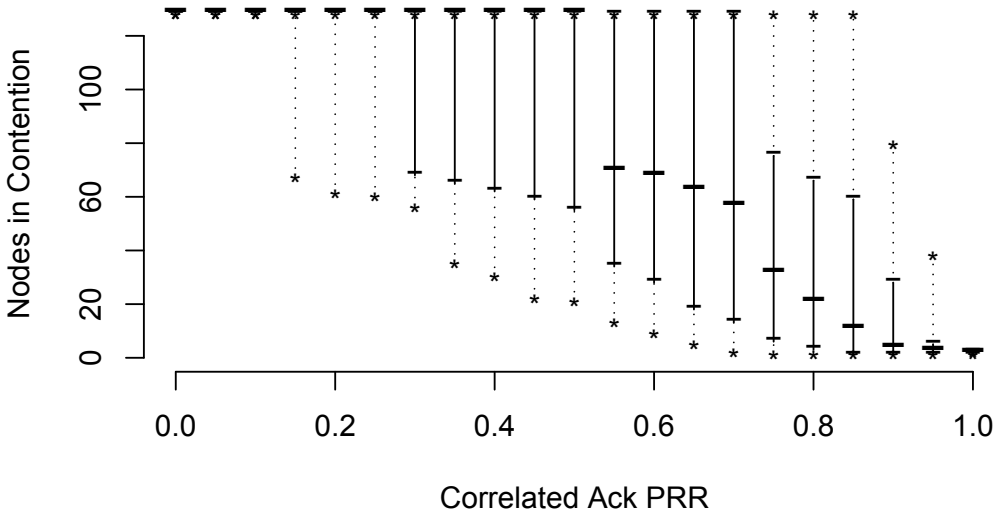
Uncorrelated Loss

The second type of loss we are concerned with is uncorrelated loss, where each sender's link experiences loss events independent of the others (due to different distances, sources of interference, etc).

The impact of probe loss is intuitively simple to understand in this setting: in order for



(a) Losing all acks stops negotiation immediately. When ack PRR is low, most negotiations do not progress very far.



(b) If acks are not reliable (e.g. due to persistent interference at the receiver), Flip-MAC is not able to effectively reduce the level of contention.

Figure 3.7: Impact of correlated acknowledgement loss on performance.

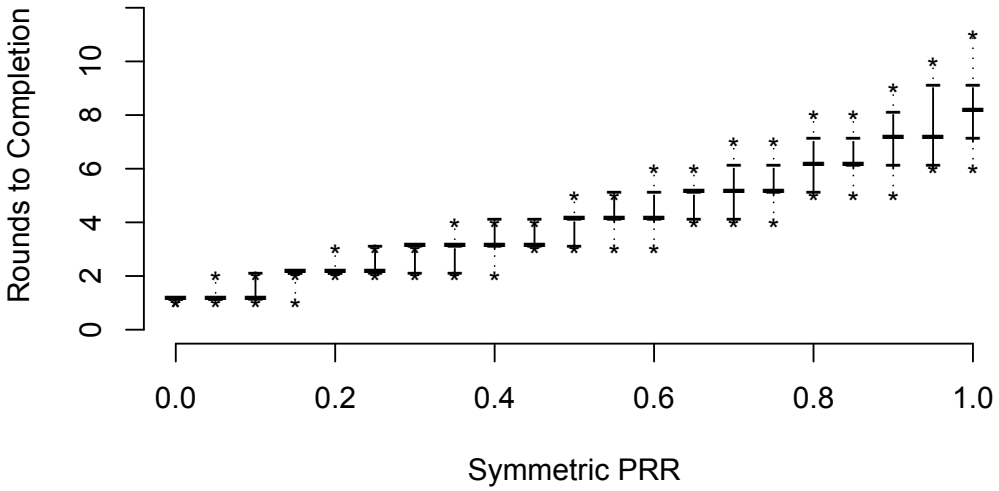
a sender to acknowledge an NC probe and continue, it must choose the NC PA correctly *and* receive the NC probe. This effectively changes p_{sel} to $p_{\text{sel}}p_{\text{rs}}$, where p_{rs} is the PRR of the receiver-to-sender link. If all senders experience identical, but independent, loss rates, then this is a simple substitution in the preceding analysis. Incorporating variable loss rates between senders, however, greatly complicates the analysis without providing much new insight, so we examine this through simulation.

The impact of uncorrelated acknowledgement loss is also fairly straightforward: if *any* acknowledgements reach the receiver, *there is no impact at all*. A node which sends a lost acknowledgement proceeds to the next negotiation round, and since the receiver gets *some* acknowledgements, it proceeds as well.

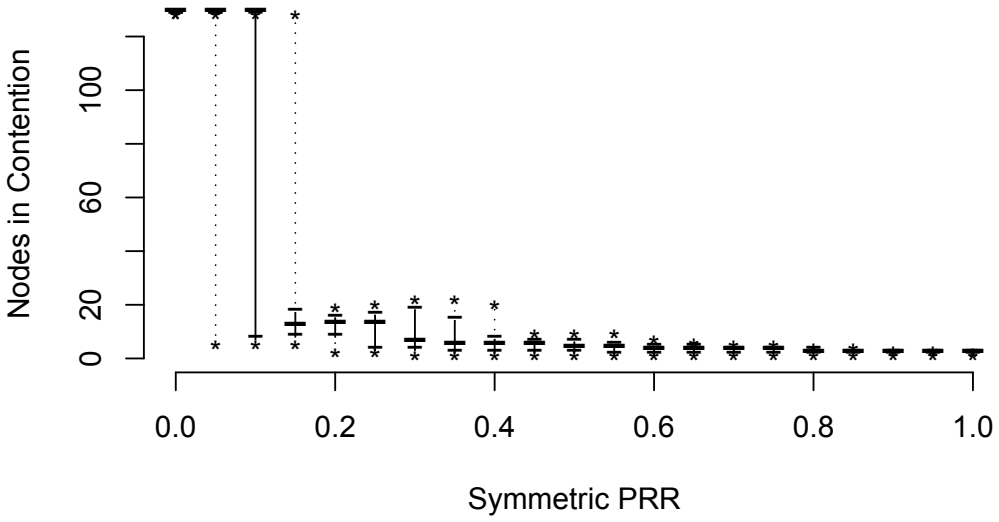
To investigate how uncorrelated losses affect Flip-MAC, we first simulated its operation on 128 senders, each with the same PRR (both for probes and acknowledgements). Figure 3.8 shows that Flip-MAC is more resilient to uncorrelated loss than it is to correlated loss: negotiations tend to last longer, and as long as PRR is not uniformly terrible, we ultimately end up with very little contention. This makes sense, as a single correlated loss event can stop the entire process, but the probability of many uncorrelated losses occurring simultaneously is much lower.

In practice, it's not likely that each sender will experience the same uncorrelated PRR. In Figure 3.9, we run the same simulation just described, but we draw each sender's PRR from a Gaussian distribution with mean 0.5 and a variable variance[6]. When variance is low, the process completes quickly and leaves a large number of nodes in contention. This makes sense: in the extreme case, where variance is 0, every node has an effective p_{sel} of 0.25. As we increase the variance, we generate more nodes with "good" PRR: these are able to remain in negotiation longer. When there is a wide enough margin between "good" and "bad" nodes,

[6]Sample PRR's drawn with value less than 0 or greater than 1.0 were set to 0 and 1.0, respectively.



(a) When link quality is poor, negotiation time is low: the receiver stops the process at the first round where no acks are received.



(b) "Early stopping" leaves many nodes in contention, as few eliminations occur.

Figure 3.8: Impact of symmetric packet loss on duration and final contention. All senders experience the same bi-directional PRR.

the "good" nodes can keep negotiation running long enough for the "bad nodes" to all drop out. The next section digs deeper into this effect.

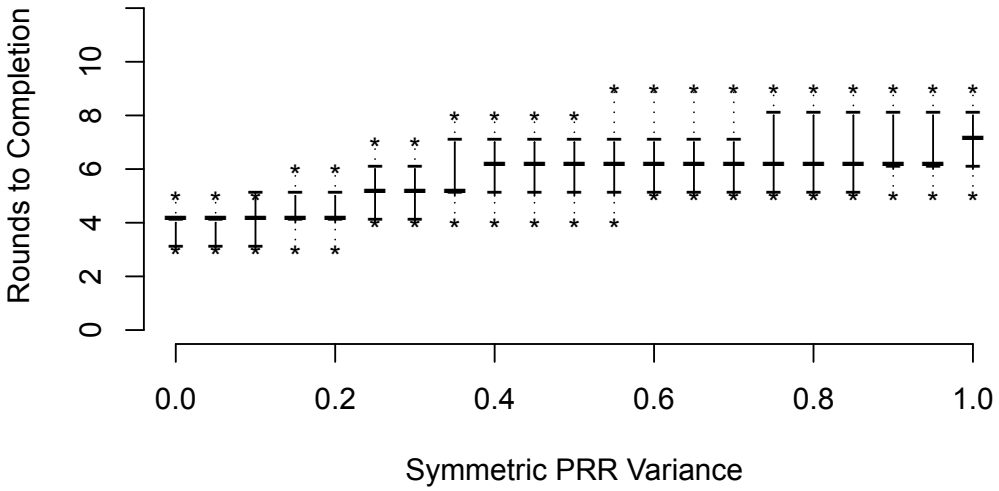
Sender Selection Bias

We suggested above that variable PRRs lead to different outcomes for senders with good links from senders with bad links. Our evaluation on the testbed showed a slight preference for nodes with higher RSSIs, but without instantaneous ground-truth PRR measurements, the connection to actual link quality is a little tenuous. However, we can simulate this effect with higher numbers of nodes and finely controlled PRRs.

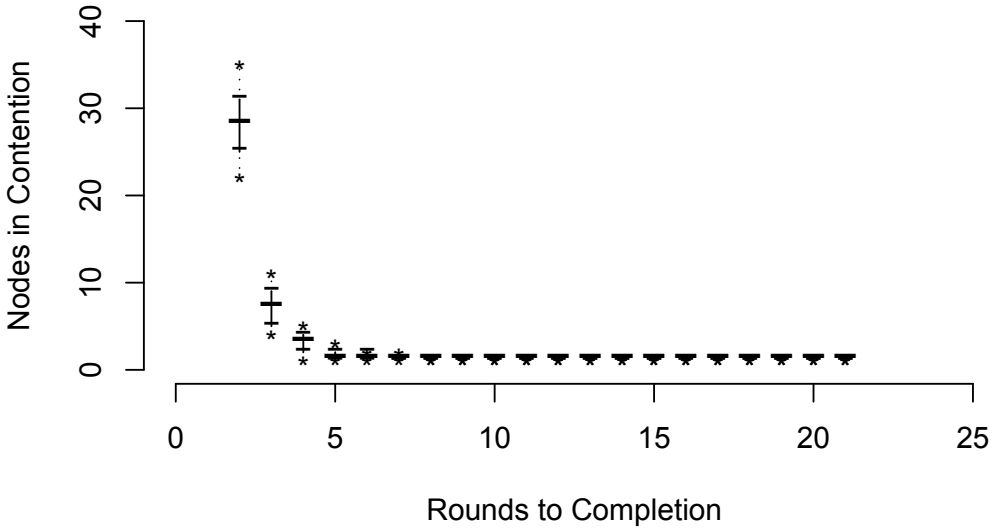
In Figure 3.10, we can observe a positive selection bias for nodes with good links. We use the same dataset used in Figure 3.9, but we look at the rank of the selected sender when the nodes are ordered by PRR. A value of 128 indicates that the selected node had the best PRR, and 1 indicates it had the worst.

The effect is striking: as long as there is a modest degree of variability in links, Flip-MAC is strongly biased in favor of selecting good links. The accumulated effect of several loss events is sufficient to weed out the worst links. Keep in mind that these effects manifest over the span of less than 12 negotiation rounds for the most part (Figure 3.9): if sender density is higher, there will be more time for this effect to work. Since this effect will be stronger as negotiation length increases, it's not terribly surprising that the (relatively-short) negotiations observed on our testbed didn't turn up a dramatic bias.

The effect of a single lost acknowledgement is very different from the effect of a single lost probe from the sender's perspective (the former may have no effect at all, while the latter instantly disqualifies them). This naturally raises the concern that when links are not symmetric, Flip-MAC will be biased in favor of good receiver-to-sender links, but not necessarily good sender-to-receiver links. Figure 3.11 explores this. For this simulation, we



(a) Nodes with high PRR tend to last longer in negotiations than those with low PRR. Increasing the variance gives us more of these.



(b) If there are enough "good" nodes to keep negotiations running for a few rounds, the final contention remains low. Short negotiations, caused by poor links, lead to high contention.

Figure 3.9: Probe and ack PRR are equal for each sender, drawn from a Gaussian distribution with mean 0.5.

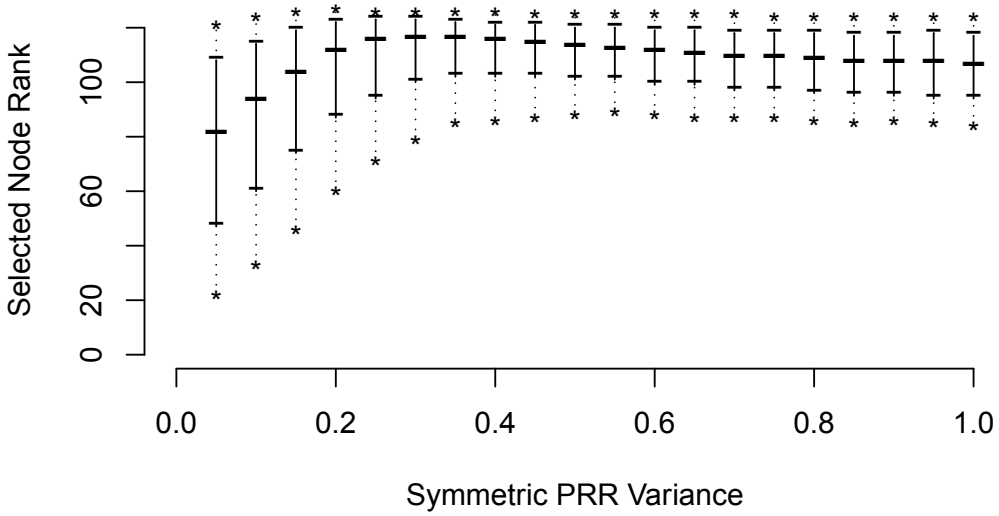


Figure 3.10: Simulation of 128 senders with mean symmetric PRR of 0.5. A modest degree of variance in PRRs is sufficient to consistently select one of the best nodes in this case. The Y-value indicates the rank of the selected sender's probe PRR (128 is the best, 1 is the worst).

selected probe PRR and ack PRR independently from a Gaussian distribution with mean 0.5 and variable variance. This shows that while the process is somewhat biased towards links with better receiver-to-sender connectivity than sender-to-receiver connectivity, the median asymmetry is well below the variance of the underlying distribution. The simulation picks the "winner" at random from the pool of nodes which remained in contention at the end of negotiation, so in practice this effect will likely be mitigated by the fact the receiver is more likely to get a data packet from a node with a good sender-to-receiver link than from a node with a poor one.

3.5 Future Work

There are several promising avenues for expansion in Flip-MAC.

In real deployment scenarios, we will likely need to deal with concurrent negotiations. In order to reduce conflict between these, it would be good to include a channel-switching

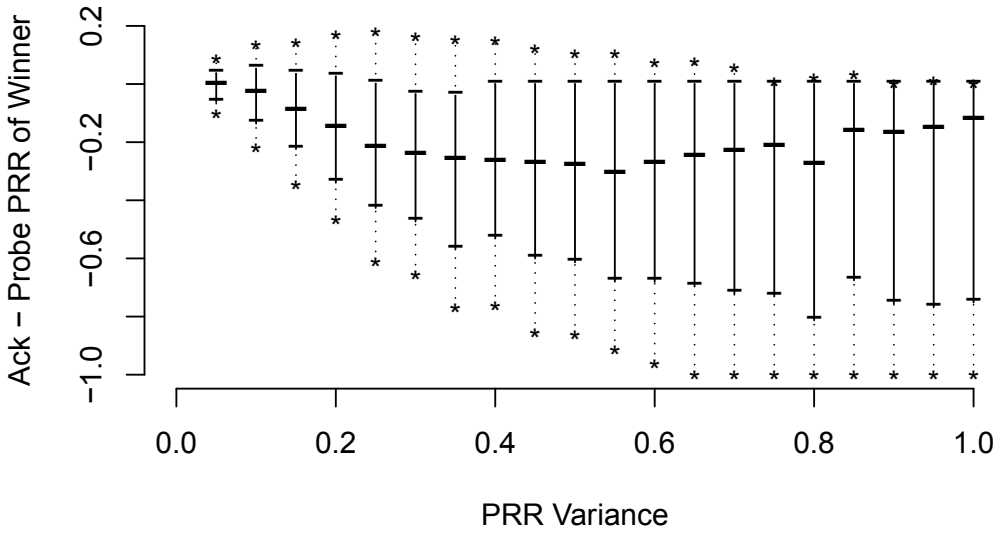


Figure 3.11: Simulation of 128 senders with independently chosen probe and ack PRRs (mean 0.5). The Y-value indicates Ack PRR - probe PRR: a value of 0 is a symmetric link, a negative value indicates that the probes have a higher PRR than the acks.

mechanism. In this manner, the DP probes would occupy a single control channel, while the negotiation could take place on a separate node-specific channel. This takes the scheme proposed in A-MAC one step further. By moving both data transfer and the lion's share of control traffic away from the most-frequently used channel, we aim to isolate receivers from each other.

Currently, we use a fixed p_{sel} during the negotiation. However, if the initial level of contention is known ahead of time to be very high, it could be advantageous to start with a lower p_{sel} (say, 0.25 instead of 0.5 on the first round) in order to reduce contention more quickly. While this carries a risk of ending with higher final contention if it is carried through the entire negotiation, it could be a good tradeoff in the early rounds. We don't currently make use of the RSSI measurements which the CC2420 provides us for received acknowledgements, but these could potentially be used to estimate the level of contention (more senders leads to more acks, which will have higher RSSI).

Perhaps the most exciting idea which Flip-MAC inspires is the concept of structuring MAC protocols into multiple phases. It would be an interesting engineering problem to design a new and more modular radio stack which promotes this viewpoint. One could imagine adaptively modifying the DP probe frequency in response to usage, adjusting initial back-offs in the low-contention MAC protocols, and other interactions between modules.

3.6 Conclusion

In this work, we have demonstrated the promise of Flip-MAC through analysis, simulation, and testbed experiments. Our testbed results suggest that Flip-MAC can work in practice, while simulation and analysis lead us to expect good performance as network density increases. The simulation indicates that not only is Flip-MAC robust to the introduction of poor links in the mix of initial senders, it compensates for this by favoring good links.

In the future, we see an exciting line of new research in combining well-structured MAC protocol modules to suit the needs of different deployment scenarios.

Chapter 4

CX: Forwarder Selection in Multi-Transmitter Networks

4.1 Multi-transmitter networks

In Chapter 2, we described some of the limitations that single-path routing imposes on low power networks. Single-path routes can be difficult to identify. They require collecting accurate link quality information. They are fragile to single node or link failures.

The previous chapter showed that on the TelosB mote platform, hardware-generated acknowledgement packets collide non-destructively: they have identical content and are sent at almost precisely the same time. However, if acknowledegements are the only type of packet that can be sent in such a manner, there is only so much that we can accomplish.

At the time of Flip-MAC's publication, Glossy [22] demonstrated that with great care, one could achieve non-destructive concurrent *data* transmissions on the TelosB platform. This capability is used to implement multi-transmitter floods, in which the flood source sends a packet, each of its neighbors rebroadcasts it simultaneously, and so on. Glossy showed that the timing precision necessary to achieve this is related to the radio's chip rate (2 MChip/s with the CC2420 transceiver). This observation was quite serendipitous for us: we were developing the Bacon wireless mote (described in detail in Chapter 6), which has a flexible transceiver capable of operating at a wide range of chip rates. Along with timer access to a

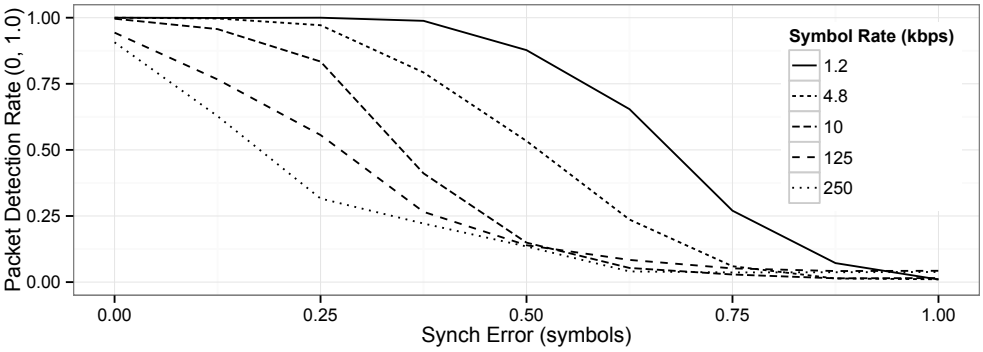
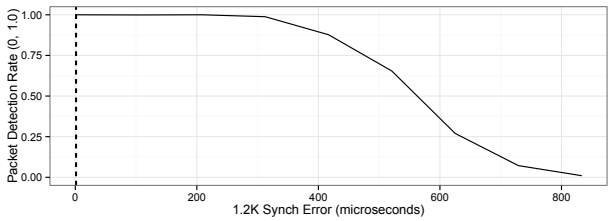


Figure 4.1: Packet detection rate as a function of symbol rate and transmitter offset in the absence of capture effect. Higher symbol rates were more susceptible to reception errors at small transmitter offsets. Note that x-axis is in terms of symbol length: the same horizontal distance represents a different length of time for each series.

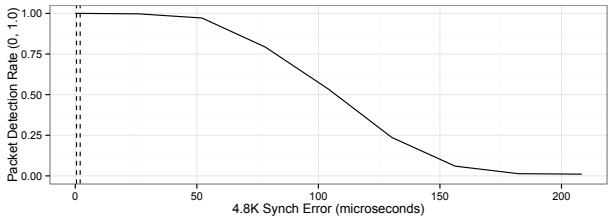
high-precision radio oscillator, this enables us to achieve non-destructive interference with much more timing flexibility. Could we leverage this to do better than simple flooding?

In order to establish whether this approach was practical on our platform, we experimentally measured the packet detection rate (PDR) resulting from transmissions between two senders. PDR measures the fraction of transmitted packets that resulted in a valid preamble detection, and is an upper bound on the packet reception rate. We connected two senders to a single receiver with coaxial cables and variable attenuators. We carefully tuned the attenuation of the senders to equalize their power at the receiver, and then varied the offset between transmissions (to simulate timing errors) for a range of symbol rates [1]. Figures 4.1 and 4.2 show the resulting packet detection rates. These results indicated that on our platform, a 125 kbps symbol rate struck a good balance between transmission speed and error rate. At a 1μs error (which we used as a rough estimate of the worst synchronization errors we would encounter), the 250 kbps symbol rate yielded a 32% PDR while the 125 kbps symbol rate yielded 77%. We decided that it was not worth lowering the speed by more than a factor of

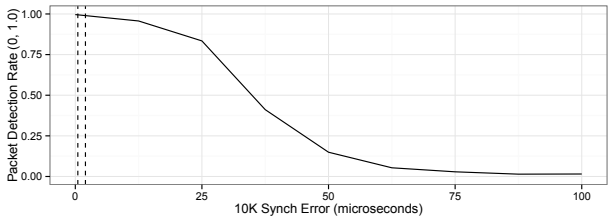
[1]We use the terms "symbol rate" and "chip rate" interchangeably in this text, as the CC430 transceiver we are using employs no direct sequence spread spectrum (DSSS), so a bit, symbol, and chip all represent the same thing.



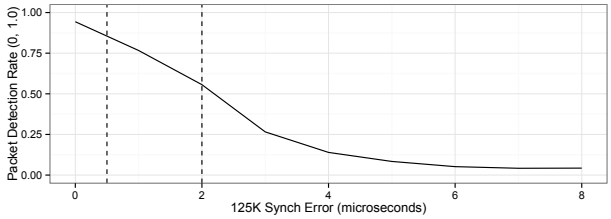
(a)



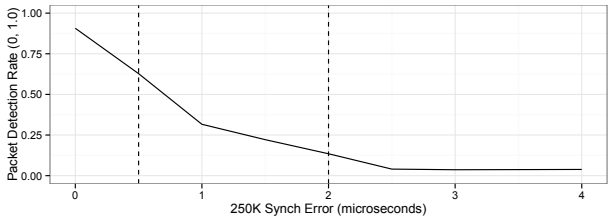
(b)



(c)



(d)



(e)

Figure 4.2: Packet detection rate as a function of symbol rate and transmitter offset in the absence of capture effect. Vertical lines are placed at 0.5 and 2 microseconds for scale.

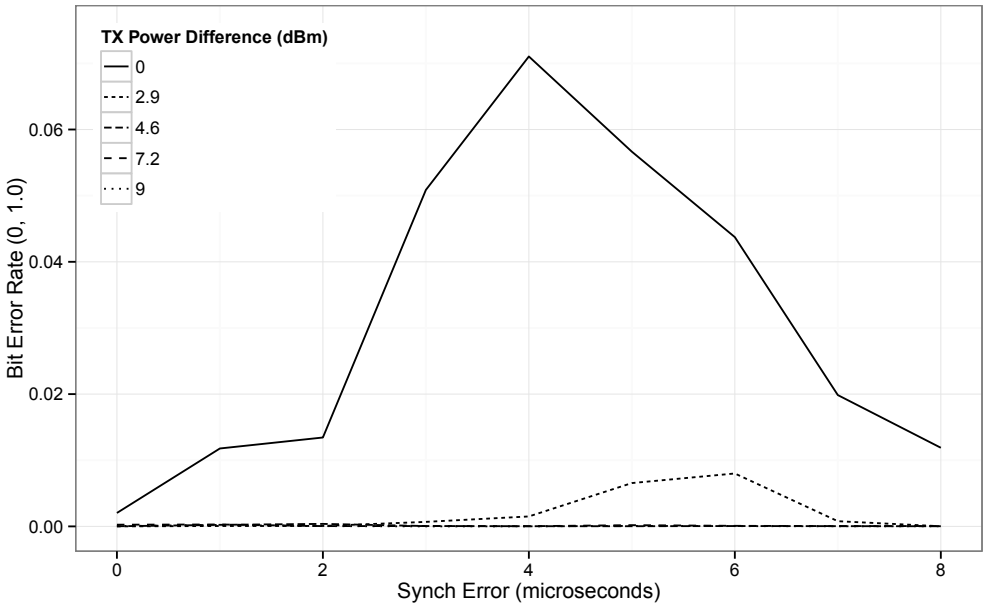


Figure 4.3: Bit Error Rate as a function of transmitter offset for a range of transmitter power differences at 125 kbps symbol rate. For well-synchronized transmissions, a 3 dBm difference in received signal strength is sufficient to achieve nearly loss-free operation.

10 to improve the best-case PDR (at perfect synchronization) from 94% to 99%.

Previous work in *unsynchronized* network flooding [41] showed that differences in received signal strength across incoming transmissions can improve reception rates. Essentially, the receiver hardware tracks and demodulates the frequency shifts in the stronger of two simultaneous signals and filters out the weaker. We wished to observe whether this phenomenon, called "capture effect," could further improve the reliability of concurrent transmissions at the time scales which we are interested in. Figure 4.3 shows how varying the relative transmission power and offset between two senders affects bit error rate (PDR was greater than 99.5% at all measured points). With this transceiver, even a modest 3 dBm difference in signal strength can greatly improve reception quality in the face of large timing errors.

We implemented the Hamming(7,4) [3] forward error correction scheme to handle the

relatively low bit error rates observed. While this reduces the effective data rate, it greatly enhances the reliability of multi-transmitter floods.

In this chapter, we design and evaluate a novel multi-transmitter approach to networking. Our goal is to take advantage of the flexible timing afforded by our platform to make on-the-fly decisions about which nodes are useful in moving data from the source to the destination and which ones are not. By limiting multi-transmitter floods to a subset of the network, we aim to save energy by turning nodes off when they are not in use and improve throughput by setting inter-packet spacing based on the source-to-destination distance. We also show that these forwarder sets are more resilient to random node failures than single-path routes.

The bulk of this chapter originally appeared at the 2013 IEEE International Conference on Distributed Computing in Sensing Systems (DCOSS 2013).

4.2 Introduction

Glossy [22], Low-power Wireless Bus (LWB) [21], Flash Flooding [41], and Insteon [54] represent examples in an emerging family of multihop wireless sensor network (WSN) communication protocols. These systems leverage a combination of non-destructive concurrent transmissions and radio capture effect [33,69] to perform fast network floods that reach all nodes with high probability: every node receiving a packet rebroadcasts it at precisely the same time, reducing destructive interference. This approach, which we refer to as *multi-transmitter* networking, carries several key benefits. In addition to the high yield, good throughput, and low energy consumption demonstrated in LWB, these methods require little routing state to work. This makes them suitable for networks with high degrees of node mobility, and may help in challenging environments where existing routing methods struggle to find high quality links.

Despite these benefits, it is obvious that not every node needs to be involved in every non-flood data transfer (collection, point to point communication, etc). In collection, a node that

is adjacent to the data sink should not need assistance from the whole network to deliver its packets, and a node which always receives packets *after* the sink cannot help other nodes, to cite two simple examples. If we can reduce the set of nodes involved in a transfer, then non-forwarders can turn their radios off to save energy. Furthermore, if the diameter of the forwarder set is smaller than that of the network, this information can be used to set inter-packet spacing and thereby increase throughput without introducing collisions. The goal of this work is to solve the problem of forwarder selection in this context and demonstrate its benefits. By introducing a better network primitive than flooding, we advance the state of the art in multi-transmitter systems.

To this end, this work makes four contributions: (1) We formally define forwarder selection in multi-transmitter networks. (2) We propose a forwarder-selection protocol, the first of its kind of which we are aware. (3) We describe the first TinyOS implementation of a network stack based on precisely-timed non-destructive concurrent transmissions, which we call CX (Concurrent Transmissions), and (4) we evaluate how our forwarder selection mechanism, CXFS, improves performance over simple flooding in this system.

Under CX, all communication takes place in the form of multi-transmitter floods. Through the use of a hop counter in each packet, nodes learn their relative distances to each other. The hop count information allows nodes to estimate whether they are between a source and destination for a given transfer. We address the inherent unreliability of hop-count and use this as the basis for our forwarder-selection method, which keeps nodes which are between the source and destination active while allowing the rest of the network to sleep.

Results from our 66-node indoor testbed show that forwarder-selection reduces duty cycle by 30% on average over simple concurrent flooding while maintaining an average packet reception ratio (PRR) of 99.4%. In the same setting, nodes increase their throughput by 49% on average. CXFS forwarder sets are also shown to deliver packets more reliably than single-path routes when subjected to random node failures and stale link quality measurements.

This chapter has six more sections. We present related work in Section 4.3 and define

the task of forwarder selection in Section 4.4. Section 4.5 describes the CX forwarder selection protocol, and section 4.6 describes the CX network stack and some key implementation details. Section 4.7 presents results from our testbed. We conclude in Section 4.8 with a summary.

4.3 Related Work

Concurrent transmissions have been studied extensively in the context of wireless sensor networks [41, 55]. However, the bulk of this work relates to the capture effect, an artifact of wireless receiver design that allows radios to lock onto and successfully decode packets in the presence of considerable interference from other packets as long as the signal-to-noise ratio is high enough. While capture effect may benefit CX, we do not rely on it for successful packet receptions.

The research community has focused less on concurrent transmissions in the form of radio signals interfering non-destructively, mainly because the timing required to perform non-destructive interference is considered difficult to achieve on resource-constrained devices. Note that non-destructive concurrent transmissions are different from beamforming and Multiple-Input Multiple-Output (MIMO) where antenna arrays at the transmitter and/or receiver are used to generate signals that interfere constructively at some desired angles.

Early work using the principle of non-destructive interference has primarily relied on the "free" transmission synchronization provided by hardware-generated acknowledgments from IEEE 802.15.4-compliant radios. Dutta et al. [19] first used these acknowledgments to efficiently wake up a large network of nodes. The same principle has been applied to medium contention and arbitration in receiver-initiated MAC protocols such as A-MAC [17] and Flip-MAC [9]. Strawman [46] demonstrates that even destructive radio interference can be a valuable primitive in low-power networks, and works without requiring precise transmission timing.

By taking full control of the MCU on the TelosB during packet reception, Ferrari et al.

succeeded in generating non-destructive concurrent transmissions using a software-based approach, and used this to build a highly efficient network flooding protocol (Glossy) with microsecond time synchronization [22] on the Contiki embedded operating system [15]. While implementation details are not widely available, Insteon [54] is a commercial product which appears to use a network protocol which is very similar to Glossy. Our CX stack enables Glossy-type flooding in TinyOS [37], though it achieves this by using our hardware platform's high-precision timer/capture module rather than deterministic execution times.

Ferrari et al. later added a scheduler on top of the Glossy flooding protocol to construct the Low-power Wireless Bus (LWB) [21], a virtual one-hop network. In LWB, a master node in the network assembles and disseminates a TDMA schedule based on bandwidth requests by the nodes in the network. Our work is focused on the performance of the network protocol (not the scheduler), though LWB demonstrates that multi-transmitter networking protocols are practical and effective building blocks for real systems.

More recently, Wang et al. studied the scalability of Glossy-like flooding protocols and found that time synchronization errors (for scheduling concurrent transmissions) could accumulate over each hop and lead to destructive interference as the network scales [68]. In response, they proposed a flooding protocol which limits the number of forwarders to reduce the timing errors. Nevertheless, this study was purely based on analytical models of baseband wireless communications, and did not consider the effects of carrier wave frequency/phase offset or model the radio hardware behaviors such as symbol clock and phase recovery. The proposed flooding protocol further assumes that the geographical locations of all nodes are known, and uses the simple unit-disk communication model. Under these assumptions, the simulation results indicate that this protocol can be more scalable than Glossy.

4.4 Forwarder Selection

Our overall goal in this work is to reduce the nodes involved in forwarding data to those that do useful work, while still preserving good connectivity. In this section, we introduce the

forwarder set concept and describe multi-transmitter flood, standard single-path, and our *Reduced Routeless* forwarder sets in formal terms. Section 4.5 describes how we approximate the reduced-routeless forwarder set in real-world settings.

4.4.1 Forwarder sets: Current Approaches

For the purposes of this section, we treat a wireless network as a directed graph G with vertices for each communication node and edges for each wireless link (having an associated packet reception ratio). We further assume that loss events are independent (e.g., no external interference). Under the concurrent communications paradigm [22], data transmissions take place in discrete communication rounds: transmissions of the same data in the same communication round are non-interfering. If a set of nodes S sends the same data packet at the same time, any node in the union of the adjacency sets of every node $s \in S$ successfully receives the packet with a probability determined by the union of packet reception ratios on the adjacent edges. In reality, synchronization errors and other effects may violate this model, though it remains useful for the purpose of explanation.

In this framework, a "valid" forwarder set from source s to destination d is any set of nodes F for which the subgraph of G consisting of nodes in F connects s to d with high probability (i.e. good end-to-end PRR).

Single-Path Routing

Single-path routing protocols effectively approximate some minimal forwarder set, $F_{min}(s, d)$, for each (source, destination) pair which consists only of the nodes on a single minimum cost path between source s and destination d. Path cost in this context is defined as the sum of link costs using metrics such as ETX [12]. In Figure 4.4, $\{e\}$, $\{c\}$, and $\{g\}$ would all fit the definition of F_{min}.

Even under perfect network conditions, single-path routing methods require coordination

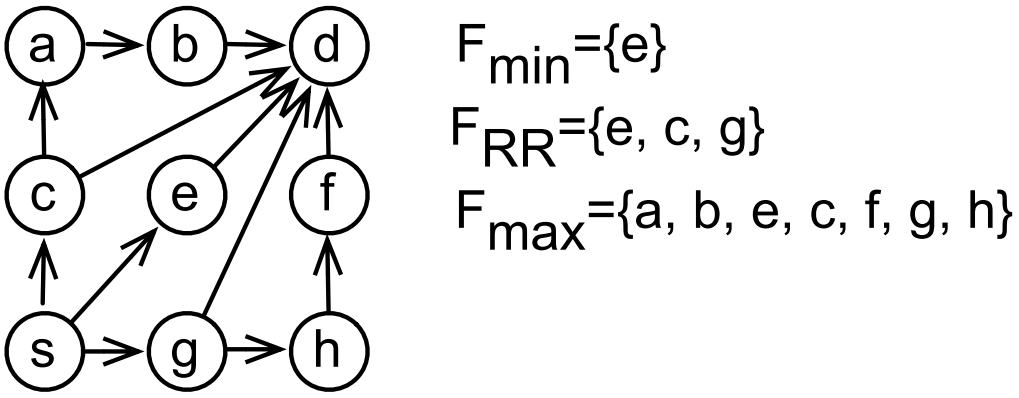


Figure 4.4: Examples of minimum (F_{min}), maximum (F_{max}), and Reduced-Routeless (F_{RR}) forwarder sets for a 3×3 grid. All edges have cost 1.

between nodes to estimate link metrics and calculate end-to-end paths. Nevertheless, when properly executed, such protocols have been shown to be very efficient, both in terms of energy and channel utilization [24].

On the other hand, single-path routing protocols can have difficulty adapting to variable link conditions, requiring either agile link quality estimation, reliance on conservative links, or a combination of the two. Such routing protocols can also experience periods of packet loss when previously-reliable routes break (e.g., due to node failures).

Multi-Transmitter Flooding

Multi-transmitter flooding selects the maximum forwarder set F_{max}: every node in the network. The validity condition is trivially satisfied for this set (as long as some reliable end-to-end path exists between s and d anywhere in the original network). Figure 4.4 illustrates the difference between F_{min} and F_{max}.

In multi-transmitter floods, many more nodes are used for forwarding than is strictly necessary. Under idealized network conditions (e.g. no external interference), any node in F_{max} which does not lie on a minimum cost path between s and d is not useful: if they were removed from F_{max}, d would still receive a packet in the same number of communication

rounds and the inclusion of these extra nodes is a source of unnecessary energy expenditure. For the example in Figure 4.4, $\{c, e, g\}$ all lie on a minimum cost path while $\{a, b, f, h\}$ are the extra nodes.

Furthermore, in order to prevent interference, a source node cannot start a new flood until the previous one has completed. When the entire network participates in flooding, this means that a node has to wait until a number of communication rounds (at least) equal to the diameter of the network have elapsed before it transmits without fear of collision.

In contrast to single-path routing, protocols that use multi-transmitter flooding have been shown to work well even in the face of link asymmetry and dynamism. Since this approach does not need to adapt routing state to link conditions, and since no bidirectional communication is required, as long as there is some reliable path from s to d at the point in time when the flood begins, d will successfully receive the packet.

4.4.2 Reduced-Routeless Forwarder Sets

Between these extremes, we define a ***Reduced-Routeless*** forwarder set $F_{RR}(s, d)$, which consists of all nodes f on *any* minimum cost path between s and d.

By the triangle inequality, a node f is a member of $F_{RR}(s, d)$ if and only if

$$d_{sf} + d_{fd} = d_{sd} \tag{4.1}$$

where d_{ij} denotes the minimum distance, as defined by a cumulative metric (e.g., ETX, hop count), between i and j.

In contrast to single-path routing, each node f can test whether $f \in F_{RR}(s, d)$ without exchanging routing information with non-endpoint nodes: it only needs to have the end-to-end cost and its cost relative to each end. In other words, F_{RR} is similar to F_{max} in protocol design (thus the word "routeless") but is closer to F_{min} in its goals.

Although $F_{RR}(s, d)$ still includes more nodes than $F_{min}(s, d)$ for any network with more than one minimum cost path between s and d, it is also smaller than F_{max} as long as not all

paths between s and d are minimum cost paths. Using F_{RR} instead of F_{max} should therefore reduce duty cycle (by decreasing the number of participating nodes) and increase throughput (by decreasing the diameter of the forwarder set and reducing inter-packet spacing) while retaining much of the path redundancy afforded by flooding.

However, if propagation patterns are not relatively stable (e.g. due to interference between concurrent senders or varying link conditions), then a node may remove itself from the forwarder set when it would be useful, or add itself to a forwarder set when it may be redundant. In the next section, we discuss several practical approaches to handling such variability.

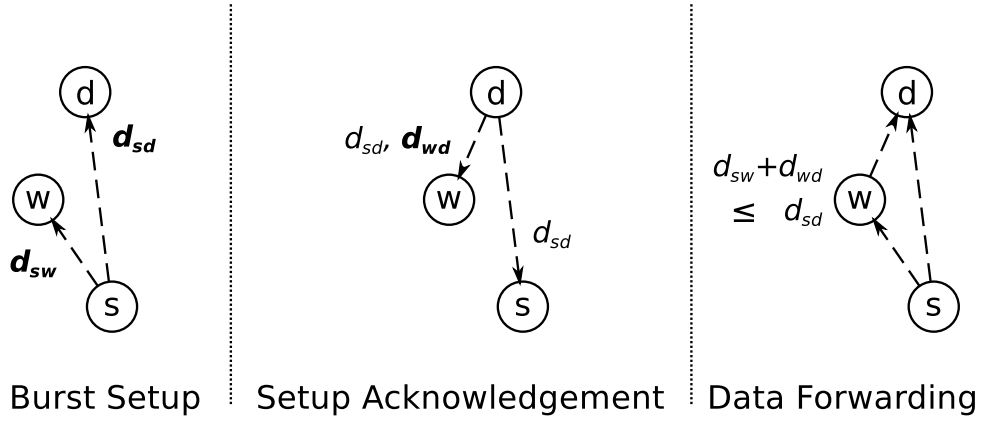
4.5 CX Forwarder Selection Protocol

The CX (Concurrent Transmissions) multi-transmitter network stack allows nodes to approximate their membership in F_{RR}, using these results to duty cycle their radios and set inter-packet spacing. Next, we provide details on the protocol used to select forwarders, CXFS (CX Forwarder Selection). Each exchange in the setup process uses a basic multi-transmitter flood, whose implementation is described in Section 4.6 and which is evaluated separately in Section 4.7.2.

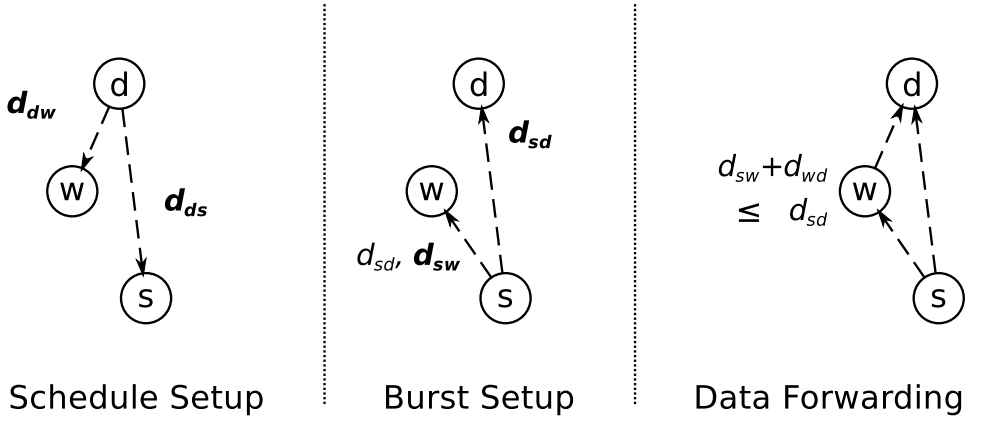
4.5.1 CXFS Operation

In order for a node w to decide whether it should forward or not, it must learn its distance from the source, its distance from the destination, and the distance from source to destination (d_{sw}, d_{wd}, and d_{sd}, respectively).

For point-to-point transmissions, Figure 4.5(a) shows the steps in CXFS. The source s sends a Burst Setup packet, which allows each potential forwarder w to measure d_{sw} and the destination d to measure d_{sd}. In the Setup Acknowledgment phase, d responds with a packet containing d_{sd} in its body, which allows w to obtain d_{sd} and to measure d_{wd} (under the



(a) CX forwarder selection for point-to-point traffic. In Burst Setup, s floods a message to inform potential forwarders w and the destination d of their distance from the source (d_{sw} and d_{sd}, respectively). In Setup Acknowledgment, d floods a message which informs the network of d_{sd} and d_{wd} (assumed from d_{dw}). In Data Forwarding, w forwards messages only if $d_{sw} + d_{wd} \leq d_{sd}$.



(b) CX forwarder selection for collection traffic. The Setup Acknowledgment phase of the general point-to-point process is replaced by the periodic schedule announcements from the data sink, d.

Figure 4.5: CX forwarder selection process. Dotted lines indicate multi-hop paths, boldface distances are directly observed, non-boldface distances are reported in packet bodies.

simplifying assumption that $d_{wd} = d_{dw}$). w now has enough information to make forwarding decisions.

In the case of data collection in CX, the network root both sends out periodic TDMA schedule packets and is the destination for all data traffic. Figure 4.5(b) shows how this can be leveraged. The schedule packets provide d_{dw} to each potential forwarder and d_{ds} to each potential source. The Burst Setup packet now contains d_{sd} and allows nodes to measure d_{sw} (again, under the assumption of symmetric distances). w can now make forwarding decisions for (s, d) packets while only requiring a single end-to-end communication in s's time slot.

This decreases the setup overhead at the cost of increasing the staleness of distance information. Our evaluation did not indicate problems resulting from this. In the rest of this work, we will restrict the rest of our discussion to the collection case.

In either setting, node w can compute its membership in $F_{RR}(s, d)$. In the Packet Forwarding phase, nodes not in $F_{RR}(s, d)$ turn off their radios, and s sends packets with inter-packet spacing determined by d_{sd}.

4.5.2 Hop-count as Multi-transmitter distance metric

The previous discussion has used the concept of "distance" in multi-transmitter networks without defining it. In this subsection, we describe the motivation behind our use of hop-count as the basis for our distance metric and how we make the most of it.

"You can't seriously be using hop count, can you?"

While hop-count is a notoriously unreliable metric in the single-transmitter networking domain, it possesses several key characteristics which make it the best choice for CXFS.

First, hop-count *obeys the triangle inequality*: a node which is 4 hops from the source is on no shortest path to a node which is 3 hops from the source. Any distance metric which cannot be applied to the definition of F_{RR} in equation 4.1 will not work in CXFS.

Second, in order for concurrent transmissions to be reliable, *simultaneously transmitted content must be identical*. In a sense, intermediate nodes *cannot* embed information in packets that is unique to them, and forwarding decisions must be made based on information that is common to all predecessors of a forwarder. Nodes cannot tell, for instance, which sender(s) participated in transmitting a packet that they received. Hop-count can convey meaningful distance information without requiring intermediate nodes to send conflicting data.

Finally, the distance metric *must make sense in the multi-transmitter networking context*. Putting aside the first two characteristics, traditional physical-layer metrics (such as RSSI and LQI) are subject to a range of effects in the multi-transmitter domain which may limit their usefulness. For instance, phase differences between two transmitters lead to wide variability in RSSI measurements at receivers, depending on whether the transmissions interfere constructively or destructively. We view the behavior and usefulness of physical layer metrics in the multi-transmitter setting as a topic for future research.

Rather than spurn hop-count for its faults, we choose to embrace it for its virtues.

Hop-count variability in multi-transmitter flooding

Previous work in single-path routing protocols has repeatedly shown that link quality can vary rapidly over time [56]. Furthermore, links can be asymmetric [5]. Thus, we can expect that the distance measurements in the multi-transmitter environment are similarly ephemeral and have to be updated over time, in order to provide accurate distance estimation.

Indeed, the results obtained using multi-transmitter floods on our testbed (described in Section 4.7) may show considerable long-term and short-term fluctuations in distance measurements for some nodes, as Figure 4.6 illustrates. Since F_{RR} is based on these hop-count measurements, the more accurately that we estimate distance and the better we handle its variability, the better our estimate of F_{RR} will be.

Estimating distance is analogous to link estimation and route discovery in single-path routing protocols, and faces the same challenges with regard to efficient neighborhood and

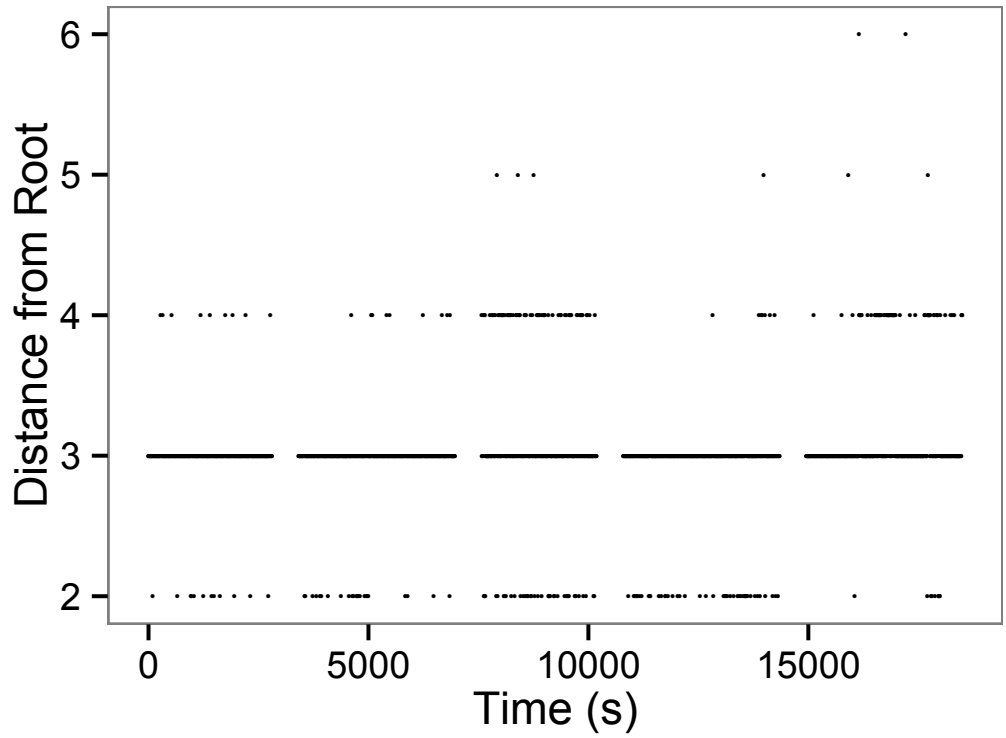


Figure 4.6: Hop-count distance measurements at node d when node s sends concurrent floods back-to-back. Tests were spaced several hours apart, but concatenated to show long-term variation. Gaps separate each test run above.

routing table maintenance. We seek a distance estimation strategy that is lightweight, yet accurate enough to provide a reliable and compact forwarder set. We propose three strategies to do this: we either use the last-observed hop-count (termed "Last"), the average of observed hop-counts, or the maximum of observed hop-counts.

"Last" is easy to implement and requires basically no routing state, but may behave poorly as distances vary. "Average" may prevent nodes from responding to rapid changes, but will handle some degree of variability. "Max" will give the most inclusive forwarder sets, making it more reliable but also less power efficient. Additionally, by conservatively estimating distances between nodes, this strategy has the potential to increase inter-packet spacing by overreacting to rare bad transmissions. While "Last" is usable on networks of all sizes and for all traffic patterns, under the collection traffic pattern it is certainly feasible to maintain an average or maximum hop-count relative to the root, and in many real-world networks it is feasible to maintain this for each node in the network.

Distance Fluctuation Boundary

Orthogonal to the different distance estimation strategies, we add a boundary zone to the shortest-path distance in Equation (4.1) to account for estimation errors, short-term variability, and the assumption of distance symmetry.

By adding the boundary zone width b and using $\widehat{d}$ to denote the estimated distance and assumption of distance symmetry, $F_{RR}(s,d)$'s definition from Equation (4.1) becomes:

$$F_{RR}(s,d) = \{f : \widehat{d_{sf}} + \widehat{d_{fd}} \leq \widehat{d_{sd}} + b\} \tag{4.2}$$

By increasing b, we include more nodes in the forwarder set $F_{RR}(s,d)$, up to and including using the entire network. Increasing the size of the boundary zone aids end-to-end packet reception by both increasing the diversity of paths that a packet can travel (mitigating the possibility of selecting a forwarder set that lacks reliable links) and adding an error margin to the distance estimations.

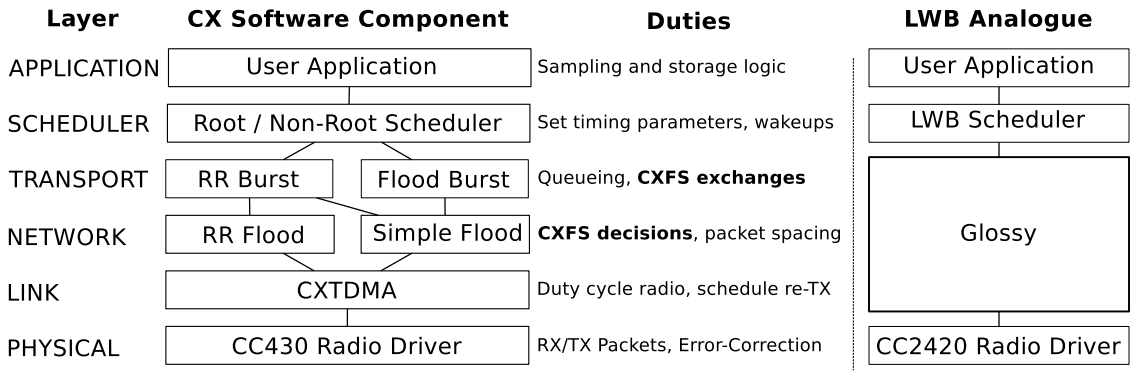


Figure 4.7: CX network stack design, with rough equivalents in LWB [21]. This work focuses on the bolded duties.

As the evaluation in Section 4.7 shows, even with these simple approximations and strategies CX offers significant improvements to the state of the art.

4.6 CX Software Design and Implementation

4.6.1 Software Design

We implemented our CX network stack in TinyOS 2.1 with CXFS based on the distance metrics, estimation strategies, and boundary zone described above. Figure 4.7 shows a high-level view of the software architecture.

The CX link layer maintains transmission scheduling information. Time is divided into fixed-length frames (on the order of a packet-length in duration), and these are grouped into fixed-size slots (on the order of 10's of frames). Each node in the network is assigned a slot, and this schedule is distributed by a single root node. Nodes turn their radios on slightly ahead of each *frame* start, and keep their radios on until either a packet is received or a fixed 1 ms timeout elapses.

When a node rebroadcasts a packet, it does so at the next frame start after incrementing its hop-count field and decrementing its time-to-live (TTL) counter. Packets with TTL of 0 are dropped.

The CX network layer provides two primitives, Simple Flood and RR Flood. Under a Simple Flood, packets have their TTL initially set to the network diameter, and a node always rebroadcasts a packet with a non-zero TTL. In an RR flood, packets have their TTL initially set to $\widehat{d_{sd}} + b$. The RR Flood component is responsible for the forwarding decisions in CXFS: nodes only rebroadcast packets for which the inequality in Equation (4.2) holds.

CX provides two transport layer protocols on top of these, Flood Burst and RR Burst. These control packet queuing and spacing (starting the next transmission after the current packet's TTL has expired). Nodes queue packets until some threshold is exceeded, and then they transfer as many as they can in their next slot. Broadcast packets are handled by the Flood Burst transport protocol, while unicast packets are handled by the RR Burst protocol.

RR Burst implements the CXFS protocol outlined in Figure 4.5(b) and described in Section 4.5.1. The setup phase uses Simple Floods, and the packet forwarding phase uses RR Floods. In a Flood Burst, packets are simply sent one after the other via Simple Flood, and there is no Burst Setup phase.

Nodes turn their radios off if they determine that they are not a forwarder for the current slot. If a node receives no packets in the first N frames (where N is an estimate of the network diameter), it assumes that the slot owner did not have any packets to send and turns its radio off until the next slot starts. Likewise, nodes turn their radios off until the next slot if they determine they are not in $F_{RR}(s, d)$ during the Burst Setup.

The scheduler sits above the transport protocols. We use a simple static TDMA schedule (with equal-length slots) in this work, as our focus is on the performance of the forwarder selection and network layers. The network root (also the destination for collection traffic) sends out the TDMA schedule once per cycle, dictating how long each frame is, how many frames are in each node's slot, and how many slots are in the entire cycle. The schedule layer is where much of the work reported in LWB [21] would reside in this design.

Applications use the standard `AMSend` and `AMReceive` interfaces, allowing existing systems to easily switch over to CX.

We note that if applications require mobility, multiple sink support, minimal state-maintenance, or high interference-resistance, they can set a compiler flag to use Simple Flood Burst for both unicast and broadcast data.

4.6.2 Platform-Specific Implementation details

A detailed discussion of the network stack implementation is beyond the scope of this work, but there are a few items that may be of interest to readers.

Our platform, the "Bacon" mote, is based on TI's CC430 [65] SoC, combining a 900MHz radio core (almost equivalent to a CC1101 [66]) and an MSP430 [63] microcontroller.

Glossy achieved good transmission synchronization through deterministic execution and forwarding times across motes, while our system can leverage a fast and reliable 26 MHz radio crystal for capturing and scheduling events. We clock one of the timer modules from the radio crystal, and capture the preamble RX/TX interrupt from the radio core with it. If the packet is to be forwarded, the mote sets a timer compare interrupt for one frame-length from the capture, loads the packet into the radio's TX buffer, and puts the radio into the FSTXON state (frequency synthesizer running and ready to transmit). When the compare interrupt is raised, we issue the command to begin transmission. The CC430 can be run with a 16MHz main clock, which keeps the potential interrupt-handling jitter within the tolerances dictated by the radio symbol rate. By using a relatively long frame length ($\sim$40 ms), the radio stack has considerable flexibility in making forwarding decisions, interleaving non-radio operations, and logging performance data to the testbed.

We found significant differences in PRR between the various stock radio settings provided by Texas Instruments, and ultimately chose a 125kb/s setting which provided the best balance of reliability and speed. Since the CC430 lacks the hardware support for forward error correction (FEC) found on other radios (e.g., CC2420 [62]) we implemented FEC in software to make transmissions more robust at low-to-moderate bit error rates. Combining our

implementation of a Hamming(7,4) [3] encoding and a 125kbit/s symbol rate (2-frequency-shift-keying modulation), this gives us an effective data rate of 62.5kbit/s, one quarter that of the CC2420. In the future, we would like to find 250kbit/s (or higher) settings which work well, and we would like to use a more efficient coding scheme.

4.7 Evaluation

4.7.1 Method and Materials

We evaluate CXFS on our wireless testbed consisting of 66 Bacon motes connected to a testbed server via TMote Connect NSLUs. Figure 4.8 shows a map of this testbed. The network is physically spread over a roughly 50 m x 50 m office area, though the large open space at the top of the map forces the network to be roughly 6 hops in diameter at an output power of -6 dBm (in the single-transmitter connectivity graph, based on offline measurements of PRR when nodes take turns transmitting non-concurrently).

Unless otherwise noted, all data presented is derived from aggregating over at least 3 test runs, and each test was run for one hour. Nodes generate one packet per minute and use a slot length of 40 frames for all tests. Each frame is roughly 40 ms in length, and nodes switch from RX to Idle if no packet is received within the first 1 ms of a frame start. Radio duty-cycles were obtained by recording the time of each radio state-change interrupt with a 6.5MHz timer driven by the radio crystal. The radio was considered to be active when it was in any of the RX, TX, or FSTXON states. Nodes used -6 dBm output power for all tests unless otherwise noted, though we do validate performance at several TX powers to confirm some degree of topology independence.

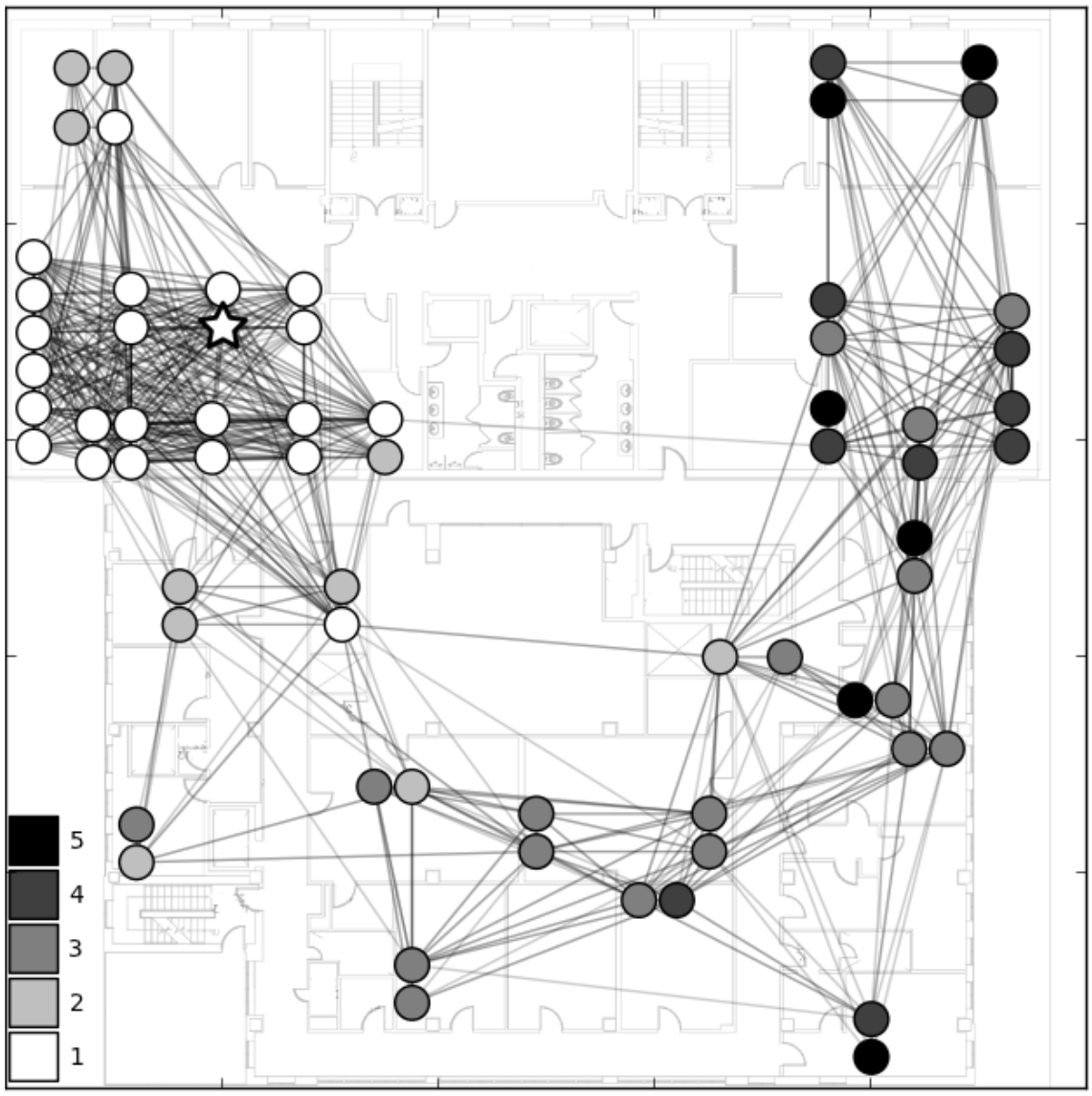
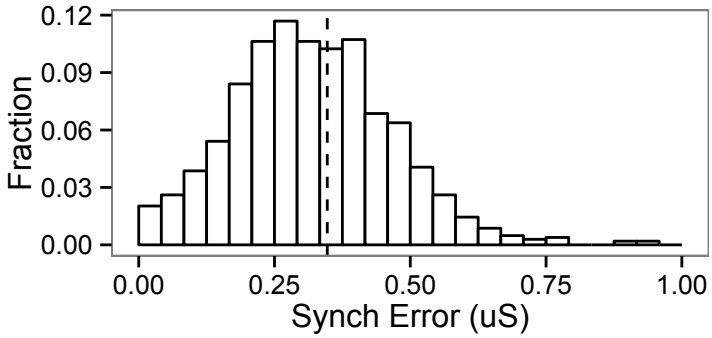
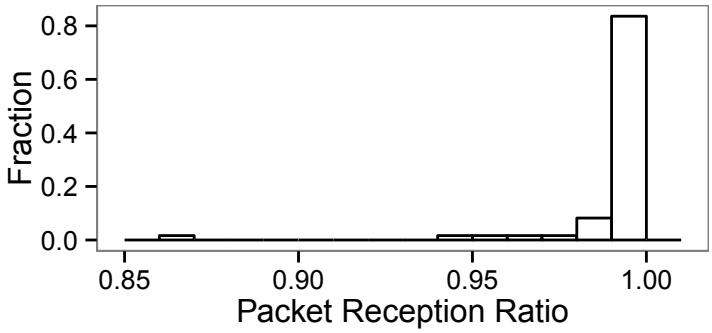


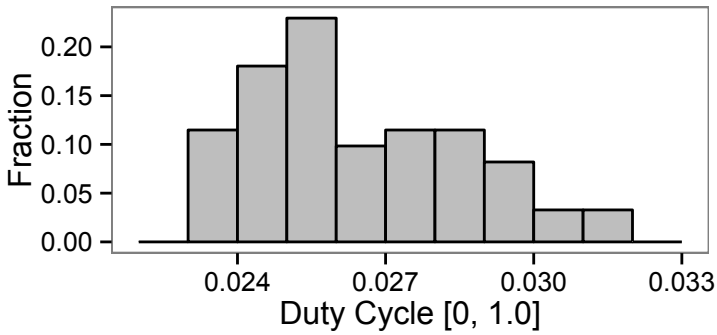
Figure 4.8: Single-transmitter connectivity graph: links shown have PRR > 95% at -6 dBm output power, with one node transmitting at a time. Labels indicate distance from root node (marked with a star). Darker colors represent farther distances, with black being 5 hops and white being 1 hop. The area shown is roughly 50 m x 50 m.



(a) Distribution of synchronization error between two concurrent senders. Mean error (dotted line) is 0.35 μs, roughly two ticks of the 6.5 MHz timer used for transmission scheduling.



(b) Histogram of packet reception ratios for each (src, dest) pair when using simple flooding. The average PRR was 99.5% and the median was 100%.



(c) Distribution of duty cycles when using Flood Burst transport on our testbed.

Figure 4.9: Baseline measurements of CX Flooding performance.

4.7.2 Baseline Performance

Our goal is to evaluate the benefits that forwarder selection provides over simple flooding. Rather than port the state-of-the-art flooding protocol to our platform and operating system, we will compare against the CX implementation of simple flooding. In this section, we justify our implementation as a fair baseline by showing that it achieves reasonable timing precision, reliability, and duty cycle, not by showing that it is categorically better than Glossy.

TX Timing Microbenchmark

Figure 4.9(a) shows the degree of timing precision we are able to achieve between two forwarders (as measured by recording the difference between the forwarders' start-frame-delimiter indicator pins on a 24 MHz logic analyer). The 91st percentile of the difference between the two transmitters' SFD signals is 0.5416 μs, somewhat worse than the 0.5μs reported in Glossy [22] with 30 transmitters. We note that the driver of multi-transmitter timing requirements is the radio's chip rate: our radio's 125 kb/s *symbol* rate should be compared against the CC2420's 2 MChip/s *chip* rate due to the CC430's lack of DSSS. In the experiments which open this chapter, we measured a bit error rate of 1.18% when two nodes transmit 1 μs apart with no capture effect (see Figure 4.3). Even the worst synchronization errors we measured are within the operating limits of the FEC mechanism, and we expect to see lower bit error rates when capture effect is present. This level of precision is adequate to support multi-transmitter flooding.

Simple Flood Performance

To demonstrate that the above results lead to reliable flooding, we perform a series of tests to evaluate Flood Burst on its own. In these tests, nodes generate a packet once every 60 seconds, and initiate a Flood Burst when they have queued 10 packets. Figure 4.9(b) shows the distribution of packet reception ratios between pairs of nodes in the network: the mean

PRR was 99.5%. While Glossy used parameter "N," for the number of rebroadcasts that each node performed upon receiving a packet, we found that our implementation provided high PRR with a single broadcast.

Figure 4.9(c) shows the distribution of duty cycles across the network, with mean/median of roughly 2.8%. To put this in perspective, the closest comparison we can make is against the FlockLab results reported in LWB [21]: their test used 54 nodes, a 120 second inter-packet interval, and a 15 byte payload vs. our 66 nodes, 60 second IPI, and 12 byte payload. They report a 0.43% duty cycle for the resulting 6.75 B/s aggregate load, while our 2.8% figure corresponds to a 12.1 B/s aggregate load. When one considers that our slower radio and software encoding cuts our effective transmission speed to 1/4 of the CC2420's, this indicates that our implementation is roughly on par with the state of the art in terms of energy efficiency.

We acknowledge that this comparison elides some of the complicating factors, but remind the readers that our goal is not to show that our implementation of flooding is better than Glossy, but to show that it works correctly on our platform and is a fair baseline for demonstrating forwarder selection.

4.7.3 RR Burst vs. Flood Burst

The goals of this work are to reduce network duty cycle for data transmissions which need only travel through part of the network, and to use distance estimates to improve inter-packet spacing. This section focuses on this impact and keeps a simple fixed TDMA schedule as described above.

As a quick sanity check, Figures 4.10 and 4.11 show how often nodes join $F_{RR}(s, d)$ for an example pair of s and d nodes and we indeed observe that nodes physically between the source and destination are the most likely to participate. However, one can also clearly see that some nodes outside of the geometrically shortest path participate with high probability, and others clearly on the shortest path participate with low probability (due to non-heterogeneity of

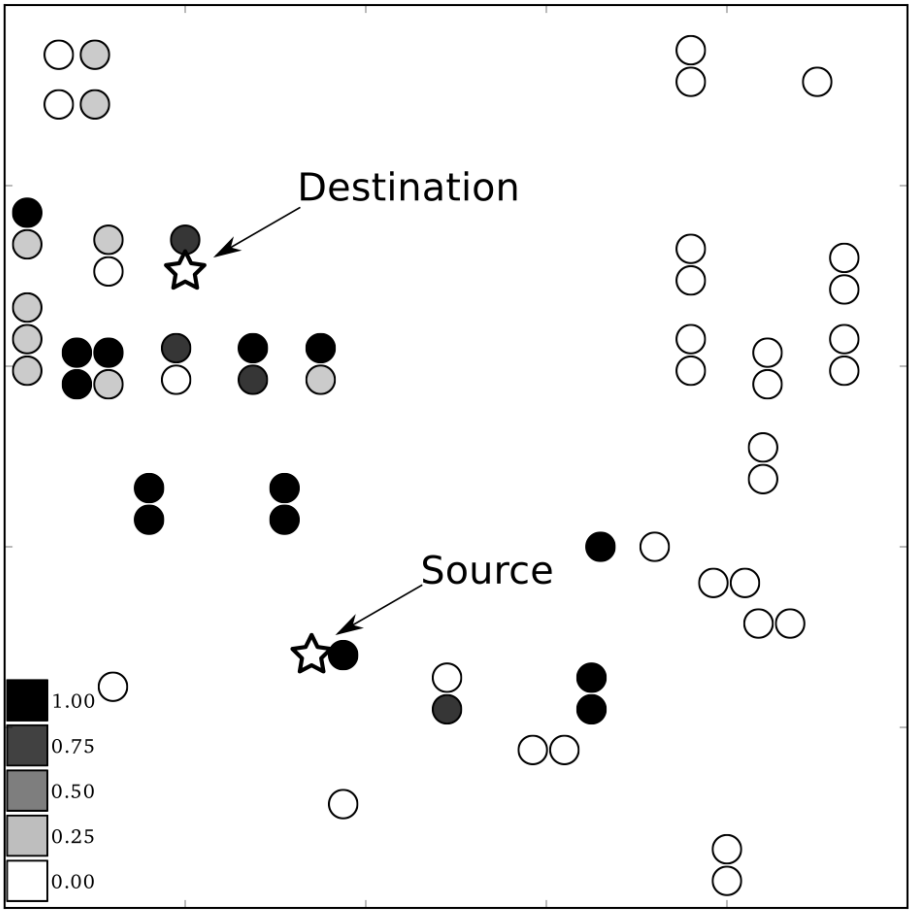


Figure 4.10: Participation frequency for a node at moderate distance. Heatmaps showing frequency of joining $F_{RR}(s, d)$ for source nodes at different distances from root. Dark colors represent higher frequencies, with black being 100% and white being 0%. Nodes used average distance with $b = 0$ to determine membership in F_{RR}.

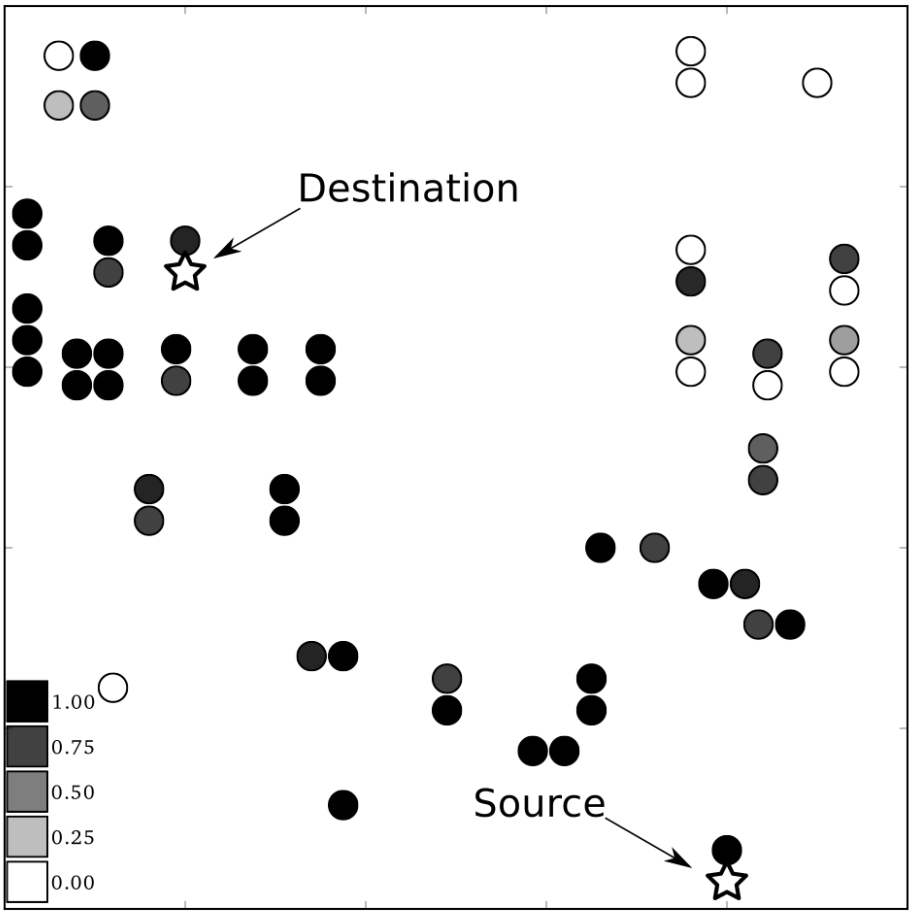


Figure 4.11: Participation frequency for a more distant node. See key description in Figure 4.10.

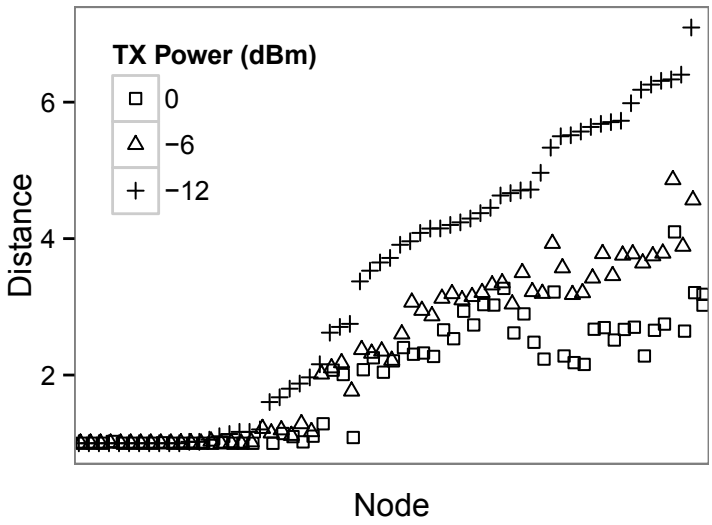
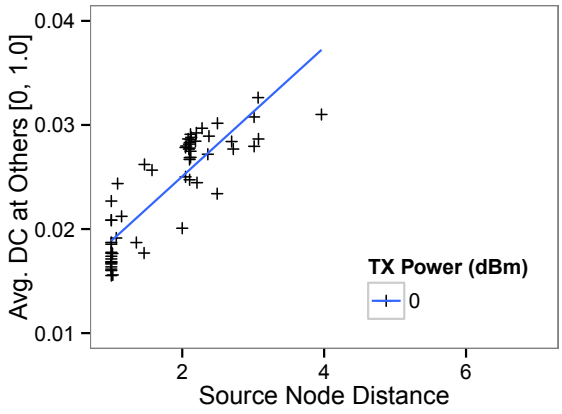


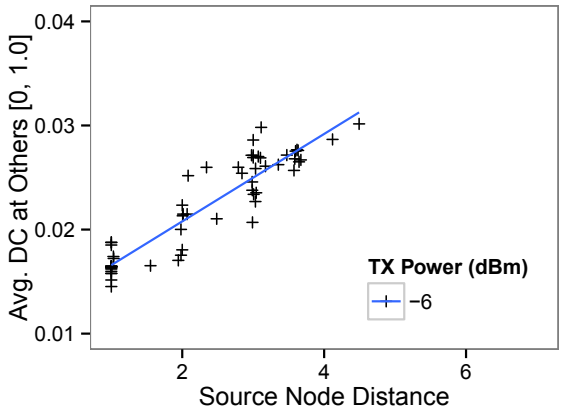
Figure 4.12: Mean of observed distances from the sink for each node at several TX power settings: each vertical stack of points shows results for the same node.

links, variability, etc.). Of particular note are the co-located node pairs where one node never participates (colored white) while the other participates frequently. These largely result from the limitations of the testbed hardware. A network-attached storage device with custom firmware is connected via Ethernet cable and power to the building infrastructure, one node is directly connected to the NAS via a rigid USB plug, while the other is connected via a USB cable of 1-2 m. While the two nodes may be spatially close, the antenna orientation is independent for them, with the rigidly-connected node vertically oriented and the second node frequently lying sideways. Since these devices are installed in academic offices, it is not uncommon for the NAS device to sit on top of a desk while the second node dangles behind furniture. A more uniform channel environment would likely show less spatial variability in forwarder selection than these figures indicate.

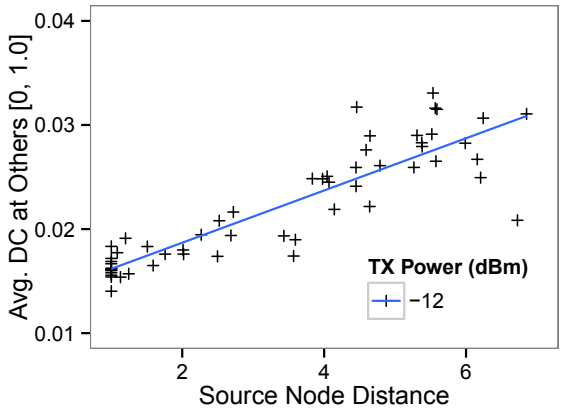
To quantify this result, we use the RR Burst protocol (with the "Average" metric, boundary width 2) to transfer data from all nodes to the sink and adjust the radio transmission power to vary the network depth and topology. Figure 4.12 shows how node distances change under different TX powers to give a sense of our testbed's size.



(a) 0 dBm TX Power



(b) -6 dBm TX Power



(c) -12 dBm TX Power

Figure 4.13: Duty cycle as a function of source-node distance. Each point corresponds to a single node's allocated slot. The X-value indicates the distance of the source node from the sink, the Y-value indicates the average duty cycle across the network during that node's slot.

In order to find the relation between where traffic originates and what impact this has on the duty cycle across the network, we first calculate the average duty cycle in each slot. Since each slot belongs to a unique node, this duty cycle is an indicator for a single node's impact on the rest of the network. In Figure 4.13 we show the average network duty cycle for each node's slot as a function of the slot owner's distance to the root at three different transmission powers.

The same trend is visible at all tested transmission powers: slots belonging to nodes close to the sink (having low depth) contribute less to network duty cycle than nodes far from the sink. Additionally, the sparser the network is, the less impact a node at a given depth has on the rest of the network. This makes sense: all other things being equal, a node at depth n will have fewer nodes between itself and the root in a sparse network than in a comparable network of higher density.

We note that the duty cycles in Figure 4.13 are for each individual slot and do not include inactive slots where the whole network is asleep. Since the duty cycles in Figure 4.9(c) also include inactive slots, a direct comparison between the two figures is not possible. However, when we calculate the overall duty cycles in these tests, we find that at -6dBm, we see a 30% improvement in the average network duty cycle (25% at 0 dBm, 20% at -12 dBm).

Figure 4.14 shows each node's normalized throughput as a function of distance to the root at the three different transmission powers. This flood throughput is computed offline by assuming the maximum observed hop-count is used for flood packet spacing: in practice, most network designers will not know this ahead of time and will conservatively set the flood TTL to some estimate of the network diameter (further hurting flood throughput). Under CX, only the setup packets of RR Burst transmissions are affected by a conservative diameter estimate. The figure shows that nodes close to the sink can indeed use their estimate of d_{sd} to increase their throughput. On our testbed, this results in an average throughput increase of 49% when nodes transmit at -6 dBm (31% at 0 dBm, and 28% at -12 dBm).

Nodes at the edge of the network will always see worse throughput under RR Burst than

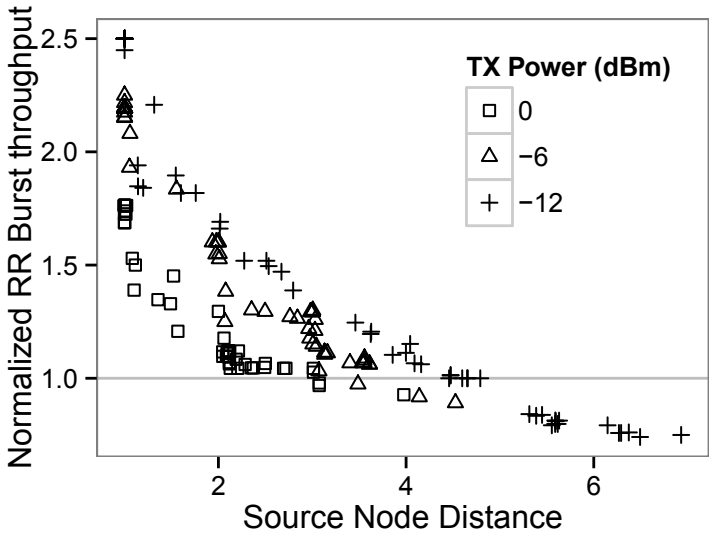


Figure 4.14: Per-node throughput as a function of source node distance. These results are normalized to flood throughput (1.0 = same as flood, $>$ 1.0 = higher throughput than flood).

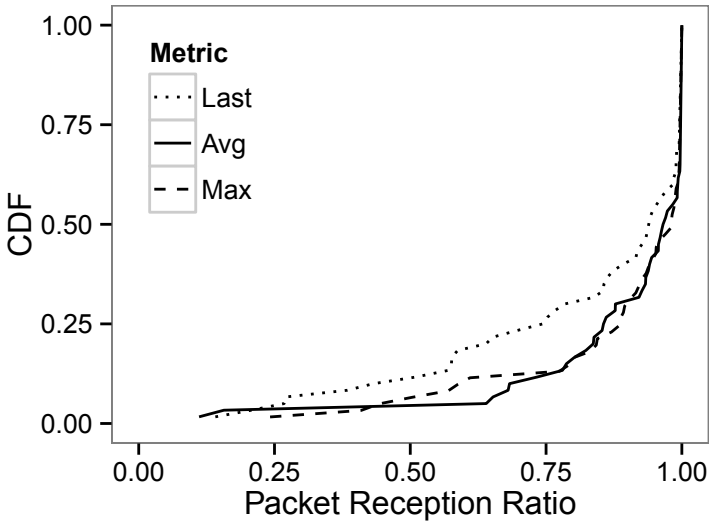
Flood Burst: they have to pay the cost of the setup phase for each burst (whereas if they used Flood Burst, they could use that time to send data). In these tests, slots were 40 frames in length (roughly five Flood Burst rounds), so the setup phase has a relatively high cost (roughly 20% of available throughput). Increasing the length of a slot in the schedule would help to further amortize this at the cost of increased latency. Such scheduling optimizations and trade-offs are an important candidate for run-time tuning in future work. That being said, when averaged across the network, throughput increases over Flood Burst for all of these tests.

Distance Estimation

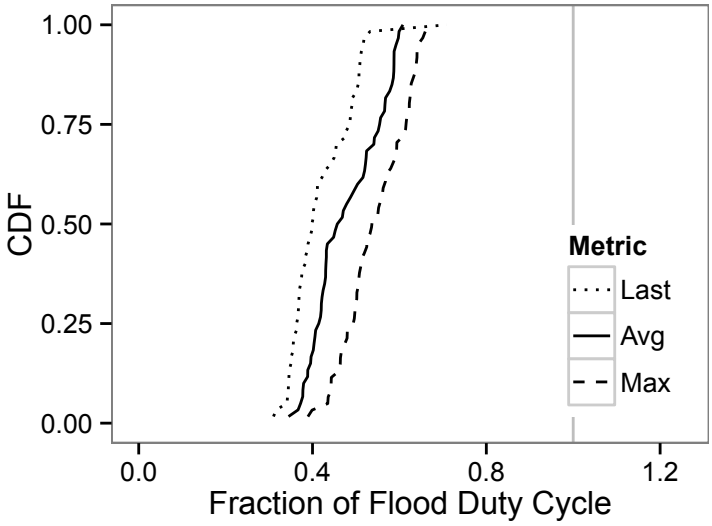
Next, we evaluate the effects of several distance estimation strategies from Section 4.5.2.

Figure 4.15(a) shows the impact of each strategy on end-to-end packet reception ratio, while Figure 4.15(b) shows their impact on duty cycle. These results were produced with boundary width $b = 0$ to isolate the effect of the metric.

As expected, “Last” achieves the best duty cycle improvement over flooding (with per-node



(a) CDF of PRR by distance estimation strategy.



(b) CDF of duty cycle improvement by distance estimation strategy.

Figure 4.15: Impact of distance estimation strategy on PRR and duty cycle. All tests conducted using boundary width of 0.

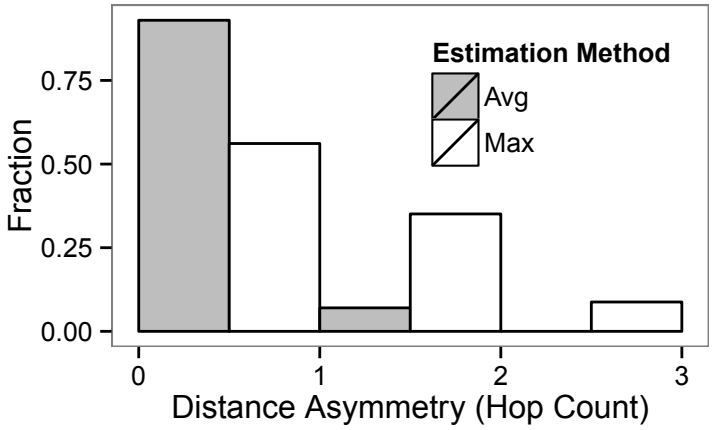


Figure 4.16: Distribution of path length asymmetry from a 1-hour test. The average root-to-leaf distance was less than 2 hops different from the average leaf-to-root distance for all nodes. The worst asymmetry was 2 hops when considering maximum root-to-leaf and leaf-to-root distances.

savings of nearly 60%), while it also shows the worst PRR. For some applications, this may be an acceptable tradeoff.

"Max" and "Average" show relatively similar performance in PRR, but the latter achieves lower duty cycles. This suggests that "Max" can be overly conservative while "Average" can strike a balance between efficiency and reliability. On our testbed, it is reasonable to set aside the 70 bytes of RAM necessary to track the average or maximum distance of each node, though in much larger networks "Last" would be the only viable option.

Boundary Width

Recall that we added boundary zone b to the shortest-path distance to account for the distance asymmetry shown in Figure 4.16 and the variability shown in Figure 4.6. As we are currently restricted to a single testbed, we don't yet know how dependent this asymmetry is on physical topology, but we can see that distances are generally not *too* asymmetric.

The boundary width essentially reflects a trade-off between efficiency (duty cycle) and reliability (PRR) as shown in Figure 4.17. Specifically, Figure 4.17(a) shows how PRR in-

creases with boundary width, while Figure 4.17(b) shows how the duty cycle becomes less efficient. When the boundary width is set to 0, we see many nodes with poor PRR due to routing failures: distance variations can easily generate cases where some nodes that are on the shortest path decide not to join F_{RR} and participate in forwarding packets. On the other hand, as boundary width increases, more and more nodes that are not on the shortest path decide to participate and waste energy.

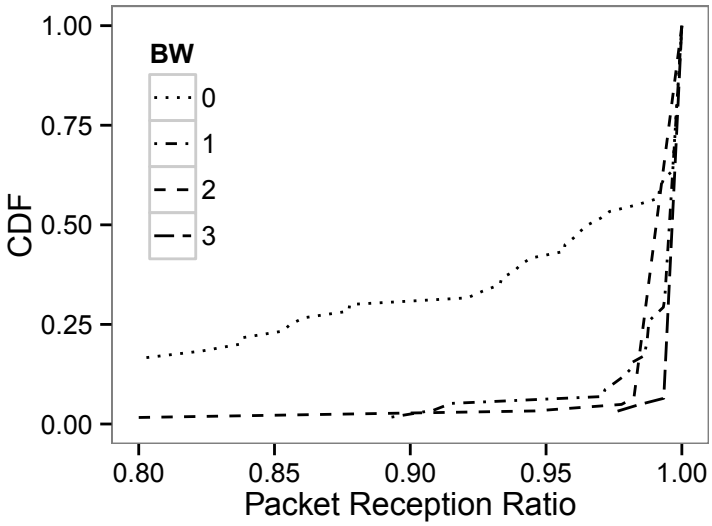
A boundary width of 2 achieves a good balance with an average PRR of 99.4% and the average node's duty cycle is only 70% of what it experiences when using Flood Burst.

4.7.4 Single-Transmitter Comparison

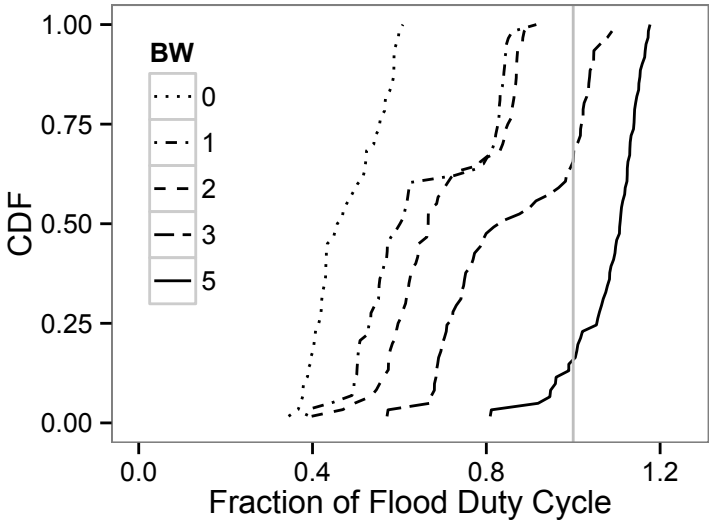
We have demonstrated that we can achieve good end-to-end packet delivery by forwarding data over F_{RR} while reducing energy consumption over F_{max}. While we don't expect to achieve energy consumption on par with F_{min}, it is instructive to look at how the size of F_{RR} differs from F_{min} and how multi-transmitter distances differ from single-transmitter route lengths, as these are the main drivers of energy consumption under CXFS. Additionally, we wish to more specifically support our claim that the redundancy present under CXFS enhances end-to-end delivery rates in the presence of unreliable links or nodes.

To investigate these questions, we collected a 24-hour trace of multi-transmitter floods on our testbed. All nodes kept their radios in RX, and a script triggered each node to send a multi-transmitter flood in sequence. The PRR for all pairs of nodes in the network was recorded, as well as the hop count of each received packet and physical layer measurements (RSSI and LQI). The records of RSSI and LQI from packets received on the first hop were used to compute aggregate single-transmitter link quality information, including packet reception ratio.

We find that many more nodes are used in F_{RR} than in F_{min} (unsurprisingly), but that the resulting inter-node distances are generally slightly shorter. On the other hand, we also



(a) CDF of PRR by boundary width. X-axis truncated and $b = 5$ omitted for clarity.



(b) CDF of duty cycle improvement by boundary width. With a 3-hop boundary zone, some nodes see higher duty cycle due to the increased setup cost of transmissions in RR Burst, and this effect is even more pronounced with a 5-hop boundary zone.

Figure 4.17: Impact of boundary width (BW) on PRR and duty cycle. All tests conducted using "Average" distance metric with large enough routing table to track all nodes.

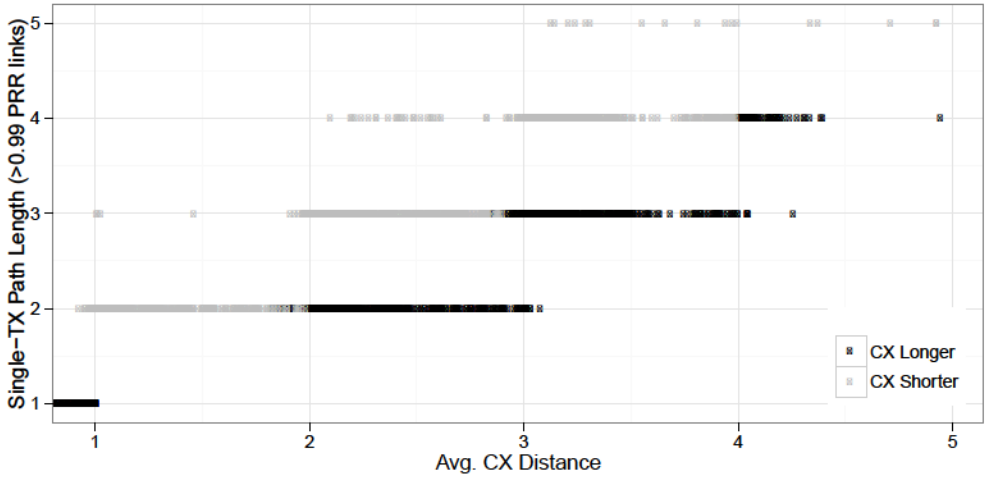
see that not only are the forwarder sets chosen with CXFS more reliable than the single-transmitter routes selected, they are also much more robust to random node and link failures.

F_{RR} and F_{min} Comparison

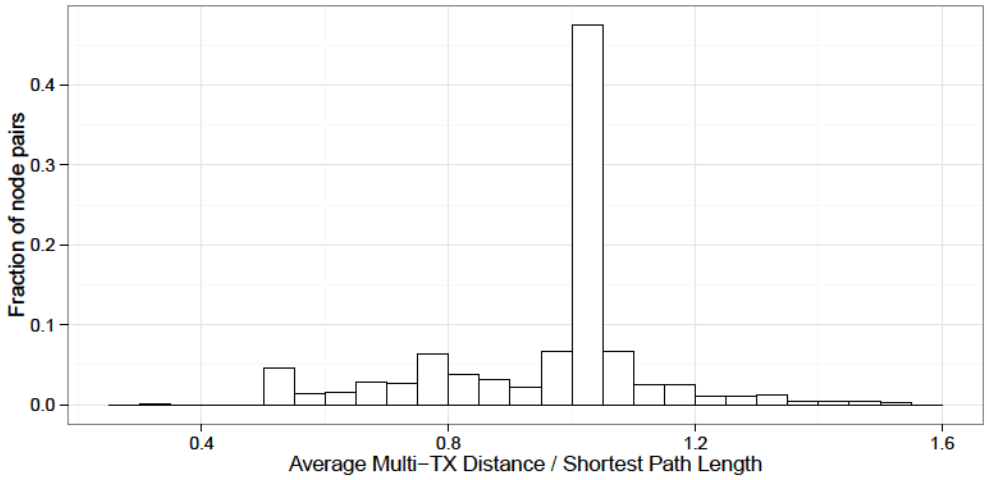
Our first goal is to evaluate how the number of nodes in F_{RR} and the multi-transmitter distances relates to the number of nodes in F_{min} and the shortest-path distances on our testbed. We used the collected traces to compute membership in F_{RR} by using the average hop count at which floods were received and setting boundary width to 2. We approximated the values of F_{min} for each pair of nodes in the network by computing the shortest path between each pair of nodes, considering only links having a PRR of 99% or higher. We note that in some cases, shorter paths were available by weighting links with ETX. However, in order to match the CX burst transmission behavior (in which each packet is sent as the previous one finishes), each link needs to be individually reliable: if a packet is slowed down because it must be retransmitted en route, intra-path interference can occur. This is not a perfect comparison, but it avoids many complicating factors specific to single-transmitter routing.

Figure 4.18 presents a comparison of CX flood distance to shortest-path length. The average point to point distance using CX is 95% of the shortest-path length, though we do see some pairs shrink in distance by 50% or incrase by up to 60%. The shorter distances are likely due to occasional receptions over poor links that essentially "jump over" parts of the network. The instances where paths are longer could be explained by the fact that when multiple 100% PRR links are exercised simultaneously, the phase offsets between transmitters and other physical effects lead to a slightly worse than 100% reception rate at the recipient. On the whole, we shouldn't expect CX to significantly hurt or help throughput over single-transmitter networking.

Figure 4.19 compares the number of nodes in F_{min} against the number of nodes in F_{RR} for each pair of nodes on the testbed. The average pair of nodes sees a 9.3x increase in size

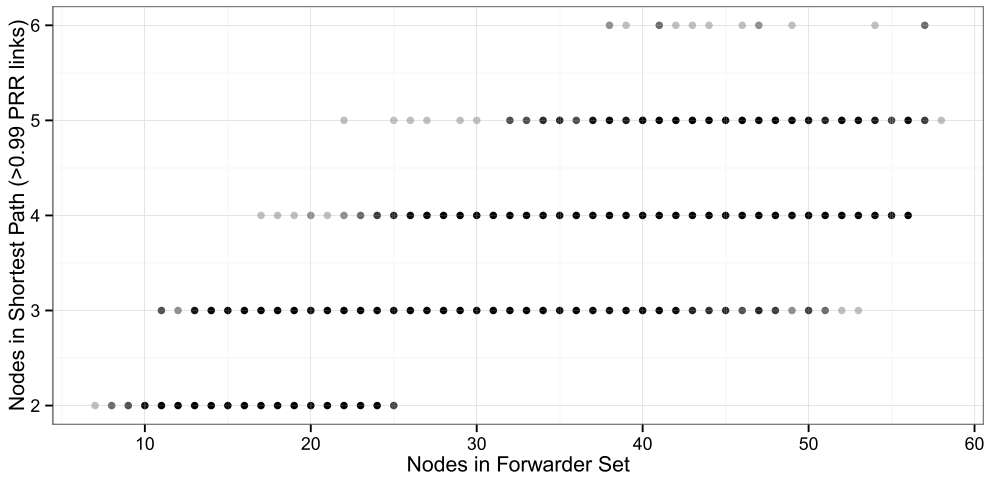


(a)

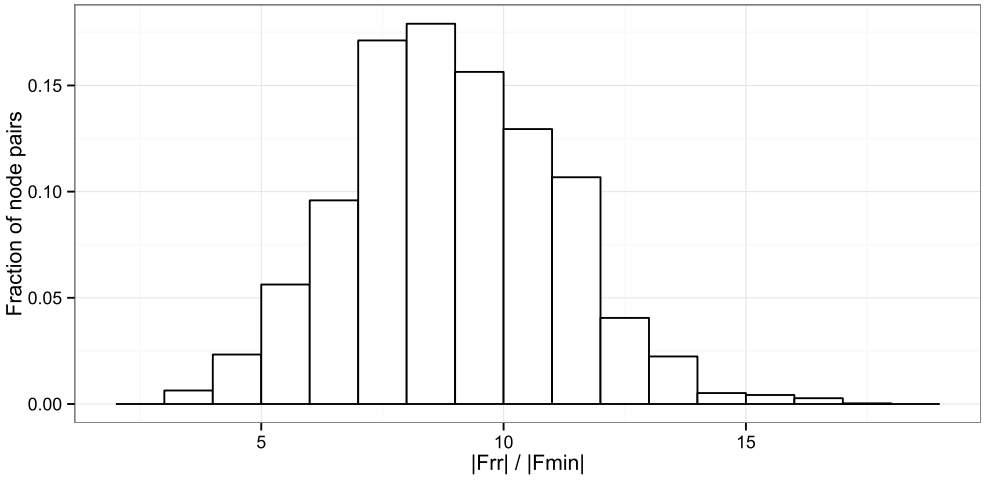


(b)

Figure 4.18: Single path vs. CX flood distance.



(a) Note difference in axis scales. Points are rendered with slight transparency to differentiate rare measurements from common measurements.



(b) Distribution of $|F_{RR}|/|F_{min}|$ for node pairs.

Figure 4.19: Single path vs. CXFS forwarder set size.

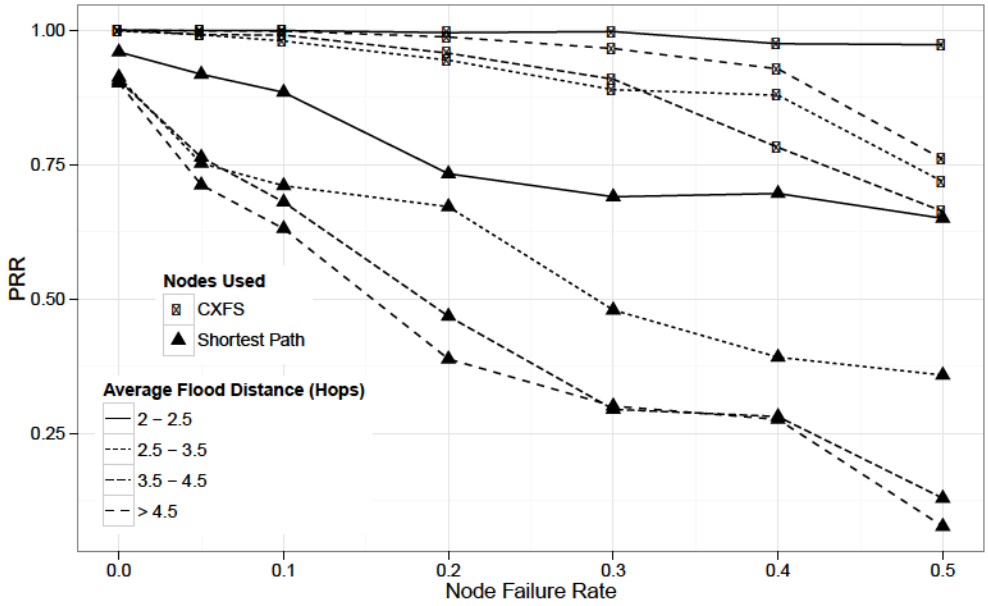


Figure 4.20: PRR as a function of node failure rate

between F_{min} and F_{RR}. Unsurprisingly, widely separated node pairs have significantly larger forwarder sets under CXFS than they do under single-transmitter shortest path selection. For pairs of nodes at opposite ends of the testbed, for example, the shortest single-transmitter path is 6 hops, but essentially the entire testbed is in F_{RR}. These figures will clearly vary from network to network, but it seems like it is not unreasonable to expect a factor of 10 increase in forwarders when moving from single-transmitter routing to CXFS. While this sounds like a step backwards, we remind the reader that CXFS avoids the vast majority of routing overhead and compares favorably to other work which has been experimentally shown to outperform single-transmitter routing.

Improved Fault-Tolerance under CX

While we do not expect CX to significantly improve energy usage or throughput over single-transmitter routing (on the basis of forwarder set size or length), we do expect that it will give better reliability guarantees, even in the face of unreliable links and nodes. We have

personally encountered situations where factors such as low batteries lead to high numbers of unstable nodes that may be selected for routing, but are unable to send and receive data reliably.

We randomly selected 10 pairs of nodes and their corresponding F_{RR} and F_{min} from our data set across the range of inter-node distances. For each forwarder set, we randomly deactivated each forwarder node with a probability f (failure rate) and measured the end-to-end packet reception ratio for a series of floods from the source node. The 140 distinct test setups were each run at least 4 times. Each test run consisted of 10 separate applications of random failures, and each of those consisted of 20 transmissions.

Figure 4.20 shows the results of these tests. Node pairs are grouped by their original inter-node distance for clarity.

The first item to note is that most of the shortest-paths selected offline actually did not have 100% end-to-end PRR. The average PRR was only 92%, indicating that even with relatively detailed PRR data and a global view of the network, short-term dynamics and link quality changes can hurt the quality of paths selected. While more sophisticated route-selection techniques may be less subject to these issues, this impact cannot be ignored.

The next item to note is that when a 5% failure rate is introduced, all of the shortest-path PRR's begin degrading, while the CX PRR remains close to 100%. In a long-term deployment, it would not be unusual to see 5% or more of deployed nodes enter a state of degraded performance, so it is encouraging to see that CX tolerates this well.

Next, we can see that the shortest-path tests are more heavily impacted the farther apart nodes are. This shows that the danger of "all-or-nothing" routing increases as more and more fragile routes are chosen. Similar behavior has been reported elsewhere in the literature [40], manifesting as diminished PRR for nodes far from the data sink.

On the contrary, we note that at the highest failure rates, the '> 4.5' series actually performs *better* than the intermediate distances under CX. Our previous results indicated that for distant pairs of nodes, practically the entire network is conscripted in F_{RR}. CXFS aims to

essentially select the nodes on a shortest path between the end points, plus nodes just off of this path. When the entire network (or close to the entire network) is in use, there may be multiple disjoint paths present. If any of these remain intact, PRR will remain high.

We can clearly see that CXFS outperforms the reliability of single-path routing for every single test. In fact, the worst CXFS test outperforms the best single-path test at every failure rate evaluated.

4.7.5 CXFS and Multi-Transmitter Flooding: Benefits and Tradeoffs

We believe that multi-transmitter flooding and CXFS are broadly applicable to collection applications in low-power sensor networks. That being said, there are important factors to consider when assessing the usefulness of these protocols. In this section, we wish to more precisely describe the overhead associated with simple flooding and with CXFS and discuss some of the inherent limitations of this approach.

Flooding vs. CXFS overhead

In our previous discussions, we pointed out that the setup cost associated with CXFS can cause some nodes to experience lower throughput and higher duty cycles than they would encounter under simple flooding.

The number of packets p a node can fit into a slot is determined by the slot length S, the network diameter D, and the distance between the source and destination node d.

When simple flooding is in use, each node can fit the same number of packets in a slot:

$$p_{flood} = \frac{S}{D}$$

When CXFS is in use a shorter distance d between endpoints allows a node to fit more packets into a slot. Regardless of d, a setup cost of two simple floods (each taking D frames to complete) must be paid.

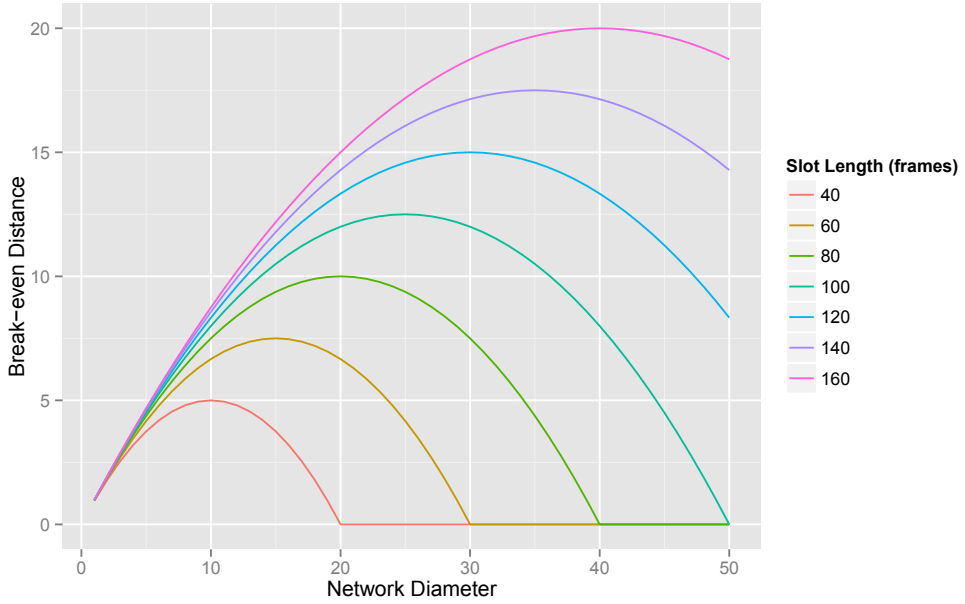


Figure 4.21: Distances at which CXFS breaks even with simple flooding for a range of slot lengths and network diameters. Nodes farther from the destination than the break-even distance will see worse performance when they apply CXFS than they would if they simply flooded data.

$$p_{cxfs} = \frac{S - 2D}{d}$$

We can set these to equal each other to obtain the break-even value for d in terms of D and S.

$$d = D - \frac{2D^2}{S}$$

Figure 4.21 plots the resulting break-even distances for a range of slot lengths and network diameters. Since each node is provided with the maximum network depth (either explicitly or by observing the sum of hop-count + TTL in packets from the root), nodes could locally compute whether it is worth it for them to apply CXFS or not.

An important item to note is that as the slot length increases relative to the network diameter, the break-even distance moves closer to the edge of the network. This makes sense:

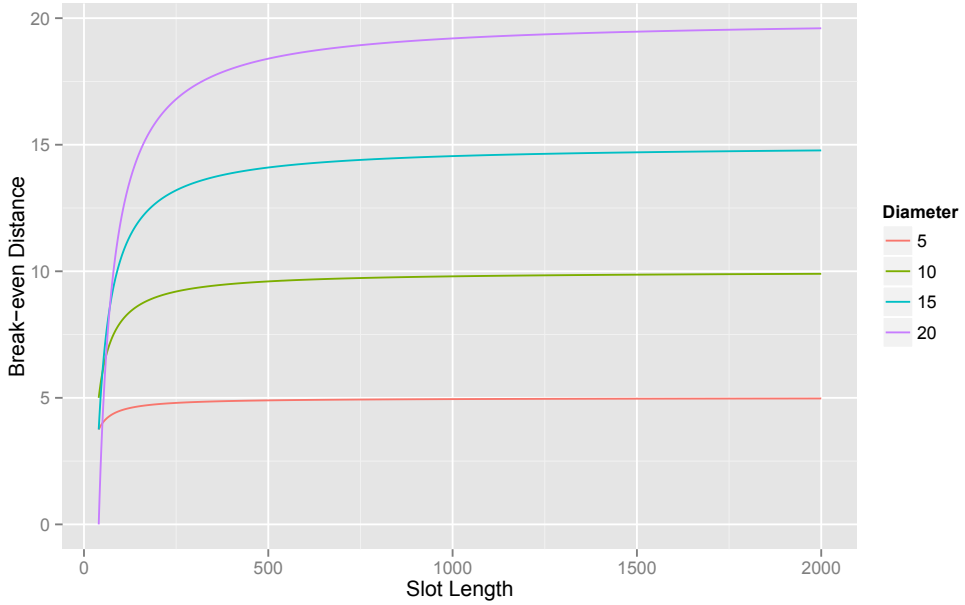


Figure 4.22: Asymptotic response of break-even distance to slot length for a range of network diameters.

the setup time decreases as a fraction of the entire slot length. Figure 4.22 illustrates this effect for a selection of network diameters. The break-even distance asymptotically approaches D as S grows large compared to $2D^2$.

Slot length and guard times

If we desire to maximize our benefit from CXFS, the above results suggest that we should strive to make slot length as long as possible, subject to the constraints of the application.

As clocks drift farther apart in long slots, progressively larger guard times have to be built in to ensure that nodes can receive packets correctly. These larger guard times add to the idle-listening cost at each node.

If we assume that clock skew is insignificant on the scale of frames (which are ms in length), nodes only have to use large guard times for the first reception in a slot (all subsequent transmissions will be based on the most recent reception from the slot owner). Under

idealized conditions, a node must start listening at $T_{slot} * maxdrift$ prior to its estimate of a frame boundary and stop listening at $T_{slot} * maxdrift$ after its estimate of a frame boundary, until it can obtain synchronization. In the worst case, this must be done in the first D frames of each slot. The worst-case idle listening time as a function of slot length, diameter, and clock skew is then $2 * D * T_{slot} * maxdrift$. If we look at this in terms of duty cycle, the idle listening duty cycle during an active slot is simply $2 * D * maxdrift$.

Our platform derives its initial frame timing from a 32KHz oscillator with a nominal precision of +/- 20 ppm at room temperature. According to the manufacturer's datasheet [1], this gives us an absolute worst-case difference between two oscillators of 353 ppm (assuming one device is at -40 C, the other is at +85 C, and their RT frequencies are at opposite ends of the tolerance ranges). Even under these extreme conditions, idle listening incurs under 1% duty cycle in active slots at networks of up to roughly 30 hops in diameter.

While this analysis will vary for different platforms and different operating conditions, it suggests that slot length should primarily be dictated by the latency requirements of the application for most practical purposes. If the time between active periods is large (e.g. for daily batch data collection), then the synchronization cost for the start of the active period will vastly outstrip the idle listening cost during the active period. The material in Chapter 5 which addresses the issues of network discovery and slot assignment opts for loose synchronization between active periods, and is not affected by clock drift during idle periods.

CX Limitations

CX confers both high reliability and low energy usage. That being said, there are some situations where multi-transmitter networking will fall short of single-transmitter networking.

When executed correctly, distributed TDMA schedules can enable spatial reuse in a wireless network: multiple nodes can transmit different data at the same time, as long as they don't interefere with each other at recipients. Many approaches have been proposed and executed in the sensor network space to perform this optimization in the single-transmitter,

single-channel context [20]. CX assumes that a single node can control the network at a time, which precludes us from reaping this benefit. By segmenting the network into distinct patches, each on a separate channel, we show how a large CX network can recover some of this benefit in Chapter 5.

CX implicitly relies on the existence of end-to-end paths in the network. Delay-tolerant and disruption-tolerant networks do not have these requirements. In sparse networks with high mobility, an approach more akin to ZebraNet [30] where nodes opportunistically transfer data when they encounter other nodes will be able to deliver data even when the network is never globally-connected. Again, in Chapters 5 and 6 we describe a system built on top of CX that does not assume a globally-connected network, but is still limited in the types of disconnections that it supports.

Likewise, CX may not be well-suited for extremely constrained energy-harvesting applications. Systems such as Tenet [49] and the system proposed by Yerva et al. [73] favor highly-asymmetric communication systems, where more powerful infrastructure nodes collect data from intermittently-powered leaf nodes. The large-scale scheduling requirements of CX (tracking active/inactive periods) would not be feasible in such a setting, where nodes are typically only able to operate for short periods of time with their radios on and are prone to intermittent failure due to lack of power.

4.8 Conclusion

Recent work from industry and academia has shown the feasibility and benefits of using multi-hop concurrent transmissions for point-to-point and convergecast traffic in low-power wireless networks. Approaches such as Low-Power Wireless Bus and Insteon offer high yield and throughput, low duty cycles, and simple operations. Despite these benefits, flood-based approaches are intuitively wasteful since they involve every node in the network for every data transfer. This work proposes adding forwarder selection to the emerging concept of a multi-transmitter network stack.

We formally defined the forwarder selection process for multi-transmitter protocols and provide simple mechanisms that nodes in a network can use to determine whether they should participate in a transmission or turn off their radio to conserve power. Further, we present the CXFS protocol that shows how the forwarder selection process can be adapted for real networks with asymmetric and time varying links.

Results from our 66-node testbed show that CX reduces average duty cycle by 30% and increase average throughput by 49% over simple flooding while preserving a 99.4% average end-to-end packet reception ratio. CX continues to deliver data reliably where random node failures and stale link quality information cripple single-path routing.

Chapter 5

Multi-tiered Networking with CX

While the results in Chapter 4 show that we can achieve high reliability and data rates by using CX and forwarder selection, the described approach falls short of an ideal solution to our application for a few reasons.

First, we consider the presence of reliable power and communication infrastructure to be the exception rather than the rule. For this reason, whatever mechanism we use for coordinating nodes with each other, it should be robust to the case where the base station is absent for long periods of time. The network coordination should robustly handle the case where unreliable nodes enter and exit the network at random.

Second, typical WSN deployments for environmental monitoring are naturally "patchy." Not only is it easier to make network schedules if the patches are coordinated separately, but we can also realize energy savings. Systems which schedule the entire network together tend to see energy consumption scale with the size of the full network. By subdividing the network, we aim to scale energy consumption at end-devices with the size of their subnetwork.

Finally, the hardware that we describe in Section 6.2 allows us to deploy a subset of nodes as "routers" to handle long-range transmissions, reducing the need for sensing nodes to participate in data forwarding for anything outside of their patch. Using such routers, we can concentrate energy usage at a few points in the network.

In this chapter, we detail the design of a multi-tiered collection protocol built on CXFS and evaluate its performance. We segment the network into multiple patches and use CXFS to

collect data from each patch at a router, then use CXFS to collect data from all of the routers at a basestation.

5.1 Multi-Tiered CX Design

Our high-level design goals in this data collection system are to maintain high end-to-end reliability while reducing energy consumption at sensing locations.

The energy consumers in a low power networking protocol can roughly be broken down into forwarding load (how much time nodes spend sending and receiving data) and control overhead (how much traffic must be sent to coordinate data transfers). Both of these scale with the size of the network: larger networks produce more data (which must be forwarded) and larger networks require more messages to coordinate.

Our general approach will be to subdivide the full deployment into multiple patches. We designate some nodes as sensing locations (or *leaves*) and designate others as *routers*, where each patch consists of one router and zero or more leaves. Routers *independently* collect data from the leaves in their patch. A *basestation* node periodically downloads the collected data from each router without relying on the leaves to perform data forwarding. We use the technique of CXFS described in Chapter 4 to perform each of these data collection steps reliably and efficiently.

Figure 5.1 shows a schematic design of such a network. Each download, whether from leaves to a router or from routers to a basestation, follows the same basic pattern. This flow is shown in Figure 5.2. All nodes are normally in a low-power idle state. Periodically, the collection point, or *sink*, for the download initiates a wakeup process. Once the network is active, the download proceeds in a series of *slots*. The sink determines which node will be assigned the next slot (if any) and sends a *Slot Assignment* or *SA* message to it. During its slot, a node sends any outstanding data it may have. When the sink determines that no more nodes have outstanding data, it stops assigning slots and returns to the idle state. After a few slots elapse with no SA, the rest of the network returns to the idle state as well.

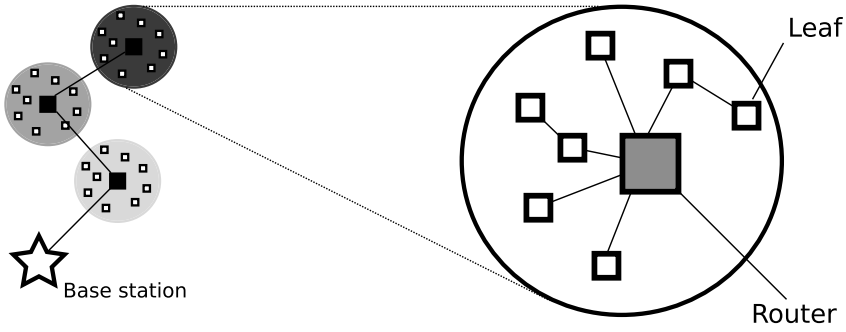


Figure 5.1: Multitier CX network. Different patches perform subnetwork collections on different channels (noted by color), all routers communicate with the base station on a common channel.

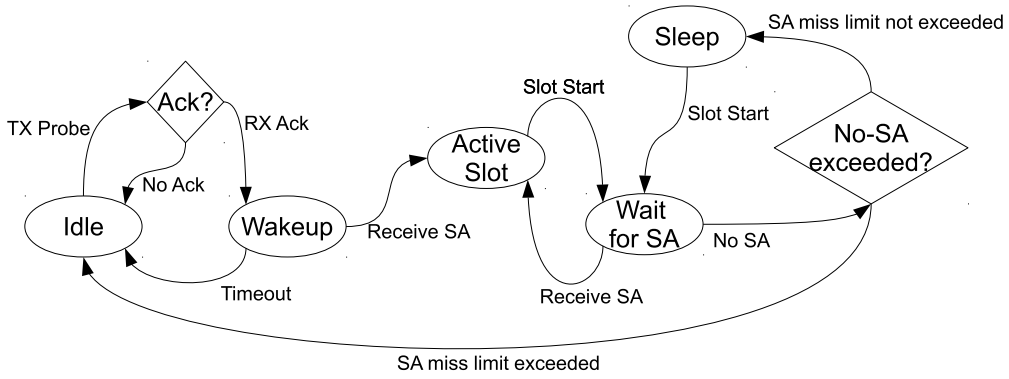
In the following sections, we will look at each portion of the download in detail.

5.1.1 Wakeup

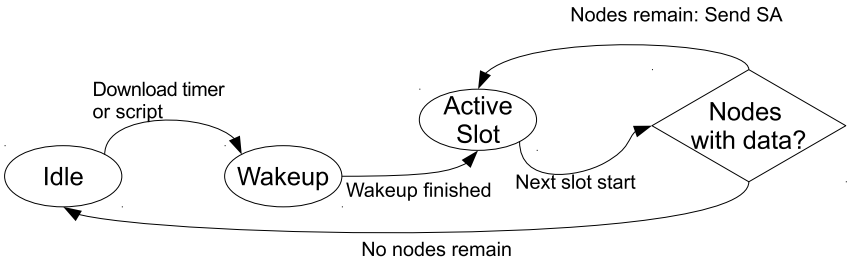
Our first practical consideration in designing this collection system is coordination in the absence of infrastructure. Existing collection protocols [21] rely on the existence of a single root node. In CTP, for example, nodes which lose their parent in the tree can expend signficant energy searching for a new parent [24]. In LWB, the coordinator node's failure leads to a period of resynchronization while a new coordinator takes over, and considerable complexity emerges from this process [21].

Rather than trying to keep all nodes coordinated at all times, we aim to make the normal uncoordinated state low in power consumption.

To do this, we adapt Low Power Probing [44] to fit into the CX networking paradigm. Every node in the network is assigned to one or more channels: one for their specific subnetwork, one for the routers, and one *global* channel. A node periodically sends a packet with TTL of 2 on each of its channels and then checks to see if it is retransmitted. Normally, the network is idle and they will not see it retransmitted. When a router wishes to download from its subnetwork, it starts listening on its subnetwork channel and forwards wakeup probes as



(a) Slave node cycle



(b) Master node cycle

Figure 5.2: State diagrams for the main phases of the multitier CX collection protocol.

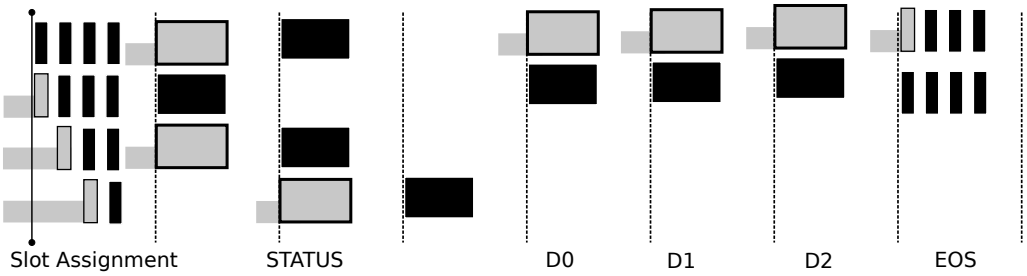


Figure 5.3: Phases within each slot of the active period. The x-axis is time, which increases from left to right. Tall black rectangles indicate transmissions, tall gray rectangles indicate packet receptions. Short gray rectangles indicate periods where a node is in RX mode but is not actively receiving a packet. Nodes use the arrival of the SA packet to determine the timing of their receive checks for the remainder of the slot.

it receives them. When nodes hear their wakeup probes being forwarded, they start listening and forwarding any probes that they hear. After a number of probe intervals equal to the distance of the farthest node in the network from the root, the entire network has been woken up. The same process is used when the basestation wishes to download from the routers (on the router channel), or when the basestation needs to contact every node in the network (on the global channel).

In this manner, we keep the idle radio duty cycle low and pay a modest cost when we need to initiate a download. Section 5.3.2 looks at the costs associated with this technique.

5.1.2 The anatomy of a slot

Each slot begins with a *Slot Assignment* (`SA`) message CX flood from the download initiator (either a router or basestation) which designates the owner of the current slot. These packets are a fixed short length, and we send them with a short retransmission interval (5.7 ms). Nodes wait in RX until they either detect an `SA` message or determine that the possible time when an `SA` could be received has ended, based on a pre-agreed maximum network depth and the retransmission interval. Every recipient uses their estimate of the original transmission time of the `SA` packet, along with a fixed and pre-defined frame length, to determine the points in time when transmissions may occur for the rest of this slot. The initial `SA` RX

timeout is relatively long, which allows nodes to have loose synchronization with the root between slots and still maintain connectivity.

At the next frame boundary, the recipient sends a `Status` message back to the root. The `Status` message conveys the second half of the forwarder-selection criteria (the end-to-end and source-to-forwarder distances), a copy of the owner's neighbor table (for network discovery), and a flag to indicate whether it has data pending or not. The rest of the nodes either immediately sleep until the next slot (if no data is pending) or use the included distance information to determine whether or not they need to stay awake to forward data.

The main period of the slot is the same as described in Chapter 4– the source transmits packets back to back, using the end-to-end distance to determine the interpacket spacing. Forwarders check for channel activity slightly before each frame boundary, and immediately return their radios to sleep until the next frame if no activity is detected.

An active slot (one in which the owner indicated data was pending) ends with an end-of-slot (`EOS`) message. This is used to indicate to the sink whether there is still more data pending.

5.1.3 Network Discovery and Slot Assignment

The original CX work assumed that all nodes were known ahead of time, that they had been assigned a slot in the download for their use, and that they agreed on a common wakeup schedule. We've addressed the problem of coordinating the network's active period and described what happens in a particular slot, but have not yet addressed the problems of slot assignment and node discovery.

During the wakeup period, nodes record the source of each probe that they overhear in a *neighbor table*. Every node that wakes up must have had a probe heard by at least one other node. So if the size of this neighbor table was large enough, we would be guaranteed to have a list of every active node somewhere in the network (appearing in the neighbor tables of one

or more nodes). Since we don't necessarily know ahead of time how many radio neighbors a node may have (and since we have a limited amount of memory to work with), we use a simple eviction policy that reliably satisfies our need to enumerate the network's members.

Each node maintains `index`, a pointer into the next free entry in their neighbor table. This starts at 0 when a node is woken up, and gets incremented whenever a probe is inserted into the neighbor table. If the number of recorded neighbors exceeds the size of the table `tableSize`, `index` wraps back to 0. After this point, when a new probe is heard, we randomly choose to either ignore it or insert it at `index` before incrementing `index`. The probability of ignoring the probe is set to `index`/`tableSize`. This ensures that 0^{th} entry is never evicted, and the earlier an entry was recorded, the less likely it is to be evicted. The intuition behind this is that probes sent early in the wakeup process have fewer active recipients that may record them, while probes sent late in the wakeup process have potentially many nodes that receive them. A simple first-in-first-out policy fails to effectively enumerate the network, as the nodes closest to the base station are the most likely to be evicted (quickly leading to undiscovered sections of the network) [1].

The slot assignment order is implicitly determined by distance from the root. It starts with the nodes within immediate radio contact of the root, and as `Status` messages are received, the newly-discovered nodes are appended to the assignment order. If a node couldn't be reached, or if it indicated that additional data was pending in its `EOS` message, the root will grant it another slot when it finishes cycling through the network.

This process continues until no nodes have data pending. A node is considered to be finished if it explicitly indicates no data is pending (in either a `Status` message or an `EOS` message), a pre-defined number of attempts to reach it have failed, or a pre-defined maximum number of slots per download have been assigned to it. These safeguards are in place to prevent aberrant conditions from allowing a node to keep the network active for long periods

[1] Assume you have a basestation with a single neighbor A in the network. A has n neighbors, and `tableSize` is $m < n$. Let B be the first of these n nodes to send a probe after A has woken up. B will be recorded at index 0 in A's table, and will not be recorded anywhere else. Under a FIFO eviction policy, once the m^{th} neighbor of A sends its probe, B will be evicted.

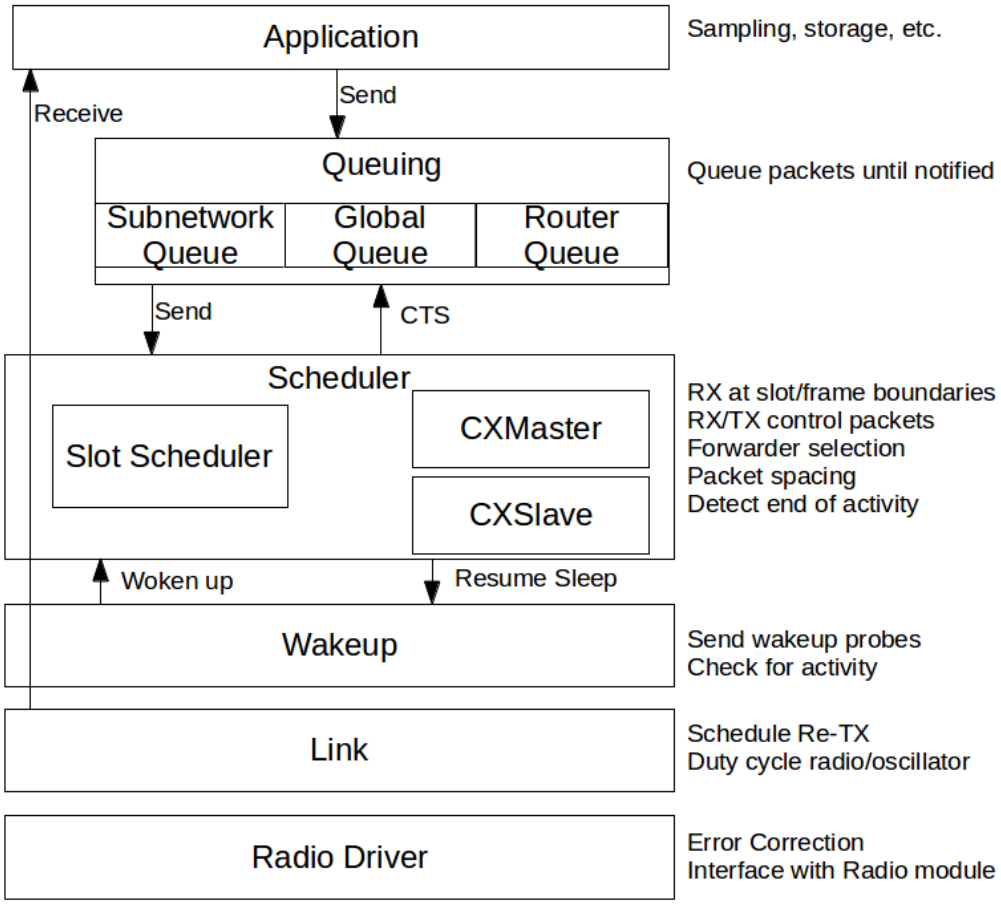


Figure 5.4: CX stack for multitier networking.

of time.

Nodes consider the download to be complete if they perform a pre-defined number of `SA` checks without detecting activity. This limit is set based on how reliable the network is. Testbed results have indicated that a early shutdowns are very rare when this limit is 4 or higher, though networks with lower end-to-end PRR may need to increase that threshold to account for occasional packet losses.

5.2 Software design

Moving from the original CX implementation to one more appropriate for field deployment leads us to some design changes from the implementation described in Section 4.6. Figure 5.4 shows the high-level view of the network stack. While the previous network software stack was designed to parallel an OSI-style layered protocol, the current network software stack is more closely aligned to the functional needs at the mote.

We need to support non-frame-aligned transmissions for the wakeup process described in 5.1.1 and variable-length receive timeouts (for `SA` and `EOS` packets as opposed to `Status` and `Data` packets). This forces us to strip down the link layer to its bare minimum: it is now responsible only for handling the precise timing of retransmissions and keeping the radio core and related peripherals in low-power modes when possible. The coarse-grained timing of receptions and the initial timing of non-concurrent transmissions is moved up the stack.

The Wakeup component is responsible for controlling the timing of wakeup probes and initiating the wakeup process (for routers and base stations), while the Scheduler is responsible for the behavior in active downloads. The Scheduler needs to be able to put the node back into its low power state at the end of a download and get notified when it is woken up. They can be thought of as residing at the same level of abstraction in a sense (one handles idle behavior, one handles active behavior), but the TinyOS conventions of down-calls (commands, e.g. "return to idle mode") and up-calls (events, e.g. "woken up") leads us to treat them as layered.

The Scheduler component coordinates the timing of the events described in 5.1.2, sends and receives control messages, and indicates when `SA` messages are received. The logic in this component is decomposed into master-specific, slave-specific, and role-agnostic subcomponents (CXMaster, CXSlave, and SlotScheduler, respectively). It absorbs the duties previously divided between the Network and Transport layers, as they are quite tightly coupled in practice.

In order to keep the interfaces presented to the application simple, we introduce a Queuing component above the Scheduler. This abstracts the logic of determining when a packet should be sent (e.g. when the destination's network segment is active) from the rest of the Scheduler logic. The Queuing component presents a standard split-phase `Send` command and matching `SendDone` event to the application level code.

The following subsections present the most important implementation details and design choices for each of these layers.

5.2.1 Link Layer

Hardware independent behavior

The Link layer, as before, interacts with the radio's hardware presentation layer (HPL) and precision timing source to schedule the time-sensitive events in CX. Our goal in this layer is to present a simple interface to higher levels of abstraction that wish to receive or send packets at specific times.

When sending packets, the radio core's transmit buffer is loaded and then the transmission command is issued either as soon as possible (for wakeup probes) or at a time designated in the packet metadata (for frame-aligned transmissions). Upon receiving the `sendDone` event from the HPL, the packet's time-to-live (TTL) counter is decremented. An alarm is set to coincide with the packet's last transmission (frame length $*$ TTL). When this expires, the `SendDone` event is signalled to the caller.

The receive behavior parallels this closely, and in fact reuses many of the same code paths. Reception has to be explicitly requested from a higher level by calling an `RX` command. This accepts a timeout length, which may be on the order of milliseconds (for frame-aligned data transmissions) or for long periods of time (e.g. the multiple-second wakeup period). This command puts the radio into RX mode and sets an alarm for the specified timeout. When the alarm fires, an `rxDone` event is signalled immediately unless channel activity has been

detected. Upon receiving a packet, a node decrements its TTL, increments its hop-count, and loads it into the transmit buffer. Its subsequent retransmission is scheduled for one frame length from its reception, and the standard `Receive` interface is scheduled to coincide with the packet's last transmission.

Choosing to wait until a packet's TTL reaches 0 before signalling its reception or transmission greatly simplifies logic at higher layers. As soon as the Scheduler gets a `sendDone` event, for instance, it can send the next packet. As soon as it `receive`'s an `SA` packet, it can `send` its `Status` packet.

In order to optimize performance for the various short packets used in multi-tier CX (wakeup probes, `SA` messages, and `EOS` messages), we use a shorter frame length for packets having total length of 18 bytes or fewer. In order to support wakeup probe behavior, we also add a flag to the packet metadata which lets the sender indicate that it wishes to receive the `sendDone` event as soon as the packet is transmitted. This allows a node to send a probe with TTL 2, switch to RX mode, and then check for its retransmission by neighbors.

The `RX` command can also be invoked with an option that disables retransmission: this exists so that a node can sniff packets without spreading them further in the network. We need this to support the Phoenix [26] time reconstruction protocol, though other systems may not need this feature.

The radio module is switched into IDLE mode when not in use, and can be put into SLEEP mode through an explicit command invoked by higher layers. It also provides pass-through access to the HPL set-channel command.

These factors should be largely independent of the hardware implementing a similar system.

Platform-Specific Details

We found that packets having fewer than 64 bytes (one full TXFIFO) exhibit a variety of length-dependent timing changes that complicate the link layer logic. When measured on a

logic analyzer, it appeared that the time between issuing a TX strobe and the time that the transmission actually began differed depending on whether a node had just switched from RX to TX, or whether it was retransmitting a previously-transmitted packet. Without being able to understand how the radio module was internally handling these cases differently, it was determined that the safest way to proceed was to pad packets to either a fixed short-packet length or a minimum long-packet length (in practice, most packets are large enough to not need padding). For the short packets, we allow a node to retransmit it more than once before returning the radio to idle. This is a somewhat inelegant solution, though the short duration of these packets lets us do this without significant damage to the overall duty cycle.

We use a 26MHz crystal for all timing-critical operations. This isn't free to operate, consuming up to a milliamp of current while running. We explicitly turn this off when there is no ongoing or scheduled transmission or reception.

5.2.2 Wakeup Layer

The Wakeup layer is primarily responsible for sending wakeup probes and determining which network segment is active. A configuration structure (see Section 6.3.1) dictates the corresponding channels for each network segment, as well as a wakeup probe frequency for each segment. We use this information to set a timer to fire, on average, with the specified interval. When it fires, the Link layer is switched to the corresponding channel, and a probe packet is sent with self-retransmission disabled. After sending it, the Link layer is immediately switched to RX mode with a timeout specified by the short-packet frame length to check for retransmission. If this results in receiving ones own retransmitted packet, we signal a `wokenUp` event to the next layer above, and indicate which network segment was woken up as a parameter. Further probe transmissions are disabled until this layer is explicitly put back into a sleep state.

We include local timing information in probe packets and support a probe-sniffing mode

in order to enable the Phoenix timestamp reconstruction protocol [26].

Additionally, we enforce a black-out period in which no probes are sent after the layer is put to sleep. This is in place so that if a node loses synchronization with the network, it is less prone to disrupt an active download by sending probes on the same channel.

5.2.3 Scheduler Layer

The Scheduler layer coordinates the timing of the events described in 5.1.2, as well as the role-specific (master or slave) behavior relevant for each network segment.

A common `SlotScheduler` module controls the timers which dictate when a slot begins, when the node is in the `SA` period of a slot, and when valid receive and transmit times occur. When this module receives a `wokenUp` event from the wakeup layer, it keeps the radio in RX mode until the wake up period has completed (determined by a maximum network depth and probe interval for the active segment in the same configuration structure described in 5.2.2) and the first `SA` message has been received or sent. It uses the time of this event to set a periodic slot-started timer (which it re-adjusts at each `SA` reception/transmission).

A `SlotController` interface, parameterized by the active network segment, determines the node's role in the download. A router, for instance, has a `CXMaster` instance wired to the subnetwork index of this interface, while it has a `CXSlave` instance wired to the router index of this interface. `SA`, `Status`, and `EOS` messages are handled in the same manner whether a node is master or slave, the only difference is who is sending and who is receiving.

The `CXMaster` slot controller implements the node discovery and slot assignment logic described above. It maintains a list of all the nodes discovered so far in the network (starting with itself), and as `Status` messages are received it appends discovered neighbors to its contact list. Each entry in this list tracks a node ID, the number of failed attempts to contact that node, and whether it may have data pending. At the start of each slot, the `SlotScheduler` component checks the active `SlotController` interface for whether it is

responsible for sending or receiving a SA message, and prepares the relevant SA message if needed.

The CXSlave slot controller tracks the node ID which is acting as a master for its network segment: this is necessary so that a node can accurately specify the destination for its packets (and allow other nodes to perform forwarder selection).

This layer is linked to the send queuing and dispatch layer by the Send and CTS interfaces. The CTS interface is signalled when an SA message is received, indicating that the queuing layer can Send the next packet. When the queueing layer calls Send, it will receive an ERETRY response if the current slot is not assigned to this node, or if there is not enough time left in the slot for a packet to be sent (e.g. there are 2 frames left, but the destination is 3 hops away). If a packet is accepted for transmission, then it is temporarily held at the scheduler layer until just prior to the next frame boundary. At this point it is given a 32KHz resolution transmission time exactly matching the next frame boundary and passed down through the wakeup layer to the link layer.

Some care is taken to set short RX_SLACK and TX_SLACK times which determine how far in advance of a frame boundary a node starts listening for packets or loads its transmit buffer. Nodes agree on a common notion of frame boundaries (based on the actual SA packet transmission), and then set their frame timers to fire prior to these boundaries. This lets us account for both the jitter in time measurement across the network (potentially off by up to 1 32KHz tick, even in the absence of clock skew) and the jitter in event handling time (dependent on the other tasks that a node is doing aside from running the networking protocol).

5.2.4 Send Queuing and Dispatch

The contents of a well-written networking stack should be invisible to the application developer. With only minor modifications, we expose the TinyOS-standard AMSenderC and AMReceiverC generic components to the application level.

In our experience, each node in the network sends all messages of a given type to one particular network segment. Leaf nodes send data from their log to their router, routers send data to the base station, and requests for missing data go in the reverse direction. With this in mind, we replace `AMSenderC` with network-segment specific generic configurations: these correspond to "send a packet of type X on network segment Y." While the details of this implementation are specific to TinyOS, the principle of separating the per-segment queuing logic from the Scheduler logic is broadly applicable to other systems following this design.

Each client instance connects to a modified version of the standard `AMQueueImplP`, which queues outstanding sends from `AMSenderC` clients and sends the next packet as the preceding one completes. In our modified `AMQueueImplP`, a `send` command which fails with `ERETRY` is queued. No further attempts are made by `AMQueueImplP` until it receives a `CTS` event from its corresponding network segment. When it gets a `sendDone` event back in response to such a `send`, it will immediately try to send the next outstanding packet if it exists.

In order to keep logic as simple as possible, we have to violate one of the TinyOS design principles by invoking a down-call (the `Send` command) from an up-call (the `CTS` event) [38]. This is generally frowned upon, as it can lead to infinite loops, deep call stacks, and long execution times. However, this lets a node immediately compute the data-pending bit in its `Status` message as soon as it receives an `SA`. We can (and do) attempt the next pending send in a successive task context after the preceding `sendDone` event is handled, as the logic in the scheduler layer will either accept the next `send` if there is enough time to handle it or reject it and indicate that data is still pending in its `EOS` message.

5.3 Evaluation of multi-tiered CX

5.3.1 Baseline Performance Validation

The results presented in Chapter 4 measured the behavior of forwarder selection in an isolated setting. They did not take into account scheduling or network discovery, for instance.

Before we proceed, we need to establish some rough benchmarks for the revised CX implementation presented in this chapter. All tests were conducted on the same indoor testbed, though due to hardware failures there are now only 58 nodes available.

For a baseline comparison to the previously-reported results, we simulated a 1 minute packet-generation rate on a flat (non-tiered) network. Each download, every node tried to send 50 packets to the root. The slot length was adjusted to fit 4 flood packets, as this is how many packets fit into each flood burst under the previous implementation. A boundary width of 2 was used when forwarder selection was applied. The maximum network depth was set to 10, and the probe interval was set to 1 second.

Baseline Packet Reception Ratio

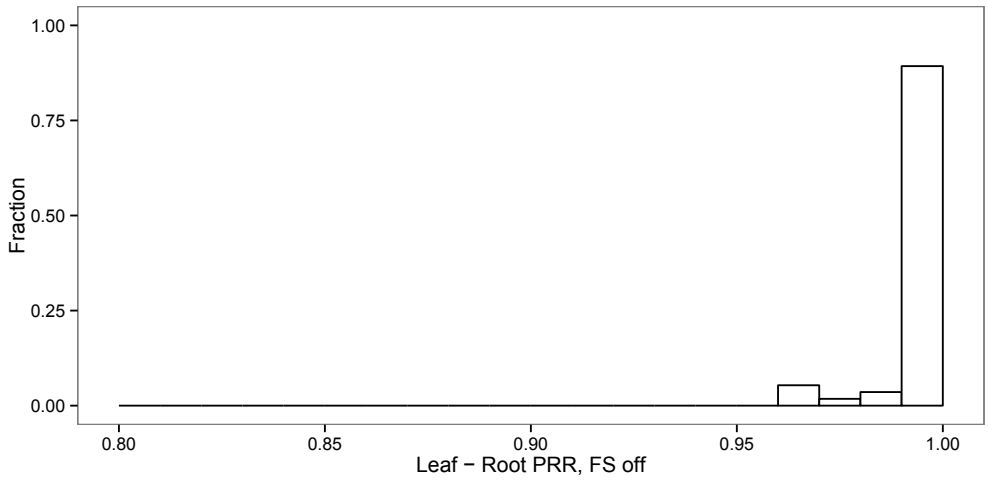
Figure 5.5 shows histograms of the end-to-end packet reception rate in each direction without forwarder selection, and Figure 5.6 shows the leaf to root PRR with forwarder selection enabled. The mean end-to-end PRR is above 99.5% from leaf to root (with and without forwarder selection), while the root to leaf PRR is 98.7%. This appears to be due to poor connectivity at a single node. The revised CX implementation achieves comparable PRR to the original implementation.

Baseline Duty Cycle

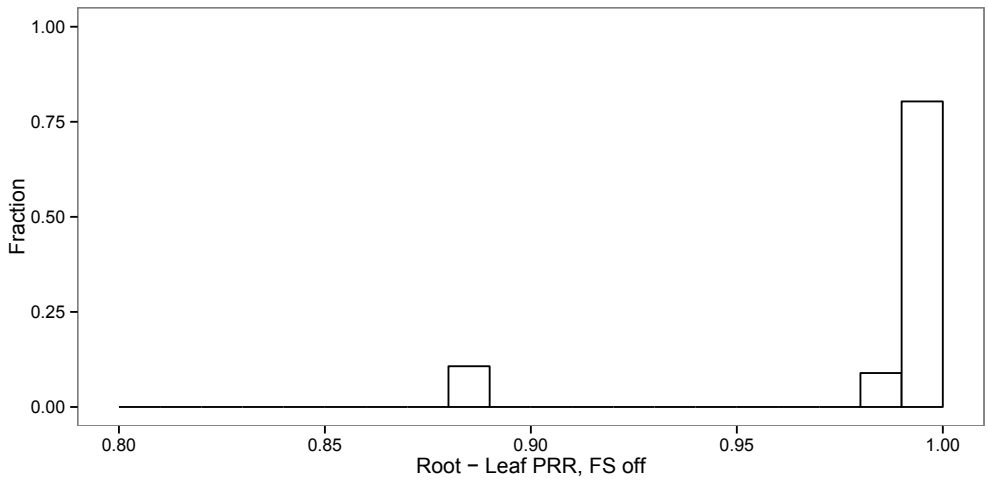
Figure 5.7 shows a CDF of the resulting duty cycle from this test. The duty cycle at each node was calculated as their active time during a download (including wakeup) divided by 50 minutes. This allowed us to run tests back to back. The average duty cycle is 5.2% without forwarder selection, and 2.8% with forwarder selection.

This is higher than the previously reported results. There are three factors that account for this.

First, since every slot now requires both an explicit `SA` from the root and response from



(a)



(b)

Figure 5.5: Packet Reception Ratio with forwarder selection disabled in a flat network. Average root-to-leaf PRR is 98.7% (dragged down by the single sub-99% node) and the average leaf-to-root PRR is 99.75%.

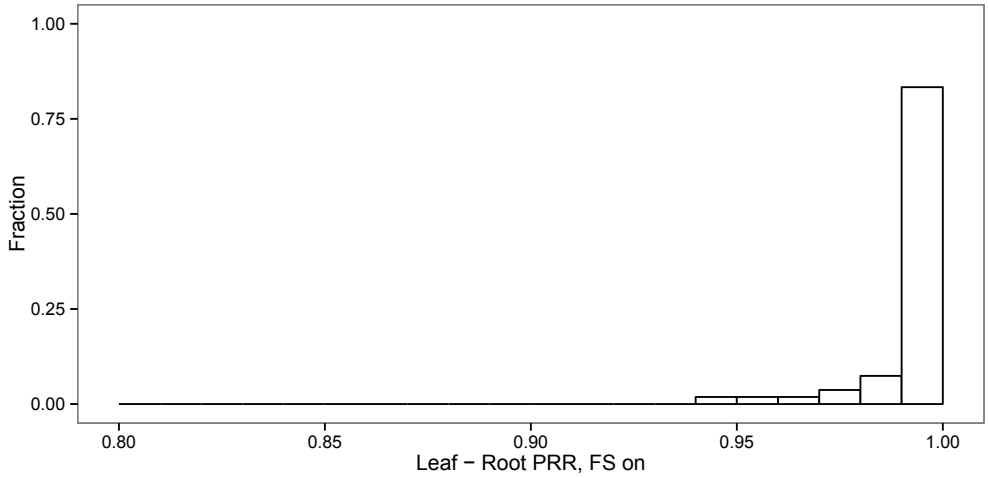


Figure 5.6: Packet Reception Ratio with forwarder selection enabled in a flat network.

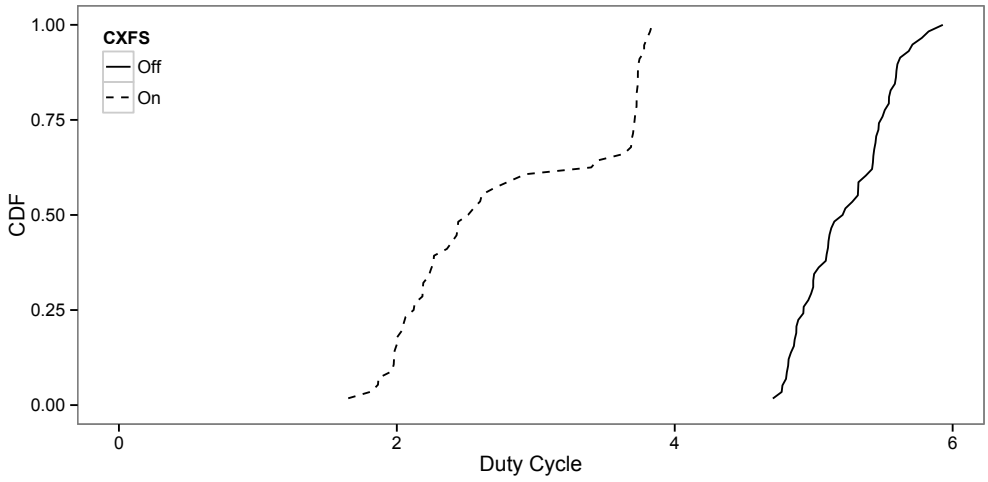


Figure 5.7: Duty cycle with and without forwarder selection enabled in a flat network at 1-minute IPI and 50-minute download interval.

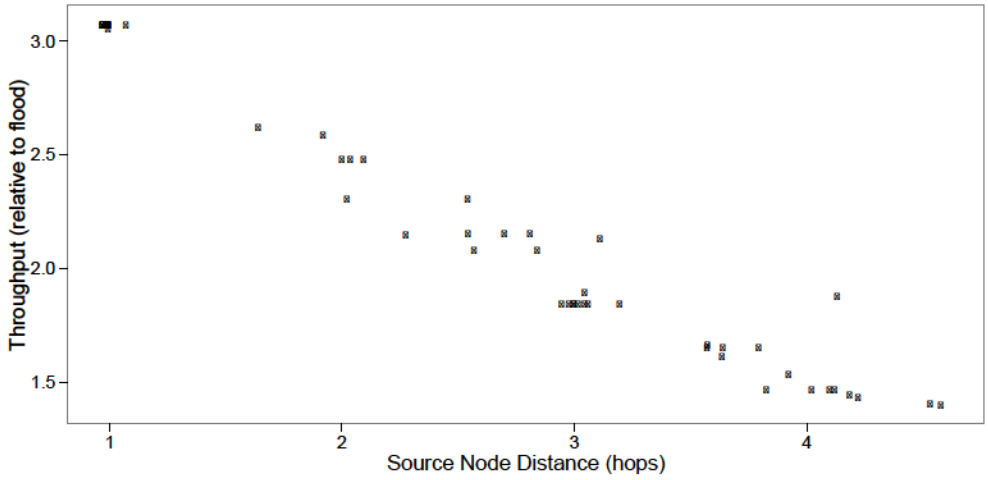


Figure 5.8: Throughput improvement with forwarder selection vs. distance in a non-tiered network. A y value of 1.0 corresponds to equal throughput with and without forwarder selection.

the slot owner, this figure includes control overhead that was not measured in the previous work.

Second, in order to enhance the reliability of the short `SA` packet floods, nodes will broadcast short packets up to 4 times each. This makes the `SA` transmissions even more expensive.

Finally, nodes leave their radio in FSTXON between retransmissions. This simplifies the logic at the link layer, but the FSTXON state consumes about half the power that the active (RX and TX) states consume.

Enabling forwarder selection cuts the average duty cycle to 53.8% of the flooding duty cycle, which is comparable to the results shown in Figure 4.17.

The revised CX implementation has a higher duty cycle than the original implementation, but that cost covers the practical factors of network coordination and slot assignment that the previous results do not account for.

Baseline Throughput and Throughput Improvement

The same tests were used to measure the maximum achievable network throughput and to quantify the throughput increase possible with the forwarder selection. Figure 5.8 shows the throughput with forwarder selection enabled, normalized to the flooding throughput and plotted against distance from the root.

These results show a more significant throughput gain than in Chapter 4, with a mean improvement of 227% (c.f. 49%). This is due to the fact that the explicit slot assignment from the root and response from the leaf take place whether forwarder selection is enabled or not: previously, the setup phase was skipped for flood bursts and this improved non-forwarder selection throughput (hurting the relative improvement).

In absolute terms, nodes could send 2.1 data packets per second without forwarder selection and 2.9 to 6.3 (average: 4.7) data packets per second with forwarder selection. These tests were conducted with 12 B payloads to match the previous test, giving 25.2 B/s and 56.4 B/s, respectively. We note that the software limitations which previously restricted packet size have been resolved since the work in Chapter 4 and PRR is not appreciably impacted by longer packets. Since the frame length is fixed for all packets regardless of the data length, we can guarantee at least 210 B/s in this network (and up to 630 B/s) with 100-byte packets. This far exceeds the typical data rates required in environmental monitoring networks.

We note that since the throughput is always greater than 1.0, this indicates that the maximum network depth (used for flood packet spacing) is overestimated. The maximum average depth was only 4.6. If we had set the flood packet spacing to 5 hops, the improvement would be cut in half across the board (mean improvement of 113.5%).

5.3.2 Overhead

The two sources of overhead in this system are the network wakeups and the slot-assignment/slot-discovery messages.

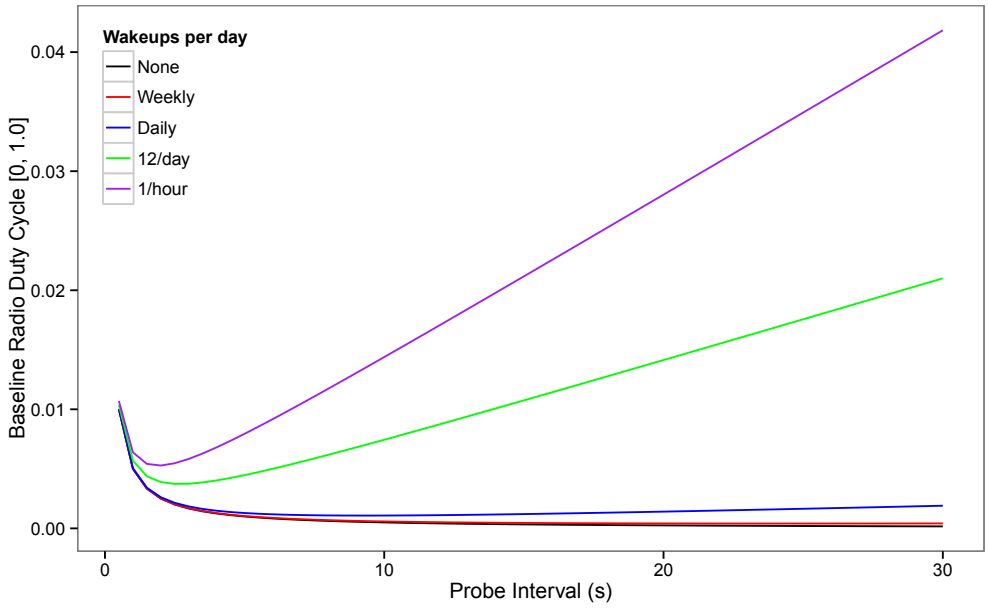


Figure 5.9: Duty cycle as a function of wakeup probe interval and downloads per day. Network depth of 5 is assumed.

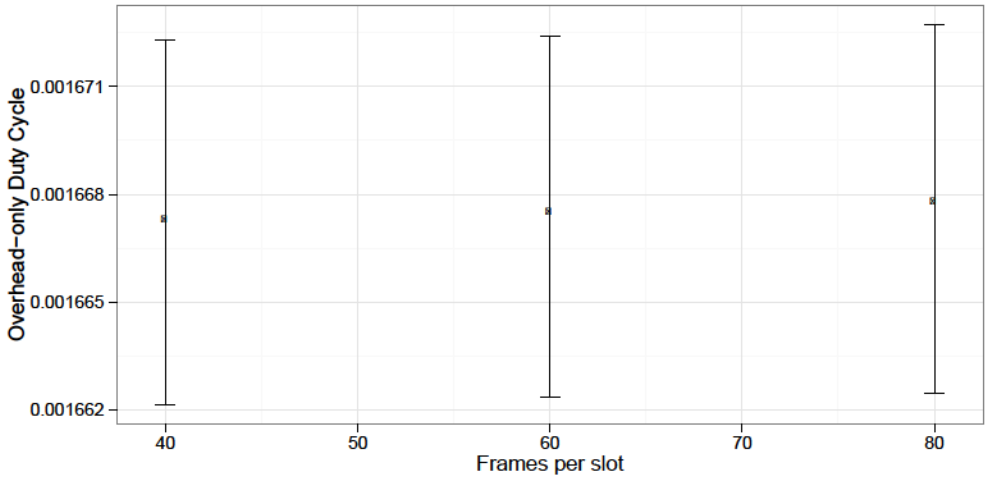
Wakeup cost

Based on the data rate, FEC scheme, and execution timing constraints, we can perform a single wakeup probe + check in roughly 5 milliseconds. Figure 5.9 illustrates how changing the probe transmission interval lowers the idle duty cycle, but increases the time to perform a network wakeup.

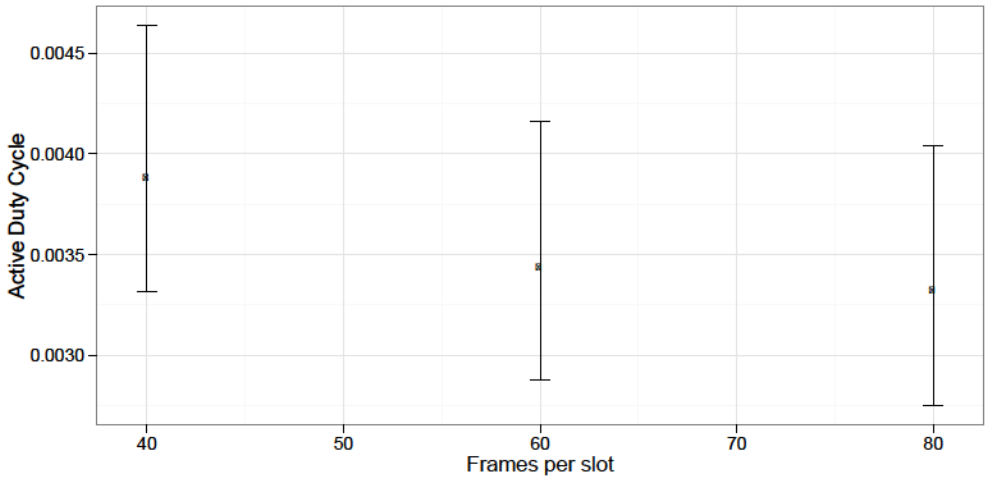
This figure was generated by computing the idle duty cycle for a given probe interval (e.g. 1 probe/second = 5 active ms/second = 0.5% DC) and the total amount of time to guarantee a full-network wakeup in a 5-hop network (e.g. 1/probe/second = 5 second wakeup). Each series represents a different number of wakeups per day.

In our experience, unattended deployments can be run perfectly well with a single download per day. The baseline duty cycle reaches its minimum at 0.11% with a 9.5 second probe interval at this wakeup frequency.

Figure 5.10 shows the impact of slot length on daily duty cycle in a flat network with



(a) No-data (overhead-only) Duty Cycle



(b) Active Duty Cycle

Figure 5.10: Minimum, maximum and median node duty cycle as a function of slot length under 0 B/day and 7500 B/day traffic loads. Assumes daily downloads.

forwarder selection enabled. Figure 5.10(a) shows the effect on a 0 packets-per-download (overhead-only) setup. The duty cycle is driven almost entirely by the wakeup cost at this traffic level, so slot length has little impact.

On the other hand, Figure 5.10(b) demonstrates the energy that can be saved by sizing slots appropriately. If a node can fit all of its packets into fewer slots, the per-slot setup cost will be reduced.

In practice, it can be difficult to determine the best slot length. If we were totally unconstrained, the best slot length would be set to fit exactly as many packets as each node generates between downloads (e.g. 75 packets in this test). Since a node's throughput depends on its distance from the root, this would vary from node to node.

5.3.3 Benefits of segmentation

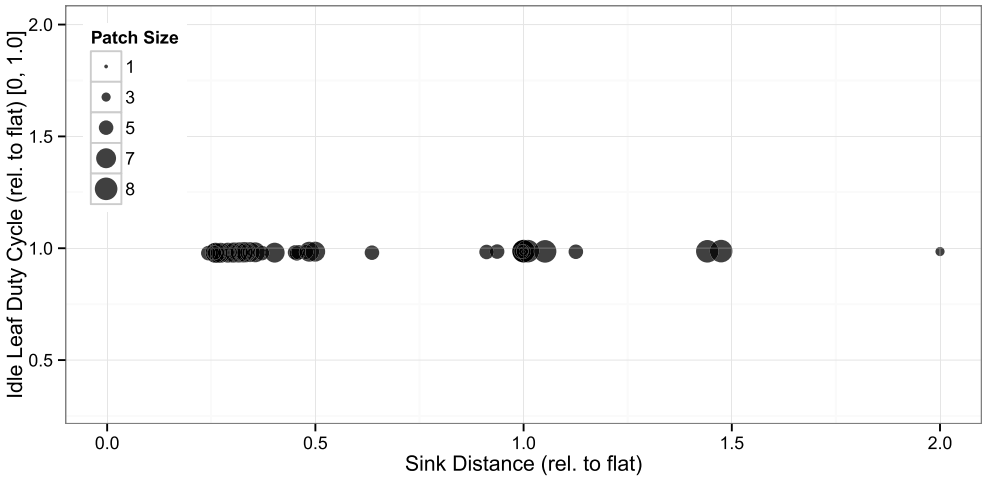
5.3.4 Duty Cycle

Our primary goal in segmenting the networking into patches is to reduce the forwarding load (and radio duty cycle) at each leaf. In order to test the effectiveness of this approach, we divided our testbed into 9 distinct patches, varying in size from a 1 leaf node to 8 leaf nodes.

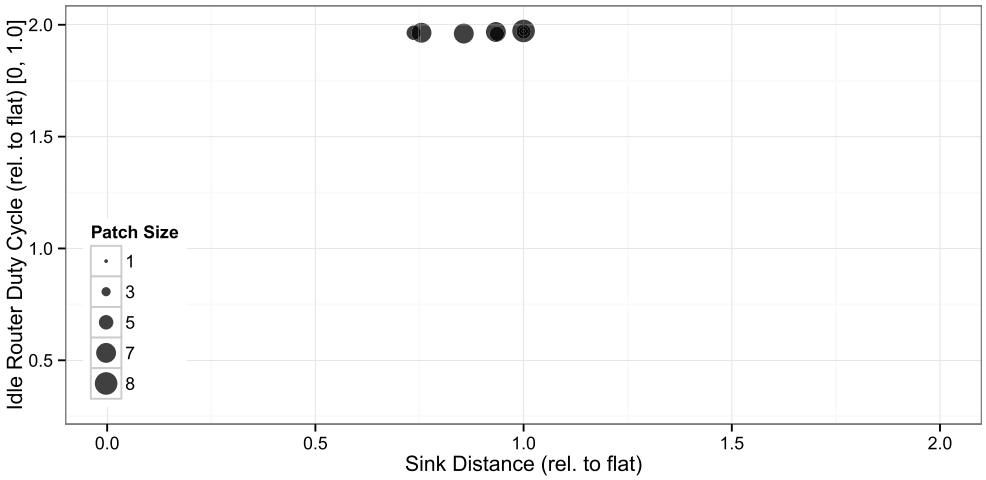
Collections were performed at 0 packets per download and 75 packets per download, with 100-byte payloads. The 0-packet traffic load shows how the overhead costs associated with each download change when the network is segmented, while the 75-packet load is roughly equivalent to a single day's worth of data under commonly-used deployment settings (a 10-minute sampling rate and a single 8-sensor multiplexer board).

The results presented compare the segmented (9-patch) network to the non-segmented (flat) network. Transmission power was set to -6 dBm at all nodes. We note that in practice, router nodes should be expected to have their radio amplifiers activated for router-tier downloads.

Figure 5.11 shows the effect of segmenting the network on the overhead-only traffic load.

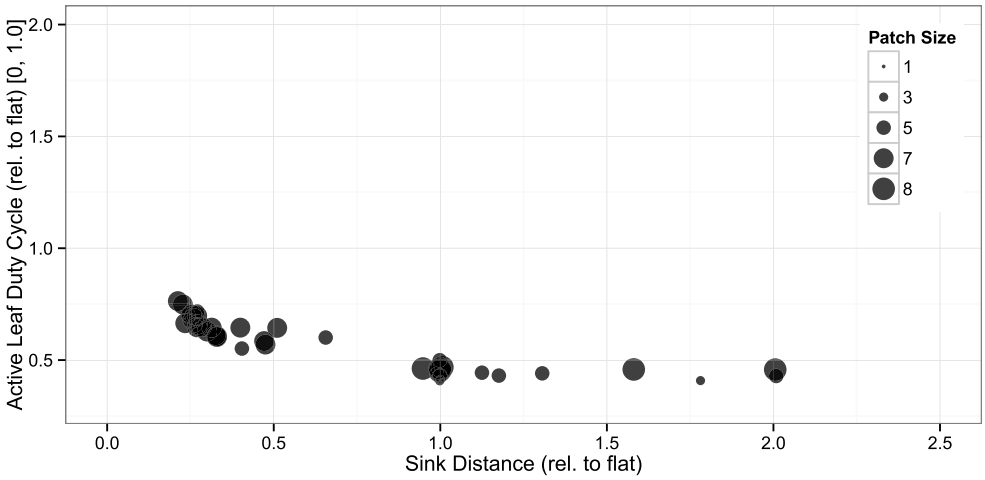


(a) Idle leaf duty cycle change with segmentation.

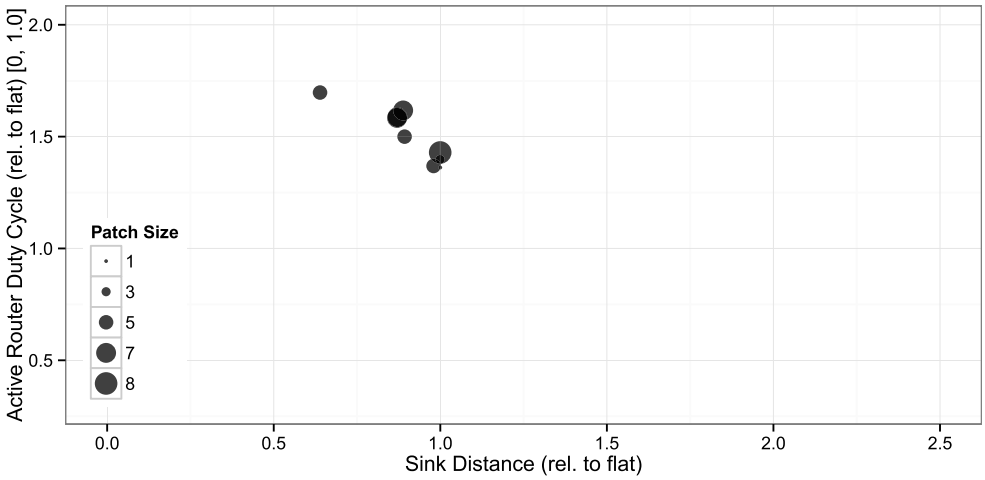


(b) Idle router duty cycle change with segmentation. Note x axis scale difference.

Figure 5.11: Impact of network segmentation on duty cycle with 0 B per node daily data rate (overhead only). The y axis shows the multitier duty cycle as a fraction of the flat network duty cycle (< 1 indicates improvement). The x axis shows multitier distance-to-sink divided by flat network distance-to-sink (< 1 indicates a leaf is closer to its router than to the root).



(a) Active leaf duty cycle change with segmentation.



(b) Active router duty cycle change with segmentation.

Figure 5.12: Impact of network segmentation on duty cycle with 7500 B per node daily data rate. The y axis shows the multitier duty cycle as a fraction of the flat network duty cycle (< 1 indicates improvement). The x axis shows multitier distance-to-sink divided by flat network distance-to-sink (< 1 indicates a leaf is closer to its router than to the root).

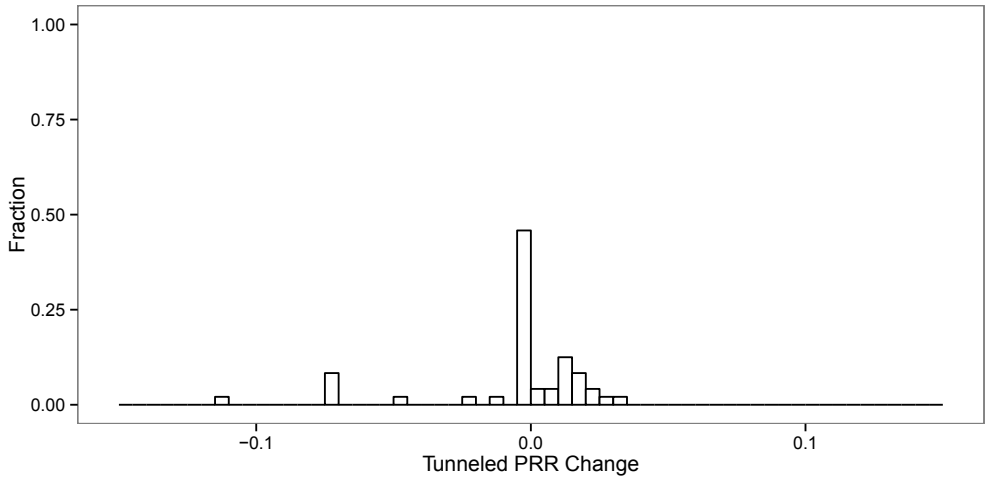


Figure 5.13: Change in PRR from flat network to multi-tiered (leaf to router, router to root). Values greater than 0 indicate an improvement in PRR in the multitier setting.

Since probes and wakeups dominate this cost, leaf nodes see virtually no improvement at this data rate. Router nodes see their duty cycle double, since they have to perform two wakeups per day in the multi-tier network as opposed to one in the single-tier network.

Figure 5.12 shows a rosier picture. The average leaf duty cycle drop is 56% of its single-tier level when they are grouped into patches, shown in figure 5.12(a). We expected smaller patch sizes and shorter sink distances to correlate with the best duty cycle improvements. These results don't seem to indicate an obvious relationship between these factors, though.

Figure 5.12(b) shows that the improvement in leaf node duty cycle is not free. While routers enjoy the same short downloads that the leaves in their patch do, they also have to retransmit all of this data to the root and suffer an additional wakeup. Their duty cycle is 50% higher than it is in a single-tier network, on average. That being said, the total energy consumption of the network decreases under segmentation, consuming only 70.8% as much as the flat network.

5.3.5 Packet Reception Rate

The end-to-end packet reception ratio for leaf nodes may change when the network is segmented. Figure 5.13 shows the distribution of the changes in PRR experienced by the leaf nodes when moving from a flat network to a tiered network. The average packet reception ratio drops to 96.91%, primarily due to the poor router-to-root PRR of a single router. 22 out of the 57 leaf nodes see improvements ranging up to 3.5%. This can occur if their router has a good PRR to the root, and their in-patch PRR is better than their flat-network PRR.

5.4 Future Extensions and Optimizations

There are a few modifications and optimizations that immediately spring to mind which could potentially improve energy-efficiency, connectivity, or both. We record them here not as necessary work that is incomplete, but as potential avenues for future research that may be of interest to others.

We currently enforce a black-out period after each download, in which nodes do not send wakeup probes, in order to not disrupt an ongoing download in the event that a node temporarily loses connectivity. We could take this black-out period further and make it close to the "normal" download interval, on the order of hours. This would cut the idle duty cycle due to wakeup probe transmissions/checks from the 0.5% range (at 1 Hz probe interval) to almost nothing. By disabling probing up until the limit of the clock uncertainty, idle duty cycle would drop to that limit. For example, if a node is using a 1 Hz probe interval in a network of depth 5, it spends 432 seconds per day sending probes and 5 seconds per day being woken up: idle duty cycle dominates the cost. If the worst-case clock skew is 5%, it can cut this to 21.6 seconds per day of probes and 5 seconds per day of wakeup time by suppressing its probes for the 95% of the day where no download is expected. This is a change from roughly 0.5% idle duty cycle to 0.025%.

We've observed receiver saturation in certain situations, which contribute to missed wake-

ups and potential packet loss. This can occur, for instance, if many nodes are in a small area (within a meter) and they are set to transmit at a relatively high power (+10 dBm). In this case, when the last node to wake up sends its wakeup probe, the resulting concurrent transmission exceeds the limit of the radio module's gain control and no packet is detected. Experiments indicate that when the receiver is saturated, it does at least register channel activity (though it fails to recognize the packet preamble sequence). One could add a qualified "activity but no packet" indication to the wakeup layer's rx response, which would instruct it to continue listening for probes and `SA` packets, but prevent it from rebroadcasting probes until it has confirmed that a wakeup has been initiated. It may also be possible to use the radio module's `CLOSE_IN_RX` setting to insert an additional attenuator block before the RF signal reaches the demodulation hardware, or to temporarily reduce output power at the retransmitter (based on the received signal strength of the original probe, perhaps).

Perhaps the most aggressive way to save energy would be to include data compression at multiple points in the system. Leaf nodes could compress their data prior to transmitting it, and routers could potentially compress the data they aggregate from their network. Unfortunately, the MCU which we are using only has 32 KB of ROM available, and the network stack and application logic use the vast majority of that. Our results in [10] clearly demonstrate the practical usefulness of even simple compression techniques, and we desperately hope that future platforms using CX have a little more space to work with.

Chapter 6

A Dirt-to-Database System

6.1 The road to field-deployability

Too often in research, we lose sight of our end goals and pursue short-term projects. While it is exciting to develop novel MAC and routing protocols, our goal is to build networked sensing systems that field scientists can deploy to support their research missions.

The previous chapters have described several important elements of this pursuit by looking at how practical concerns such as deployment topology and expected failure modes inform protocol design. This chapter looks at the remaining elements necessary to move data from the dirt to the database.

Our research group's early sensor network deployments [10, 60] revealed a mismatch between the preferred methods of field ecologists and the capabilities of standard WSN hardware.

Before proceeding with a description of the existing platform and our replacement, let us briefly sum up the usage patterns we observed.

It is rarely feasible to uniformly blanket a study area with sensing nodes at a scientifically-useful level of granularity. The favored approach is to identify regions that are qualitatively similar (e.g. old growth vs. secondary growth forest) and densely instrument a subset of these regions to capture the heterogeneity within (either naturally occurring or due to experimental manipulations). This leads to deployments with dense patches of sensors spread

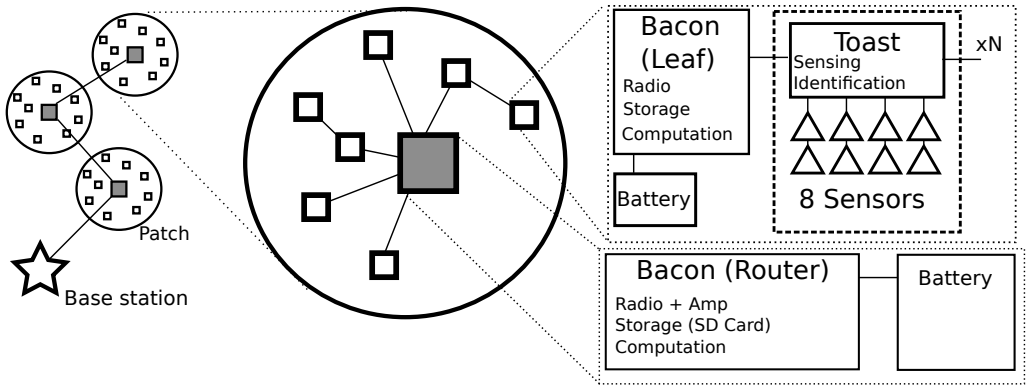


Figure 6.1: Block diagram of a Breakfast deployment.

over kilometers of study area.

We sum up the system requirements as the following:

- We need to place wireless nodes that can accomodate the analog sensors required for the study.
- We must place additional wireless nodes as needed to connect the network.
- We need to record sensor associations and calibration records of deployed hardware.
- Once deployed, we need to get usable data into the hands of domain scientists.

We developed a new suite of hardware and supporting software, code-named *Breakfast*, to address these requirements. Rather than having a single device which does everything in the network (analog sensing, data storage, and routing), we have developed the *Toast* analog sensor board and the *Bacon* wireless sensor mote. The Bacon mote can be configured as a *leaf* (with an antenna printed directly on the circuit board) or as a *router* (with a power amplifier and an external antenna connector). The Toast multiplexer board maintains identification and calibration metadata for its attached sensors and delivers analog measurements to the Bacon mote.

Table 6.1 summarizes the capabilities of the devices, and Figure 6.1 shows how they are deployed and connected. A single Bacon mote connects via commodity USB cables to a digital

(Inter-Integrated Circuit or I^2C) bus of one or more Toast boards. Each Toast board supports up to eight analog inputs.

Separating sensing duties from storage and transmission allows us to reduce cost, decrease software complexity, and automate labor-intensive processes.

By allowing an arbitrary number of analog sensors at each wireless node, we can slash the required number of devices to support a given sensing setup. At USDA, for example, we had to deploy 20 nodes to supply analog sensors for two study areas, despite the fact that each was under 5 meters in diameter. We could instrument this deployment with two Bacon nodes, each connected to a chain of 5 Toast boards. This would save over 50% in fixed hardware costs and reduce the number of monitored and maintained devices in the field.

We described a networking approach to patchy networks in Chapter 5. In practical terms, however, router nodes need to have longer communication range than leaf nodes. At our SERC deployment, for instance, one third of our deployed nodes were in place just to keep the two patches connected. Bacon addresses this issue with router nodes, which add up to 16.4 dB of transmit amplification and 8 dB of receiver sensitivity over the basic mote (before adding an external antenna).

Our old platform connected the mote directly to the analog sensors, so all sensor associations had to be manually recorded and entered into the data processing pipeline (there is no way for the mote to tell what, if anything, is connected to its analog inputs). This is especially important for sensors which must be individually calibrated to provide good data. Tracking these associations is annoying for small deployments, but becomes a major bottleneck for large deployments, especially in campaign settings where time at the study site is limited. The Toast board has enough on-board storage to record unique identifiers for each of its analog sensors, and this data can be collected automatically by the Bacon node.

Taken as a whole, these improvements reduce both the cost of deploying a sensor network and the effort required to maintain it.

We start this chapter by presenting the hardware suite that we developed for long-term

Parameter	**TelosB**	**Bacon-Leaf**	**Bacon-Router**	**Toast**
Free Space Radio Range [4]	0.2km	0.7km	24km/3 km *	NA
ROM	48KB	32KB	32KB	16KB
RAM	10KB	4KB	4KB	2KB
Flash	1MB	8MB + SD card	8MB + SD card	256B
Analog Inputs	4 ext, 4 int	2 int	2 int	8 ext
Active Power	6.4mW	5.4mW	5.4mW	5.0mW
Shutdown Power	15.3μW	7.8 μW	8.1μW	2.1 μW
Std TX Power	60 mW	51.3 mW	802.5 mW	NA
Std TX Output	0 dBm	+ 0.3 dBm	+22.6 dBm	NA
Max TX Power	60mW	87.6 mW	1107 mW	NA
Max TX Output	0dBm	+8.7 dBm	+ 25.1 dBm	NA
RX Power	60 mW	50.1 mW	62.4 mW	NA
RX Sensitivity	-90 dBm	-90 dBm	-98 dBm	NA
Unit Cost	$\sim$\$31	\$22	\$34	\$21

Table 6.1: Comparison of standard TelosB-based platform and Breakfast hardware suite. TelosB cost is based on estimated manufacture cost assuming the same ratio of fabrication/assembly/NRE cost to BOM cost as Bacon/Toast (commercial price is $\sim$€85, \$116). *: router-to-router range estimated at 24km, router-to-leaf 3km.

environmental monitoring, detailing the design process and performance characteristics. We then describe the supporting software and systems we have developed for tracking the assembly, calibration, and deployment of sensors, as well as the data processing pipeline. Finally, we close with a discussion of our early deployment experiences.

6.2 The Breakfast Hardware Suite: hardware for data-hungry scientists

Previous deployments were based on the TelosB [50] mote platform with some supporting hardware. Each sensing location was instrumented with a TelosB mote, a custom-designed analog multiplexer board (allowing the connection of up to 4 external analog voltage inputs to the mote's single ADC input), and a custom-built analog sensor assembly (enclosing 4 analog sensors in a waterproof housing).

There are a few important areas where this platform fails to align with our needs.

First, the limited communication range of the TelosB mote is typically much less than 200

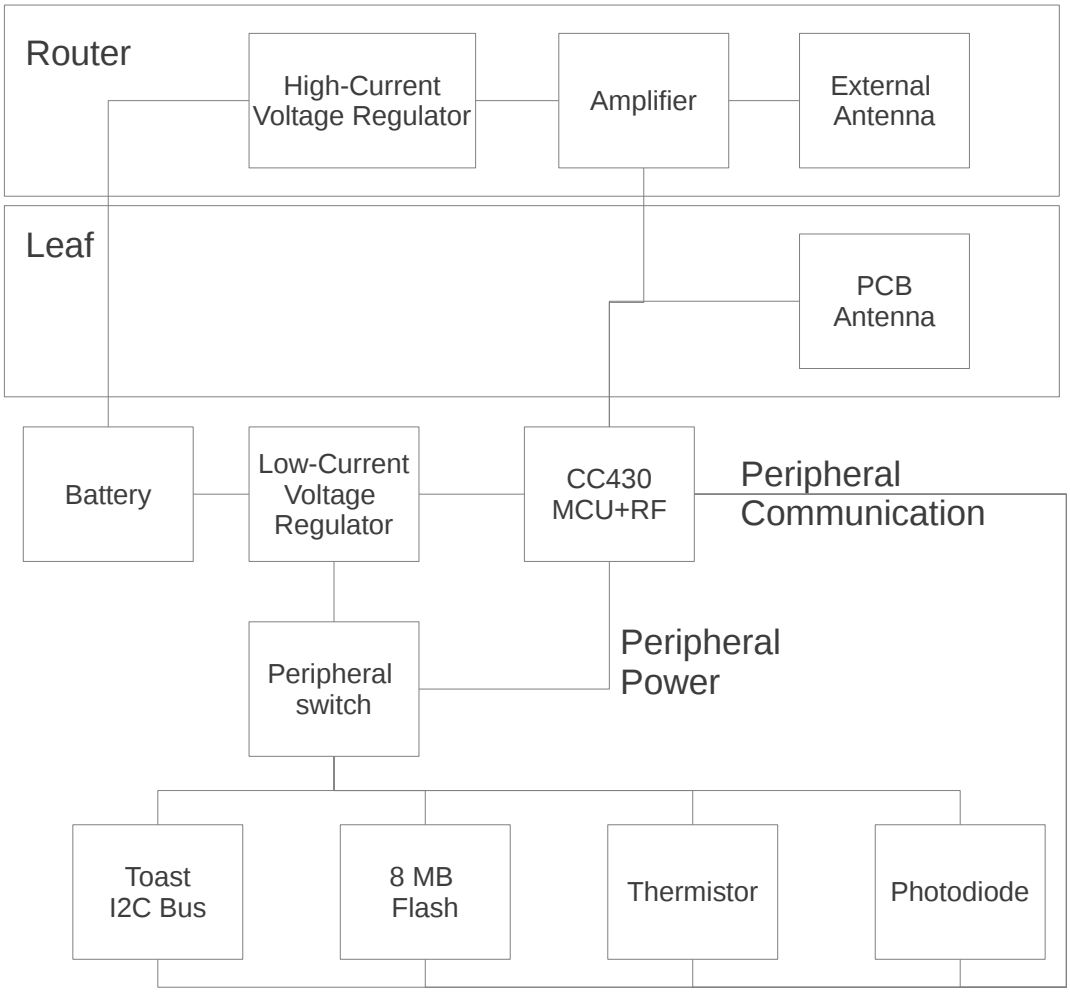


Figure 6.2: Bacon sensor node block diagram. See Appendix A for details.

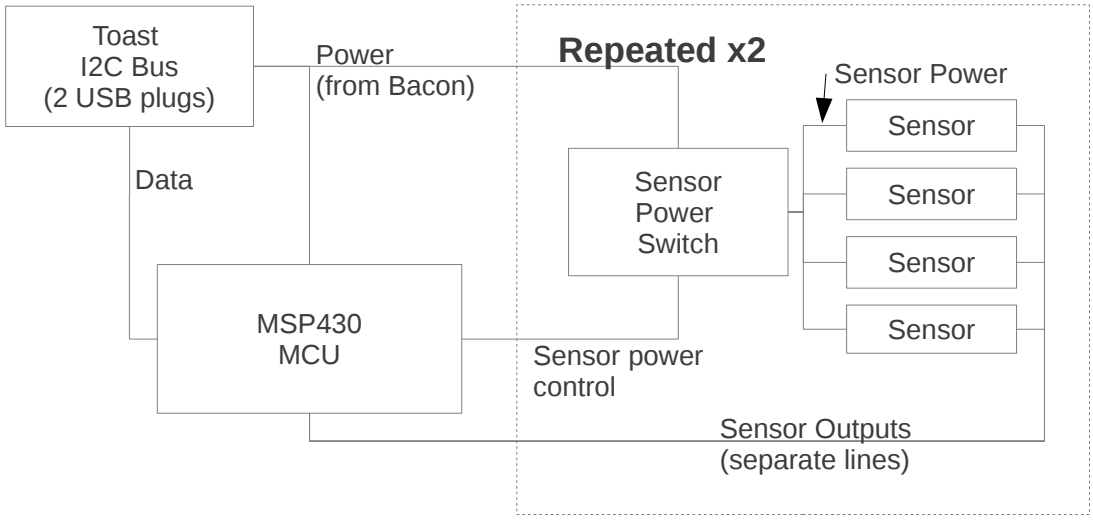


Figure 6.3: Toast Sensor Multiplexer block diagram. See Appendix A for details.

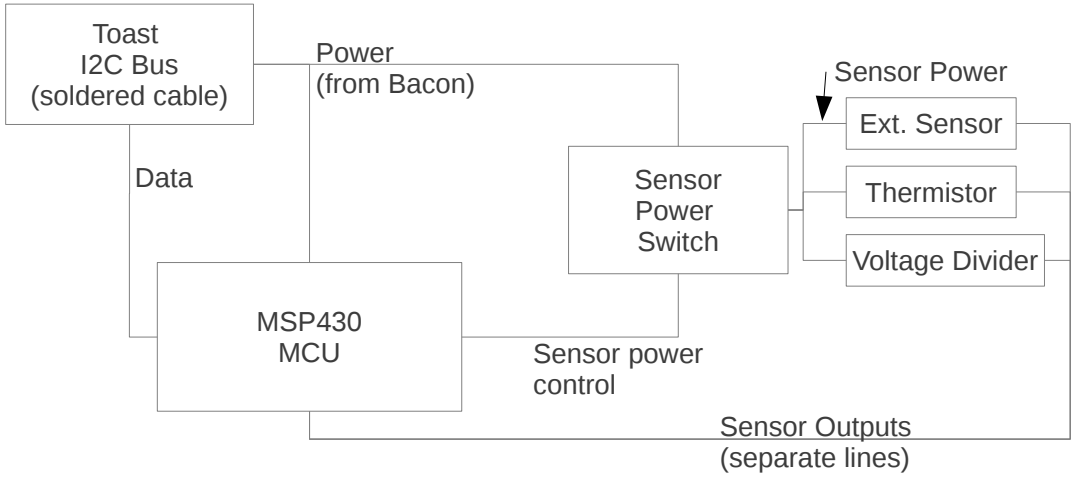


Figure 6.4: Mini-Toast block diagram.

meters in a forested area. This necessitates long chains of relay nodes to connect patches together. This increases the number of points of failure in the network and increases the number of locations that must be visited during maintenance.

Second, the four-sensor-per-node limit artificially inflates the number of nodes we must deploy. Ideally, the factor which dictates node placement is communication range: you add more nodes to the system when you need to extend the connectivity. However, we found ourselves deploying columns of sensors within centimeters of each other, inflating the setup cost and maintenance effort at the site.

Next, we found the bookkeeping required for sensor testing, calibration, deployment, and replacement to be a constant source of headaches and errors. This stemmed from the fact that the device collecting data had no way of telling how many or which sensors were connected to it: all it sees are analog voltages.

Finally, this platform is expensive. TelosB-compatible motes currently retail for €85 [2]. Adding an external antenna, waterproof housing, and the supporting hardware more than doubled the cost for each sensing node. This further aggravates the problems of heavy relay placement and requiring one node for every four analog sensors.

Our general approach was to design hardware that we could match to the deployment scales in common use. We want to be able to have a high ratio of analog sensors to communication/storage nodes, and we want to have a high ratio of sensing nodes to communication-only nodes. Additionally, we want to drive down the unit cost and energy consumption of the individual devices to maximize the number that we can deploy while reducing maintenance efforts.

The high-level design, component-selection, and schematic review were carried out at JHU, while the layouts and schematics were developed by Pacific Design and Engineering.

6.2.1 The Bacon mote

In order to address the communication needs of patchy deployments, we replace the TelosB mote with the *Bacon*[1] mote. Bacon replaces the separate MCU and radio transceiver of the TelosB mote with a single Texas Instruments CC430 system-on-a-chip (SoC) [65], increases the capacity of the flash storage chip, and takes pains to reduce idle power consumption and manufacturing cost.

The CC430 operates on the 900 MHz frequency band, which has a theoretically greater communication range in outdoor environments than the 2.4GHz CC2420 chip [61] on the TelosB, largely due to its worse absorption by water. To further improve flexibility, the Bacon mote can be connected to three different radio front-ends. The most basic option uses a PCB antenna. By moving a single capacitor in the circuit, this can be replaced with an external u.fl antenna connector. We also support a "router" node variant that uses the CC1190 power amplifier/low-noise amplifier [67] and a secondary high-current voltage regulator to add up to 16 dBm of amplification and 8 dB of receiver gain to the mote (as well as an external antenna). Table 6.1 summarizes these differences.

By totally powering down the external flash memory, replacing the voltage regulator, and adding switching hardware to cut power to the on-board light and temperature sensors, we were able to cut the shutdown power consumption in half relative to the TelosB and reduce the active power by a full milliwatt (15.6% lower than the TelosB). When compared to the figures reported in the TelosB datasheet [43], the Bacon Leaf mote consumes 14.5% less power in TX mode and 16.5% less power in RX mode. These improvements help to offset the fact that we are operating at a lower radio data rate (125 kbps without forward error correction, 62.5 kbps with).

The CC430 SoC not only cuts the unit cost over a separate microcontroller and transceiver

[1]The name "Bacon" was chosen for a few reasons. First, a platform called Egs [34] was previously developed in our research group, and Bacon goes well with Egs. Second, it's easy to type "Bacon" (and it gets typed a lot when developing software). Third, people tend to smile when you talk about bacon in an academic setting, and smiling is important.

but it also reduces the total chip footprint. We removed the programming and USB communication hardware from the mote to further reduce cost and space, as it is only used to program the mote and not used in the field. This hardware was moved to a separate USB adapter board, of which we only need a few (for assembly/testing). By reducing the chip count, we were able to come up with a design that is slightly smaller than the TelosB, but uses a single-sided 4-layer PCB. This cuts the cost of the raw materials down to roughly 2/3 of the cost of a TelosB mote.

6.2.2 The Toast analog sensor multiplexer

In order to address the problems of low sensor-to-node ratios and manual metadata collection, we developed the *Toast*[2] analog sensor board. This puts a microcontroller between the analog sensors and the main mote. The Toast MCU samples the analog sensors attached to it and communicates the results to the Bacon MCU digitally (using the I^2C standard). These devices are designed to be connected in a bus of arbitrary length, and each supports up to eight analog sensors via a set of terminal blocks. This lets us support dense sensor clusters with fewer multiplexers and far fewer motes than our previous hardware allowed.

The addition of an MCU to the multiplexer also allows us to persistently store information about the attached analog sensors. In Section 6.3 we describe the labeling and assembly process that lets us assign unique type and ID barcodes to each analog sensor, automatically discover and record this information, and combine it with the raw data stream in the database. Together, this limits the manual data collection at deployment time to the geographic locations of devices.

For cases where domain scientists require relatively few sensor locations, Dr. Szalay has designed a reduced-size, reduced-feature *Mini-Toast* board. This replaces the eight terminal blocks with a set of three solder pads (for a single analog sensor) and puts a thermistor and

[2]The name "Toast" was chosen for the same reasons as "Bacon." Plus, Toast on its own is pretty bland, but when you add stuff to it, it's great.

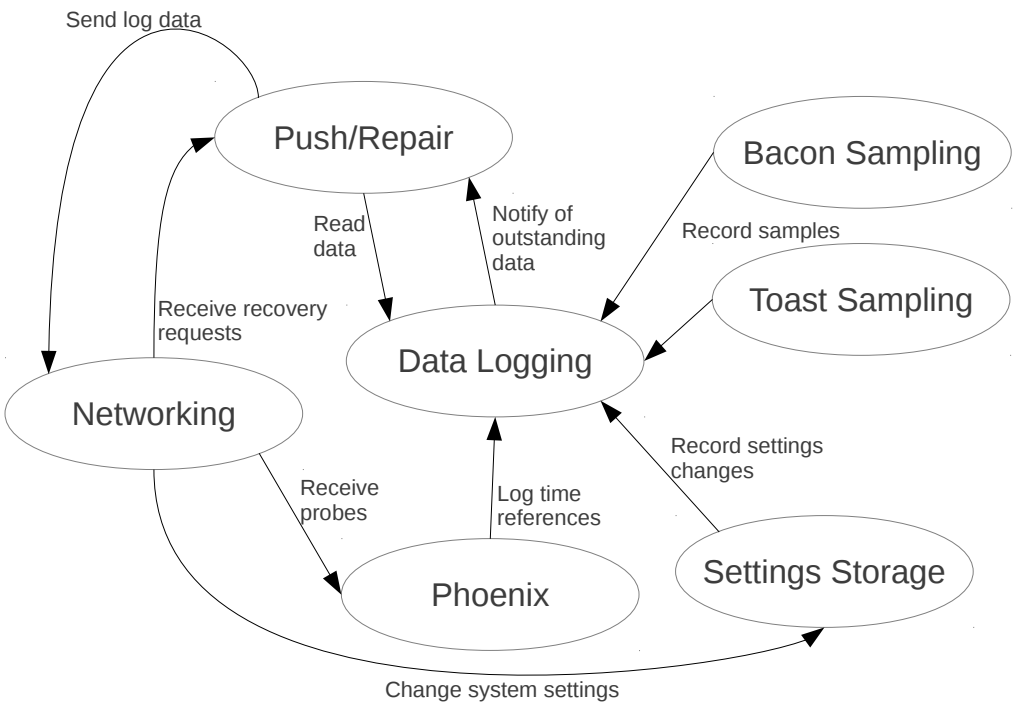


Figure 6.5: Main components of mote software and their interactions.

supply voltage measurement circuit onto the same board. It does not support daisy chaining. This board can be fully potted in epoxy to weatherproof it. The same firmware is used to run this board as the full-size Toast.

6.3 Supporting software

The networking software we've previously described doesn't do us very much good if we don't have data to send over it. In this section, we briefly explain the software that runs in the field, the end-user tools for assembling and configuring equipment, and the data-processing pipeline.

6.3.1 Field-deployed Software

The mote software is designed with modularity in mind. It can roughly be broken down into the configuration storage, data storage, Toast sampling, Bacon sampling, timestamping, and networking subsystems. Figure 6.5 shows a high-level view of how these components interact.

Settings storage

A common settings-storage component manages the configurable elements on each mote, including its unique identifier, sensor sampling rate, and radio channel assignments. This component uses the Tag-Length-Value (TLV) format which Texas Instruments uses to store calibration data and other information in its MCU's flash memories [64], and exposes a normal get/set/delete map interface. Essentially, this structure is a checksum followed by a sequence of (`tag (record identifier)`, `length`, `value`) tuples. A special `TAG_EMPTY` tag indicates an unused section of the storage.

We take a similar implementation approach to that of the core TinyOS MSP430 `InternalFlash` for safe persistence. This splits the available storage space into two units, and ensures that at least one of these units is valid and defines an order between successive modifications. We accomplish this by adding a "version" tag in the TLV structure and using the previously-defined TLV checksum. At boot, we check to see whether segment A or segment B has a valid checksum. If they both do, then we accept the one with the higher version number as the correct TLV. If this is in segment B, then we copy it to segment A and validate that the copy was performed successfully. When there is a run-time modification to the TLV storage, it is fetched from segment A, modified in RAM, checksummed, and written to segment B. The contents are then validated and copied to segment A if all is well.

The same component provides persistence and loading to the Toast and Bacon MCU's, parameterized by the addresses of the two TLV copies and their lengths.

A separate component sits on top of the load/persist methods this provides and implements the logic necessary to `get`, `set`, and `clear` TLV entries.

The final element of this system is a component which connects these methods to the networking subsystem and the data logging subsystem. This not only enables changing system settings remotely (e.g. sampling rate), but it also logs settings changes to the external flash as they occur. These changes get collected along with the sensor data stream, and so they provide an accurate view of each mote's configuration throughout the life of a deployment.

Data Storage

The core TinyOS distribution's `LogStorage` components were modified to address integrity and concurrency problems and simplify the logic surrounding reads from the external flash log.

First, we modified the logic of `Stm25PLogP` to safely support multiple `LogWrite` clients on a single log volume (physical section of the external flash chip's memory). Having a single volume for all data (rather than a separate volume for each data source) simplifies the bookkeeping associated with retrieving data and obviates the need to allocate storage space to different components at design time. This involved maintaining a single shared reference to the end-of-log address and restructuring several safety-checks to take place at the time that a client is *granted* access to the log (rather than at the time that the client *requests* access to the log).

Second, we added a 2-byte checksum to each block header. The log is divided into 4096-byte "blocks," each of which starts with a 4-byte logical block address and a 2-byte checksum of this address. These are used to provide the appearance of a full 32-bit address space for log entries (e.g. to support a log that fills the entire flash and wraps around). The header is used at boot to find the end of a log (by checking for the block header having the highest validated address) and is used when seeking to a point in the log (the physical address is calculated for a given logical address, and the block at that location is checked for a match). We augmented

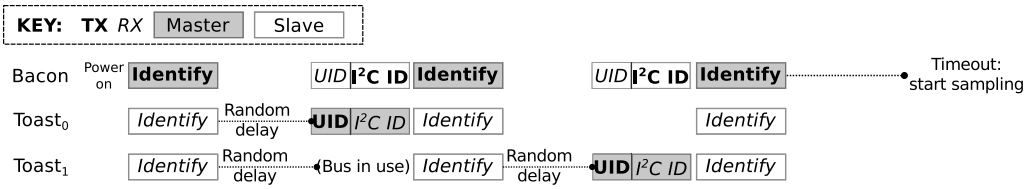


Figure 6.6: Toast discovery process.

the logic in these steps to skip over any blocks that could not be validated. The atomicity of flash writes was preserved from the original TinyOS implementation, and we validated on our testbed that randomly resetting nodes at arbitrary points in this process may lead to data loss or corruption, but does not violate the log structure itself.

Finally, we modified the implementation and semantics of the `LogRead` interface to read records back from the log atomically. This greatly simplifies the logic necessary for identifying missing data (e.g. due to packet drops) and processing data once it leaves the network. For example, say that a node records 30-byte long samples, and sends them out in 100-byte long packets. 10 of these samples can fit into 3 packets, but if the middle packet is lost, we are left with a partial record at the end of the first packet and a partial record at the beginning of the last packet. The next step in data processing, then, has to keep these fragments available and wait until the middle section can be recovered. This task is further complicated when records vary in length (e.g. when different clients log to the same volume). The `read(void* buf, storage_len_t len)` command now reads at most one record from the log and advances the read pointer to the start of the next record. If the supplied buffer is not long enough to store the next record, it signals `readDone` with an `ESIZE` error code. `readDone(void* buf, storage_len_t len, error_t error)` always indicates the number of bytes that were actually read into the supplied buffer.

Toast Identification and Sampling

The fundamental task of this system is to record sensor measurements, and that's handled by the Toast identification and sampling subsystem.

Periodically, the Bacon mote enumerates the attached Toast boards. All communication with the Toast bus is carried out using the I^2C standard. Figure 6.6 shows the sequence of events in the enumeration phase. First, the Bacon powers on the bus. Next, it sends (as master) a command to request identification to a broadcast address and switches into slave mode. The connected Toast boards randomly pick a time to respond to the Bacon mote. They do so by switching into master mode, sending their globally-unique 8-byte ID (stored in their TLV structure), and reading out a bus-local 7-bit I^2C address. The Bacon switches back to master and repeats the request identification step until no Toast boards respond. If the Bacon mote has found any new Toast boards (unique ID was not discovered on the last round), the Bacon mote reads the contents of the Toast TLV storage over the I^2C bus, determines which ADC inputs are in use, and logs the whole structure to its external flash. Likewise, if a previously-detected Toast board is *not* enumerated, the Bacon mote records a disconnection event in its log.

Following the enumeration step, we send a sample-ADC command to the Toast board (indicating the sequence of channels to be sampled and ADC module settings to be used), and then read back the response and log it to flash. Samples are recorded as (Toast unique ID, bacon reboot counter, bacon timestamp, [ADC value]) tuples, where the length varies depending on the number of analog sensors attached to the Toast board. These records can be paired with the previously-recorded TLV structure records to view the series of data from the perspective of individual sensors.

Optionally, we provide a method for recording references between the Bacon's internal clock and the Toast's, which lets us assign timestamps to the individual channel samples with 32KHz resolution. This feature can be disabled to save ROM space (in which case samples are recorded with millisecond accuracy).

Once all Toast boards have been sampled, we power off the bus again.

Bacon sampling

Similarly, the Bacon mote samples its on-board sensors periodically and writes the results to its flash. It switches on each of its on-board sensors in turn, records their voltage, and then switches them back off. These are recorded as (bacon reboot counter, bacon timestamp, battery voltage, photodiode voltage, thermistor voltage).

Phoenix timestamp reconstruction

Our previous work [26] illustrated the importance of recording enough information to support post-mortem timestamping of data. Data is recorded in a mote's local time frame (reboot counter and internal time), but needs to be translated to some global time scale to be scientifically meaningful. We leverage the wakeup probes as presented in "Phoenix: an Epidemic Approach to Timestamp Reconstruction" to accomplish this task: a mote periodically turns its radio on to listen for wakeup probes, logging their local time state and the transmitter's local time state to flash. In post-processing, these sets of references are used to establish mappings from the global time scale to each mote's local time scale. Once again, the interval with which we record these references can be configured through the settings storage component.

Auto-push and data recovery

Finally, we need some method of getting data into the networking stack. This is primarily accomplished through the `AutoPush` component, which keeps track of how much data has been logged to the flash but not yet collected and handles requests for specific data.

Dr. Marcus Chang implemented the core of this logic. The `LogStorage` component provides a `LogNotify` interface to the `AutoPush` component, which notifies it when the number of outstanding records reaches some threshold. At this point, the `AutoPush` component reads records into a packet until it is full (or until there are no records left outstanding) and attempts to send it (either to the most-recently-used parent node or to the broadcast address

if no parent has been found). This packet will be queued as described in 5.2, and then the `LogNotify` interface will be informed of the transfer.

This component also uses the standard `Receive` interface to receive requests for missing data. These consist of `cookie` and `len` parameters, which tell the mote where to start reading data from and how many bytes it should attempt to read. Recovery requests are granted higher priority than data pushes, and only one can be queued at a time.

The packets which we deliver precisely specify what data they contain to facilitate the identification of losses. Each packet consists of a sequence of (`cookie`, `length`, `data`) tuples and a `nextCookie` field. The `nextCookie` field indicates the read pointer value after the last record was read: if the data processing pipeline doesn't have a record that matches this, it indicates a gap.

Packet Tunneling

When Router nodes receive log record packets from their patch, they need to queue them for eventual delivery to the base station. They do this by appending a `TUNNELED_MESSAGE` record to their logs. This record indicates the packet type and the original sender, and contains the payload of the received packet.

Routers use the same auto-push logic as leaf nodes, so the `TUNNELED_MESSAGE` records they record will be subject to the same behavior as described above. The base station can first attempt to recover missing data from the routers, only making requests to the leaf nodes after eliminating all gaps from the router's log.

Implementation notes

In order to fit all of these components into the meager 32K of ROM available on the mote, we had to use our knowledge of what hardware components were and were not shared (and what features are not being used) to aggressively cut code space.

First, we took advantage of the fact that most of the hardware modules on the CC430 were only in use by one component, and so did not need to be protected with resource arbiters. The CC430 has two universal serial communication interface (USCI) modules, so we were able to assign one of them to the SPI bus (flash) and one to the I^2C bus (Toast). Likewise, there is only one user of the RF1A radio module. We implemented an extremely light weight `DummyArbiterC` component which provides the same interfaces and semantics that a real arbiter provides while consuming a fraction of the code space.

Second, we use low-level access to the CC430 ADC module rather than the TinyOS ADC reader abstraction to sample the on-board sensors. In addition to avoiding unnecessary arbiters, this saves a great deal of ROM by ensuring that only the necessary code paths are included. The TinyOS ADC driver uses the values of configuration structs to determine in which mode to operate. This approach resists the static analysis of the nesC compiler, so it includes all possible code paths in the binary.

Third, we allowed for static versions of most of the configurable components. These were mainly used in development, though we use an auto-push component with fixed thresholds for initiating a transfer in the field (as a well-chosen download interval will almost always lead to all nodes requiring a transfer).

Finally, we were able to simplify much of the log storage logic by having a single flash volume. This streamlines the calculation of physical log addresses from logical addresses at many points in the code.

6.3.2 End-user tools

All of the effort which went into developing the hardware and software we've presented so far would be wasted if it could only be used by a computer scientist. To that end, we've invested considerable time into developing tools that allow non-specialists to deploy their own sensor networks.

In order to deploy a WSN, one must configure hardware, record sensor metadata, and periodically download the collected measurements. To this end, we've developed a Labeler utility to barcode and initialize hardware and a deployment Dashboard to interact with the network once it is deployed.

Dr. Marcus Chang developed the Python front ends for these utilities, while I implemented the Labeler mote firmware, the PC-side download logic for the Dashboard, and rehabilitated the defunct Python TinyOS serial stack (used by the Labeler and Dashboard).

Labeler

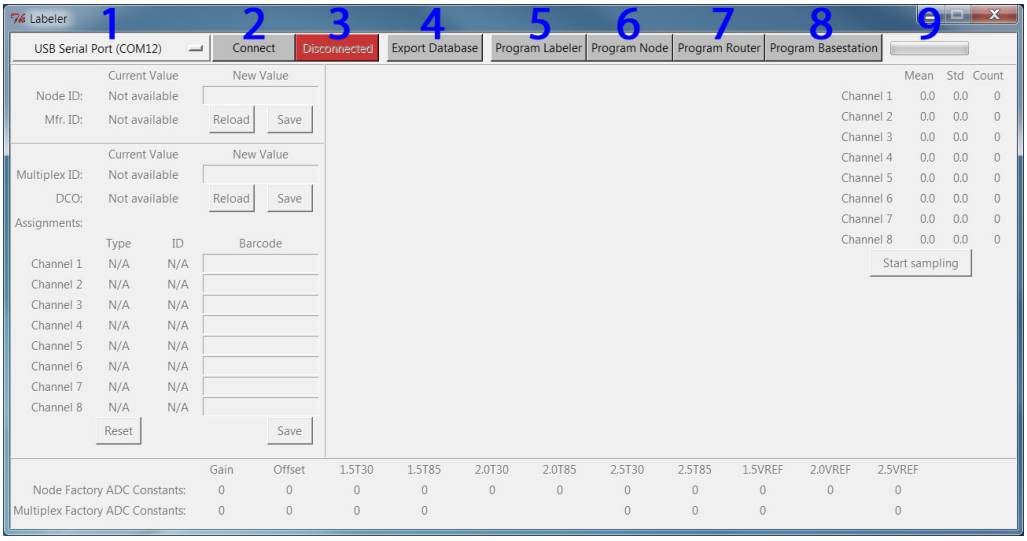
The goals of the Labeler utility are to make it easy for users to assign globally-unique barcodes to all hardware, record associations between analog sensors and Toast boards, test analog sensors, and install Bacon firmware.

Figure 6.7 shows an annotated screenshot from the Labeler utility.

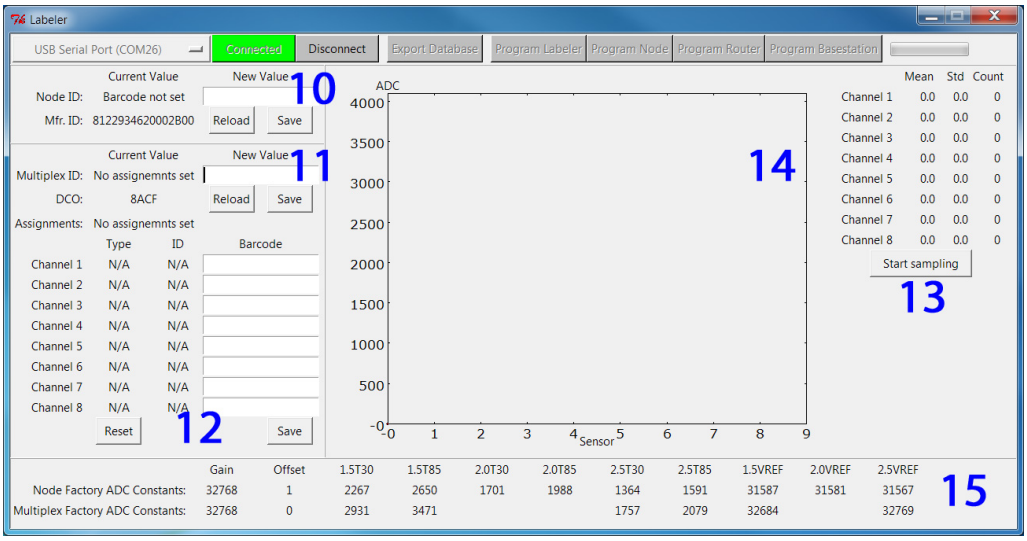
The procedure for assigning barcodes and associating sensors to Toast boards is fairly straightforward. The user connects a Bacon mote (via USB adapter) to the PC, selects the relevant USB device from the dropdown, and programs it as a "Labeler." They then connect a Toast board to the Bacon mote via USB cable. They click "reload" in the Toast panel, then scan the barcodes for each piece of hardware, field by field. Once they've recorded the Toast board's unique ID and the ID's of each connected sensor, they click on "Save" and this data is written to the Toast board's flash. If they wish, at this point they can click the "Start Sampling" button, which will show the user a graphic representation of the voltage on each analog input channel as well as summary statistics for each channel.

Bacon motes are initialized by connecting them to the PC, programming them as "Labeler," and then scanning their barcode. The barcode is persisted to their internal flash. The user can either leave them configured as labelers, or click on the relevant button to program them as sensor nodes, routers, or base stations.

Each of these operations is recorded in a local SQLite database [48]. In general, we expect



(a) (1) selects the physical device, (2) and (3) connect and disconnect from the attached Bacon. (4) Dumps the local record of labeled devices to a .csv file. (5) through (8) install firmware on the attached Bacon. (9) shows the progress of a currently-running task.



(b) (10) and (11) display/edit the Bacon and Toast barcodes. (12) displays/edits the analog sensor types and barcodes. Clicking (13) causes the Toast board to start reading its analog sensors, the results are shown in text above and graphically to the left in (14). (15) Shows the factory calibration settings of the Toast MCU.

Figure 6.7: Labeler utility user interface.

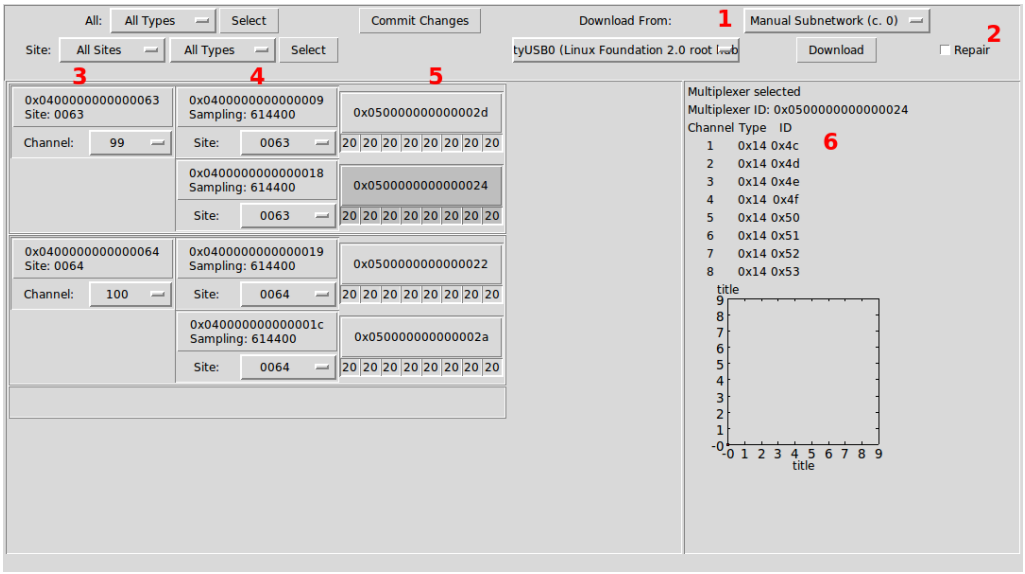


Figure 6.8: Dashboard user interface. (1) Selects the network segment to download from. (2) Indicates whether missing data should be recovered as well as newly-collected data. Column (3) shows the routers detected in the network, and the leaf nodes in each patch are shown in column (4). Column (5) shows the detected Toast boards for each Bacon, and clicking on these entries shows the details of the analog sensors in area (6).

that sensors will be permanently attached to their respective Toast boards. In this case, all the metadata required for sample processing (channel assignments, factory ADC calibration results) can be obtained through the same data stream as the sensor measurements. However, in the event that the history of an analog sensor's associations needs to be tracked, this information can be obtained from the labeler database or dumped to a CSV file.

Dashboard

Many of our deployments have taken place in areas with no permanent power or Internet connectivity. In these deployments, we rely on periodic visits from researchers to download outstanding data from the network. The "Dashboard" application provides users with an easy tool for this task. Figure 6.8 shows an annotated screenshot of the parts of this application.

The main panel shows the network hierarchy as of the last download: the leftmost column shows the discovered routers, the nodes in each patch show up to the right of their router,

and the Toast boards attached to each node show up to the right of their node. Selecting a node shows a detailed list of its attached sensors and its sensor sampling rate.

In order to perform a download, the user selects the network segment which they wish to contact (typically a specific subnetwork for single-patch deployments and the routers for multi-tiered deployments). They then click on the "Download" button. This opens a second window which shows the running status of the download, followed by a summary of the download results. When the download completes, this will inform the user of the last contact time and last sample time from each node in the field, as well as its most recent battery voltage measurement. Checking the "Repair" box will cause the base station to attempt to recover missing data as well as any outstanding data.

Immediately following a download, the collected data is dumped into locally-generated text files.

6.3.3 Data Pipeline

What we have presented so far gets us as far as transferring the contents of the motes' flash logs to the base station mote (potentially tunneled through a router). We need to go through a few more steps before we have data that a domain scientist can actually use.

The general flow of data from the motes to a usable form is depicted in Figures 6.9 and 6.10. At a high level, nodes send the contents of their logs to the base station (potentially passing through a router on the way). These packets are stored in a SQLite database in their raw form, and then broken down into their constituent log records. Each of these log records is then parsed according to its record type, and the results are loaded into the corresponding database tables. SQL and Python scripts are then used to dump these contents into a set of text files which can be used by domain scientists.

By maintaining a local SQLite database (schema shown in Figure 6.11), we gain considerable flexibility in how we can store and operate on the data. We store a canonical record of

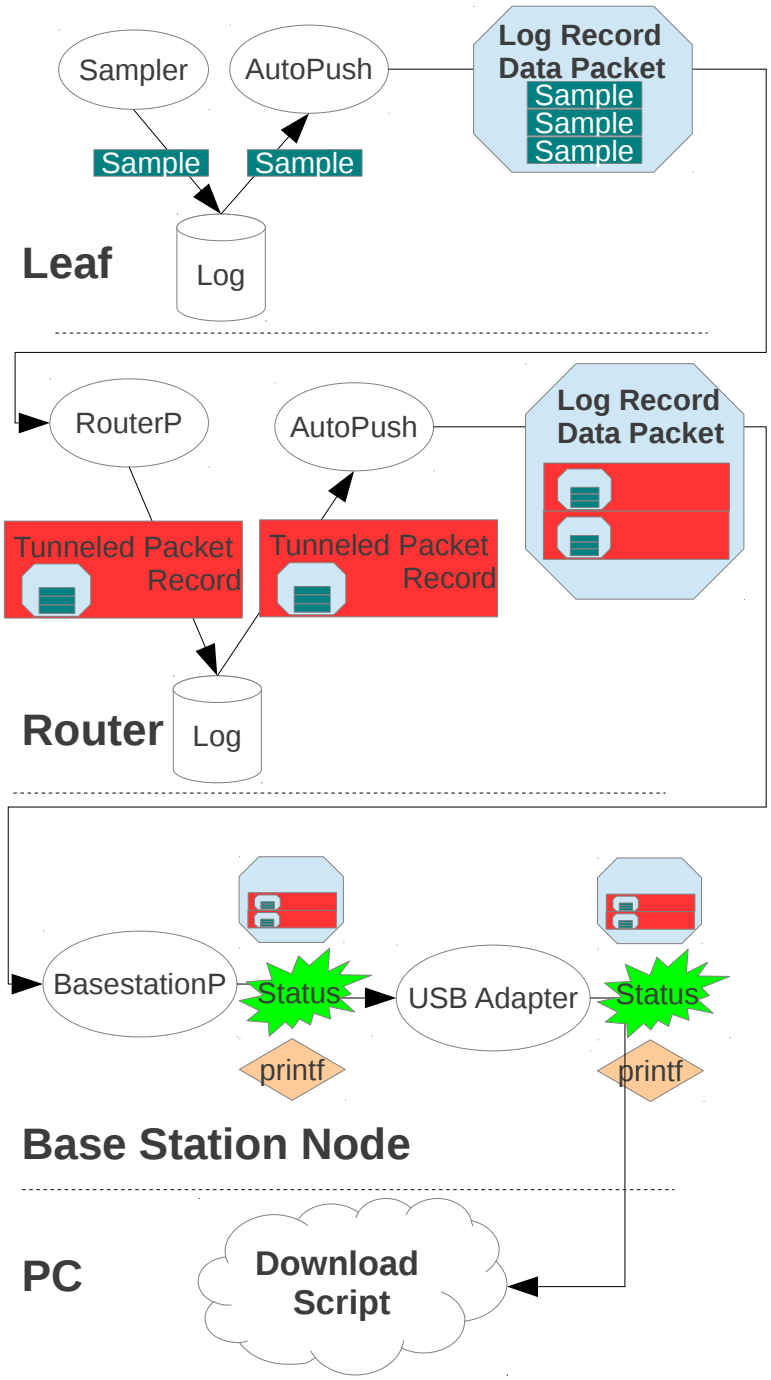


Figure 6.9: Data flow from Leaf node sample to packets received at download script.

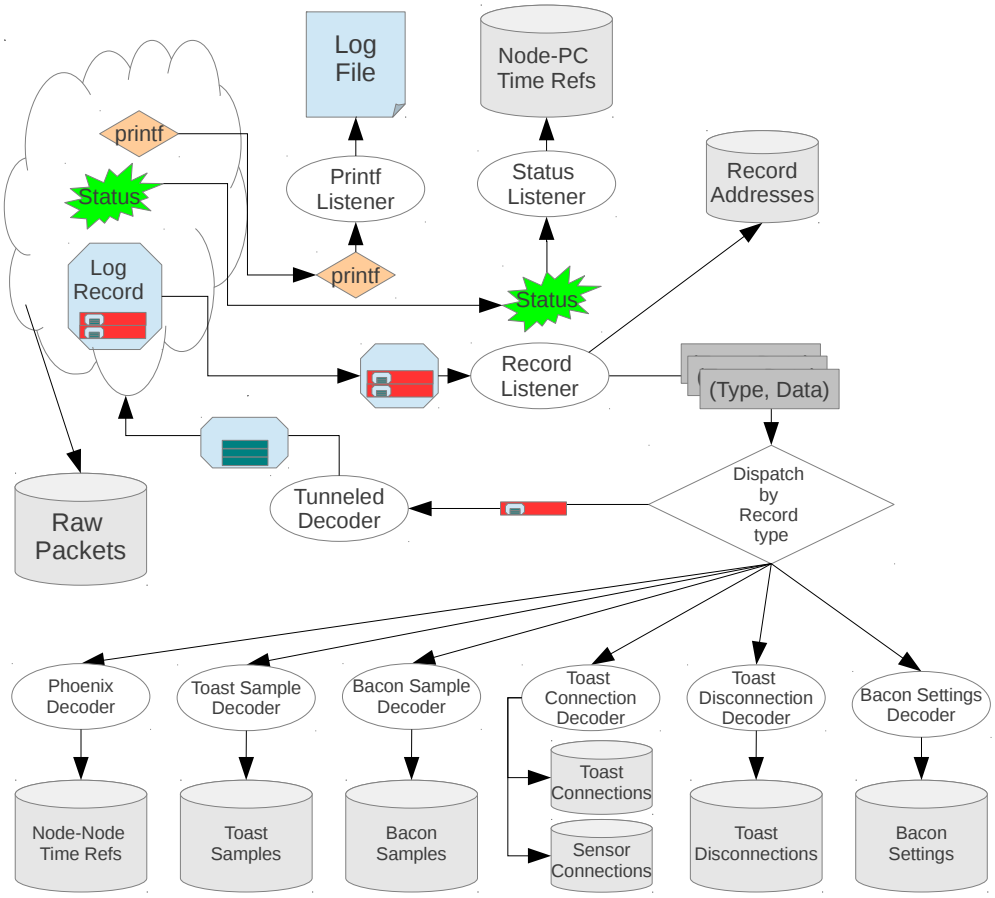


Figure 6.10: **Data flow within download script.**

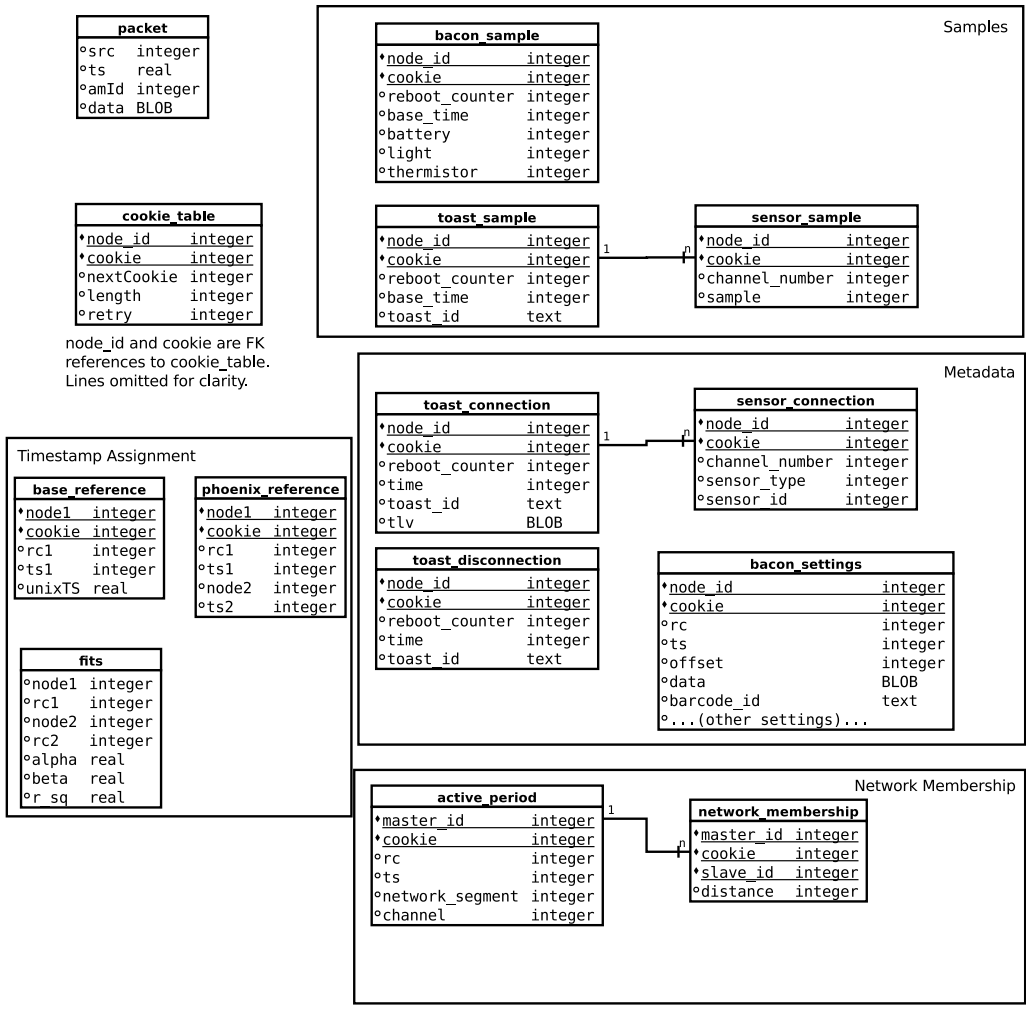


Figure 6.11: Local SQLite DB schema.

the actual raw packets received from the network and a fully-normalized representation of the data they contain. We use the normalized representation to generate whatever output is useful for a given purpose: either a flattened view (appropriate for visual inspection or processing in a tool such as Excel or Matlab [25]), or a translation to some structure suitable for ingest into another database. We track sufficient information to integrate with the existing Life Under Your Feet Staging and Science databases.

The transfer from node flash to the base station mote was described in 6.3.1. As packets are received at the base station mote's radio, they are forwarded over its UART through the

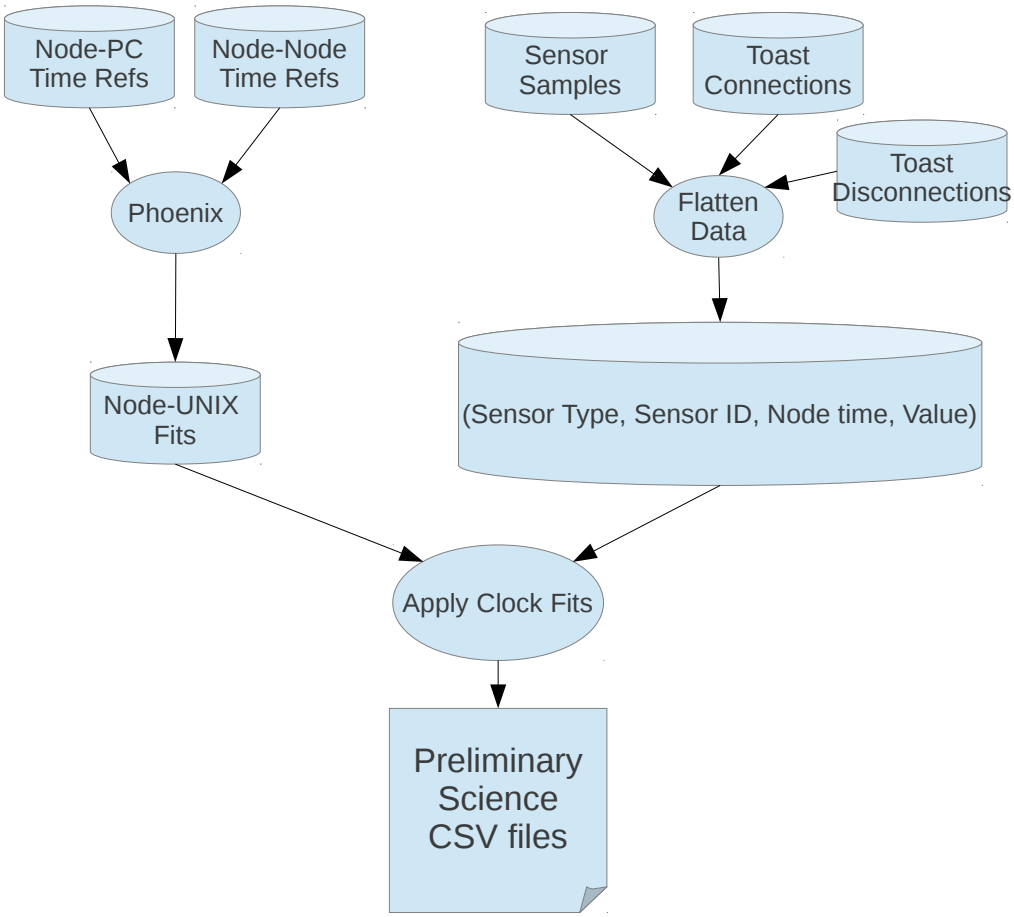


Figure 6.12: CSV file generation flow.

USB adapter and to the download script running behind the Dashboard. Packets received here are written in their entirety to the local database, and then dispatched to one or more "listeners" based on the packet's Active Message (AM) id. We assign listeners to handle log record data packets (`RecordListener`), the `Status` messages received at the master during the download (see 5.1.2), and `printf` messages (used for monitoring download progress).

The `RecordListener` logs the record start addresses, lengths, and succeeding record start addresses for every record in the packet. It iterates over each record contained in the packet, reads its record type (the first byte of the record) and hands the record data off to its corresponding `Decoder` (e.g. `BaconSampleDecoder`, `ToastConnectionDecoder`,

`PhoenixReferenceDecoder`). This framework makes it easy to add new types of log records to the system.

In this design, tunneled data records are received in Log Record data packets from the router. These are handled by the `RecordListener` as above, and the tunneled message decoder simply instantiates a copy of the originally-received packet and re-inserts it into the stream of incoming data packets from the network. We note that this scheme allows for arbitrarily deep network hierarchies, though we only explicitly support a single router tier.

In order to generate human-readable .csv files, we implement an extremely stripped-down version of the Phoenix timestamp reconstruction algorithm [26]. We only use direct global time references (obtained from the `Status` messages) to generate the mapping from local timestamps to global (UNIX) timestamps. For cases where only a single reference is available, we assume that there is zero clock skew.

Sample records are denormalized (joining them to their associated sensor, Toast and Bacon ID's) and these approximate clock fits are applied to convert their local timestamps to global timestamps. After discussion with our collaborators in Earth and Planetary Science, we have decided on a single file for onboard sensor samples (with columns: Barcode ID, date, time, light voltage, temperature voltage, battery voltage) and a separate file for each type of sensor (with columns: sensor barcode ID, toast barcode ID, toast input channel, bacon barcode ID, date, time, voltage).

6.4 Preliminary deployment experiences

In this section, we will relate the qualitative experiences of the deployment maintainers from assembly to field visits. We rely on the more detailed observations from our testbed to characterize the technical performance of our system (radio usage, packet reception rates, etc). Due to code space limitations, we were not able to fit the necessary instrumentation for these measurements and the rest of the required software on the mote. Our goal in this section is to evaluate how well the hardware and supporting systems meet the needs of field

researchers.

These observations are drawn from a conversation with Michael Bernard, an Assistant Research Scholar with the Earth and Planetary Sciences department. Mr. Bernard's duties span the full duration of a sensor deployment, including hardware assembly and testing, determining sensor layouts, and visiting deployments to collect data.

We can be heartened to note that, at a high level, the Breakfast suite of hardware and software is "Certainly better than the previous system, where we called you every time we needed to do something. This at least gives us the chance to be independent."

6.4.1 Preparing for Deployment

With our previous platform, Mr. Bernard's pre-deployment responsibilities involved building sensor assemblies, testing sensors, and sealing node enclosures. Every time that a new batch of hardware was needed, he "needed a refresher" on the test process.

In terms of hardware assembly, Mr. Bernard said that the new hardware is much easier to package. With its PCB antenna, the node and battery can be placed in a plastic tube, with a single small opening at the bottom for connection to a Toast bus. This is "not even close" to the old enclosures, which required drilling holes in a plastic box, connecting waterproof gaskets to them, and applying silicone sealant to each connection.

On the other hand, it is somewhat time-consuming to attach analog sensors to the terminal blocks on the Toast board. Mr. Bernard found it necessary to tin the wires before connecting them, and even then it was sometimes be a struggle to get the wires properly seated.

That being said, Mr. Bernard was able to use the Labeler utility to quickly validate that connections were good and sensor measurements were consistent. He was "pleasantly surprised" by how few bad connections he saw with the terminal blocks. Furthermore, he appreciated being able to see the test results as he assembled hardware: this allowed him to find

and fix problems as he went along. This is in contrast to the previous workflow, where sensor assemblies would be tested in batches after they were put together. Having to switch context from assembly to testing to repair was a cumbersome and awkward workflow.

6.4.2 Experiment Design

One of our main goals with this platform is to match the hardware platform to the deployment needs of environmental scientists. We want to avoid situations where "the tail wags the dog," and researchers have to design their experiments around the limitations of the hardware.

Mr. Bernard's advisor, Dr. Szlavecz, had need of a small-scale experiment at SERC, and he has managed the details of this deployment from assembly to download. The experiment consists of four rainfall shelters, each served by four Bacon nodes and four Toast boards (for a total of 128 instrumented locations). The shelters are grouped in two sites. No routers are deployed, field researchers visit each site separately. This deployment was started on September 21, 2013.

In Mr. Bernard's words,

> The research goals of the experiment are to simulate soil moisture conditions under future changes in rainfall patterns predicted by climate change models and to test how differences in rain volume and intensity might affect carbon storage within forest soils. Using shelters covered with clear polycarbonate panels and rain barrels to collect rain, the researchers are able to distribute water so that one treatment ("dry conditions") receives 0.5 x natural input and the other ("wet conditions") receives at least 1.5 x natural input (additional water amendments are used on occasion). Soil moisture sensors are helpful because they allow us to continuously monitor conditions and correlate them with various environmental measures, such as soil respiration (the rate of carbon dioxide efflux from the soil).

The experiment requires a dense, uniform grid of soil moisture sensors at a single depth.

Mr. Bernard knew, based on the data needs, that he would be populating all eight channels of each Toast board.

Mr. Bernard found that the main limitation in deploying the sensors was the cable length of the individual moisture probes. Since the Toast board is still anchored in one location, it was sometimes difficult to pick locations that let them put all the sensors where they wanted them. That being said, he noted that there's always tradeoff with cable length, where short cables are easier to manage and transport, but long cables give you better placement flexibility.

Mr. Bernard said that cable lengths would be less of a hassle if there was a single analog sensor for each Bacon node. This is a fair point, though we maintain that obtaining additional batteries and assembling additional enclosures might prove to be more trouble than the cables in practice. For experiments where one sensor per Bacon is the best pattern, researchers have the option of partially populating the Toast board or using Mini-Toasts.

6.4.3 Site Visits

Mr. Bernard has been able to visit the site four times, including the initial deployment, between 09/21/13 and 11/1/13.

In terms of hardware, he noted that it is much easier to access everything, and it is possible to do real work in the field. With our previous platform, the only maintenance that was feasible was replacing nodes having failed batteries or leaky enclosures with pre-programmed, pre-assembled nodes. Mr. Bernard was able to use the Labeler utility to perform a firmware update in the field without assistance from us, for instance. He was also able to place nodes on one day, and then return to attach Toast boards on a separate occasion.

In comparison to what he saw at our previous campaign deployment in Ecuador, he had a few comments. He noted that downloads still seem to take a long time. However, he also said that it was not a very big deal to him, since there is normally plenty of other work that

needs to be done, and he can start the download and walk away. While he had not used our previous download utility in Ecuador (a command line tool which a different researcher was responsible for using), he did like that the download process informed him of which nodes it was contacting. He would have liked to see a progress bar or some other indicator of how much time remained in the download. This is not something that we have immediate plans for, but it would be possible to add a "bytes remaining" field to each of the log record data packets to give some more insight into the download progress.

Mr. Bernard appreciated the better communication range afforded by the 900 MHz radios and 10 dBm output power. He described it as "very nice!" and liked being able to stay in the van at the side of the road rather than carrying out a laptop to the site itself. The estimated distance from the road to the site is 50-70 meters.

6.4.4 General Observations

If we are trying to develop something that can be handed off as a package to researchers, they need to be confident that it is working as expected. To this end, I asked Mr. Bernard to describe how confident he was in the system.

Mr. Bernard said that the metadata which he had looked at was all consistent with the ground truth, so he is confident that the sensors are being identified and associated with Bacon motes correctly.

In terms of the sensor data, he said that he "believed that the data was there, but it's not in a format I can read yet. I'm not worried about this step, though." That is to say, the local .csv files indicate that data is being collected as it should be, but the presentation as voltage measurements and lack of spatial information make that dataset incomplete. This is a fair point, and it would be helpful to include data conversion for at least a few common sensor types in the local data-processing steps.

In his ideal system, he would like a better way to record sensor locations. He said that

this process (done with a clipboard and pen) is always painful: you need two people to do it effectively, it takes a while, and "things always go wrong."

Furthermore, he said that when working with the Breakfast suite, he would typically put it down for a week, and then forget about some of the details (what connects where, mainly).

While the Breakfast suite is not perfect, we have made great strides in usability from our previous platform. Improving documentation, labeling, and user feedback would go a long way toward improving the end-user experience.

Chapter 7
Conclusion

Wireless sensor networks have long promised to enable explosive growth across a range of field sciences. In this work, we have explored how the domain-specific requirements of Long-Term Environmental Monitoring shape a unified approach to low power wireless networking.

In this thesis, we designed protocols and hardware with this application in mind. Rather than constraining our imaginations, this application has inspired us.

We didn't rue the need for medium access in dense environments, we delighted in the novelty of the challenge and developed Flip-MAC to excel in these situations. We didn't despair (very much) that unreliable hardware and links lead to routing failures, we built CX, which works without routing. We didn't just acknowledge the need for spatially heterogeneous sensor deployments, we took advantage of it by segmenting our networks and building hardware that concentrates energy costs in this setting. We faced the tedious and error-prone processes in real deployments by automating and simplifying them.

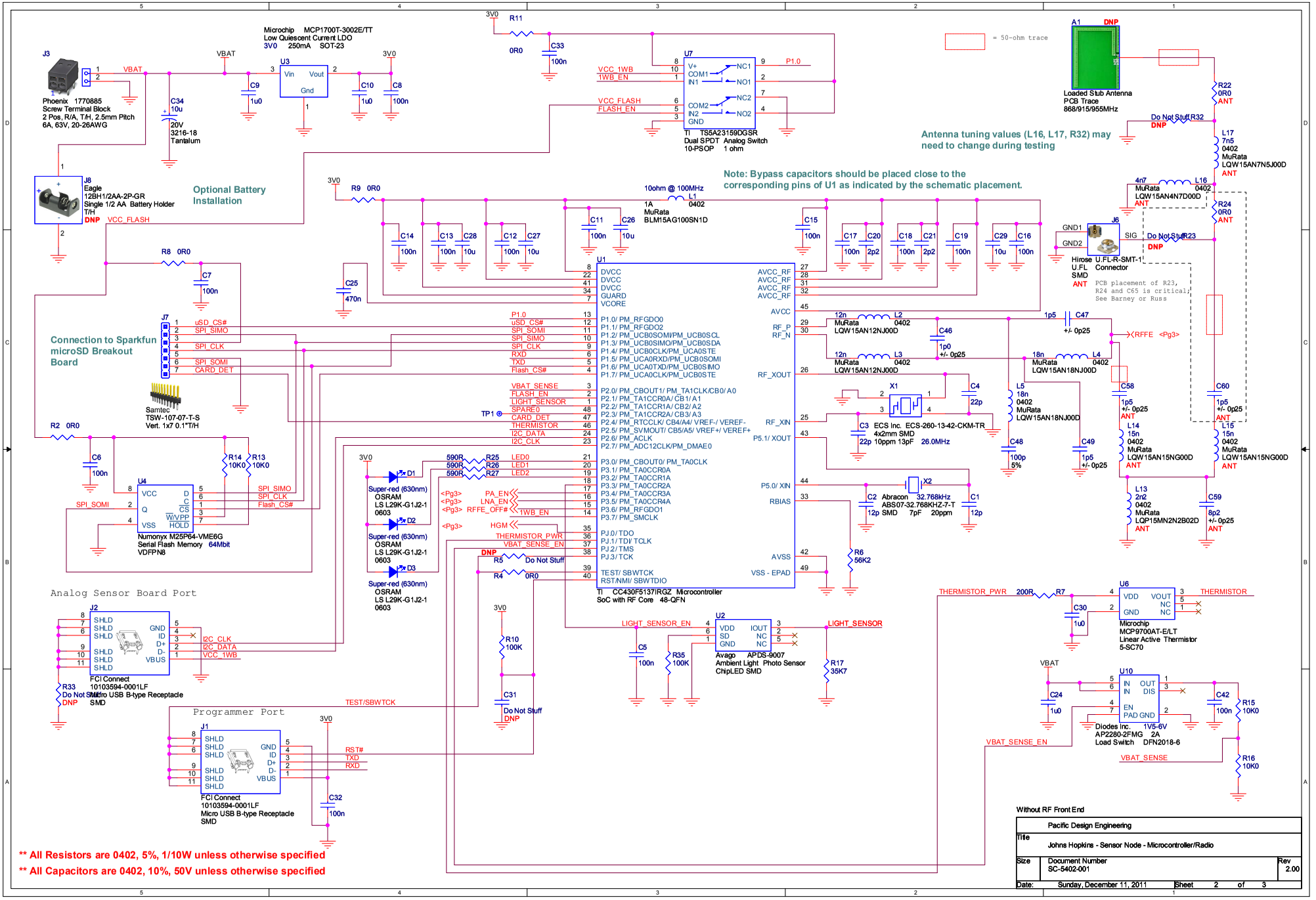
We hope that our system designs can not only enable domain scientists to perform previously-impractical research, but can also help other WSN researchers build more useful tools. We see a bright future in multi-transmitter networking, and hope that interesting research continues to develop in this field.

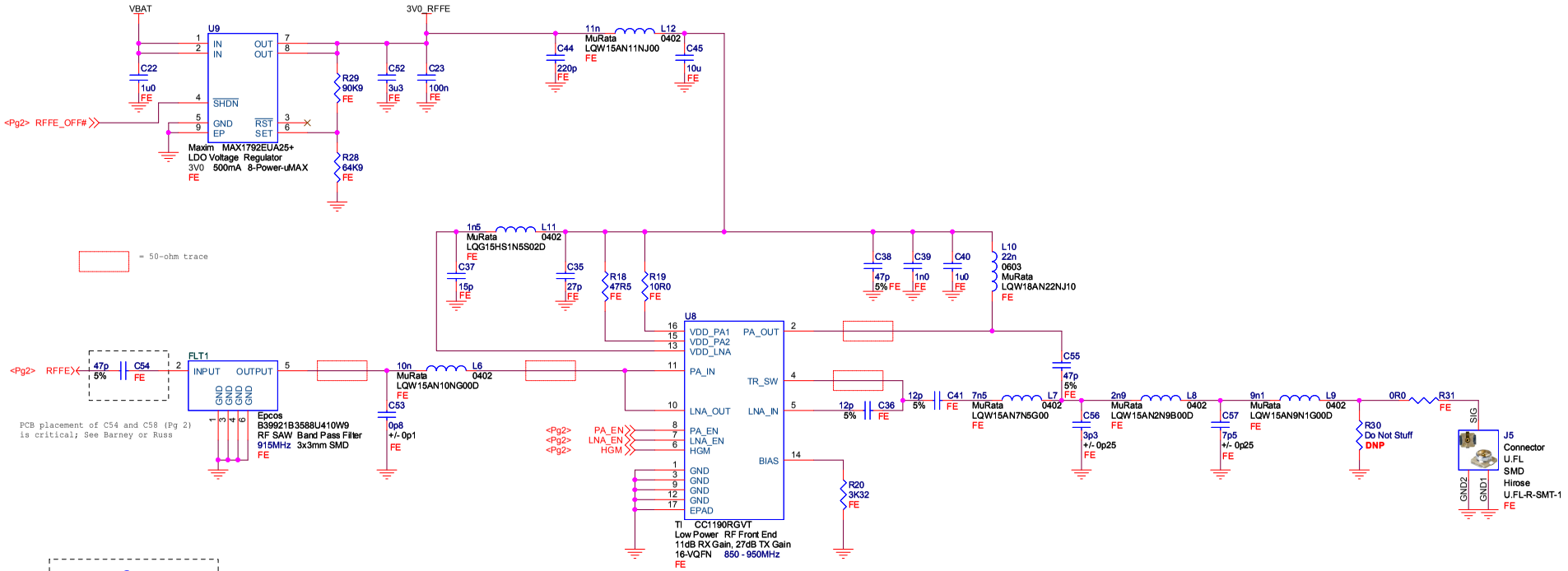
Appendices

Appendix A
Bacon and Toast Schematics

While we came up with the high-level design, selected the components and provided their associated reference designs, the Bacon and Toast schematics were created by Pacific Design and Engineering. Dr. Alex Szalay designed the Mini-Toast schematic.

The Bacon Sensor Node schematic is divided into the base components and the modular RF front-end (for supporting an additional power amplifier and high-current voltage regulator).


Microchip MCP1700T-3002E/TT
Low Quiescent Current LDO
3V0 250mA SOT-23
Phoenix 1770885
Screw Terminal Block
2 Pos, R/A, T/H, 2.5mm Pitch
6A, 63V, 20-26AWG
20V
3216-18
Tantalum
TI TS5A23159DGSR
Dual SPDT Analog Switch
10-PSOP 1 ohm
= 50-ohm trace
Loaded Stub Antenna
PCB Trace
868/915/955MHz
Antenna tuning values (L16, L17, R32) may need to change during testing
Note: Bypass capacitors should be placed close to the corresponding pins of U1 as indicated by the schematic placement.
Optional Battery Installation
Eagle
12BH1/2AA-2P-GR
Single 1/2 AA Battery Holder
T/H
Connection to Sparkfun microSD Breakout Board
Samtec
TSW-107-07-T-S
Vert. 1x7 0.1"T/H
Numonyx M25P64-VME6G
Serial Flash Memory 64Mbit
VDFPN8
TI CC430F5137IRGZ Microcontroller
SoC with RF Core 48-QFN
ECS Inc. ECS-260-13-42-CKM-TR
4x2mm SMD
22p 10ppm 13pF 26.0MHz
Abracon 32.768kHz
ABS07-32.768KHZ-7-T
SMD 7pF 20ppm
Hirose U.FL-R-SMT-1
U.FL Connector
SMD
PCB placement of R23, R24 and C65 is critical, See Barney or Russ
Super-red (630nm)
OSRAM
LS L29K-G1J2-1
0603
Analog Sensor Board Port
FCI Connect
10103594-0001LF
Micro USB B-type Receptacle
SMD
Programmer Port
Avago APDS-9007
Ambient Light Photo Sensor
ChipLED SMD
Microchip
MCP9700AT-E/LT
Linear Active Thermistor
5-SC70
Diodes Inc. 1V5-6V
AP2280-2FMG 2A
Load Switch DFN2018-6
** All Resistors are 0402, 5%, 1/10W unless otherwise specified
** All Capacitors are 0402, 10%, 50V unless otherwise specified
Without RF Front End
Pacific Design Engineering
Johns Hopkins - Sensor Node - Microcontroller/Radio
Document Number SC-5402-001
Rev 2.00
Sunday, December 11, 2011
Sheet 2 of 3



MT1
0.100" Non-Plated Hole

MT2
0.100" Non-Plated Hole

MT3
0.100" Non-Plated Hole

For M2 plastic screws

Component Population Options

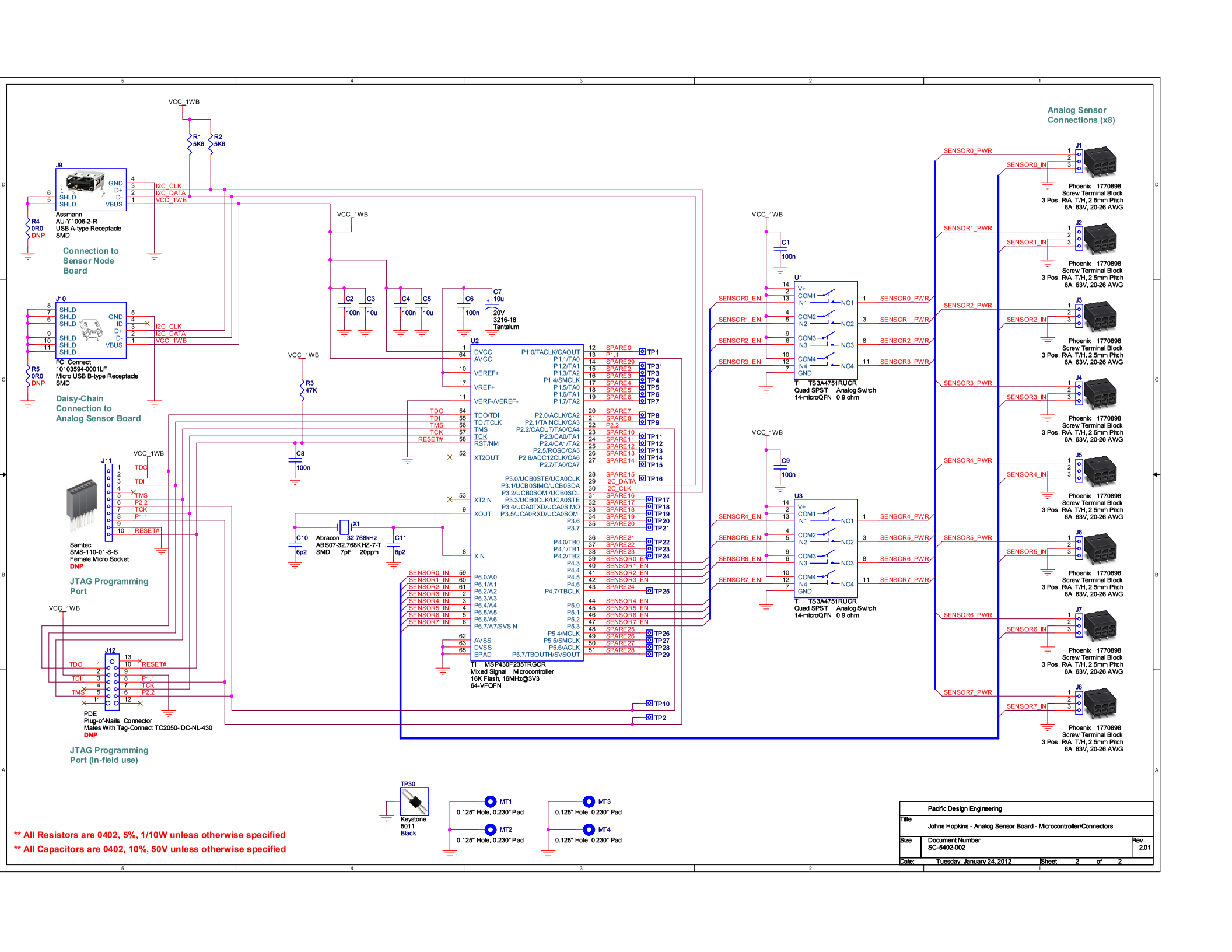
With RF Front-end Pop Option = FE		Without RF Front-end i.e. Using onboard antenna Pop Option = ANT	
Do Not Populate These Parts	Populate These Parts	Do Not Populate These Parts	Populate These Parts
- C58 - L14 - L13 - C59 - L15 - C60 - J6 - R24 - L16 - L17 - R22	All components on Page 3 except R30	All components on Page 3	- C58 - L14 - L13 - C59 - L15 - C60 - J6 - R24 - L16 - L17 - R22

** All Resistors are 0402, 1%, 1/10W unless otherwise specified

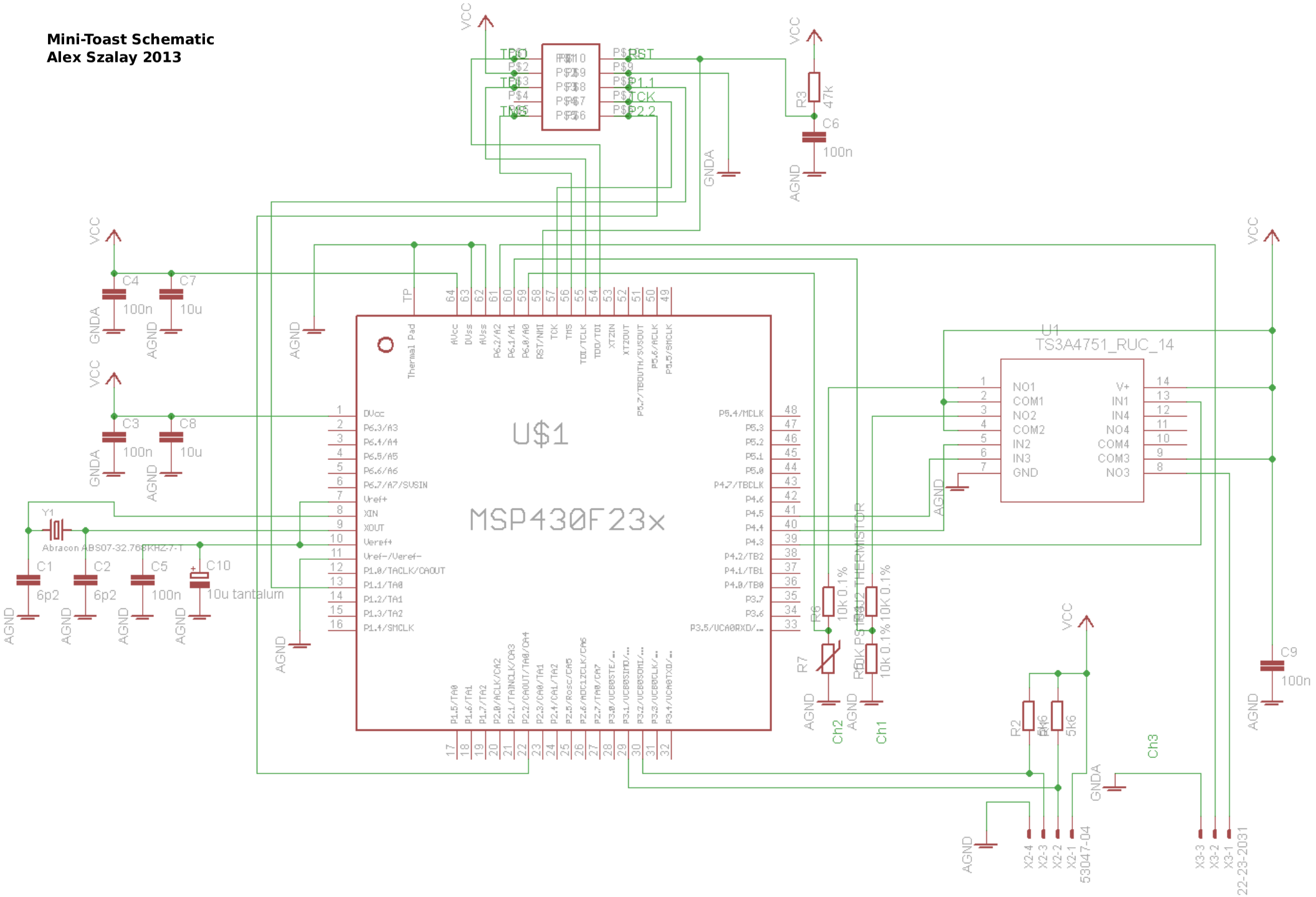
** All Capacitors are 0402, 10%, 50V unless otherwise specified

Without RF Front End

Pacific Design Engineering		
Title: Johns Hopkins - Sensor Node - Radio Front-end		
Size	Document Number SC-5402-001	Rev 2.00
Date: Monday, December 12, 2011	Sheet 3 of 3	


Analog Sensor Connections (x8)
VCC_1WB
R1 5K6
R2 5K6
J9
Assmann AU-Y1006-2-R USB A-type Receptacle SMD
R4 0R0 DNP
Connection to Sensor Node Board
J10
FCI Connect 10103594-0001LF Micro USB B-type Receptacle SMD
R5 0R0 DNP
Daisy-Chain Connection to Analog Sensor Board
I2C_CLK
I2C_DATA
C2 100n
C3 10u
C4 100n
C5 10u
C6 100n
C7 10u 20V 3216-18 Tantalum
R3 47K
C8 100n
U2
TI MSP430F235TRGCR Mixed Signal Microcontroller 16K Flash, 16MHz@3V3 64-VFQFN
J11
Samtec SMS-110-01-S-S Female Micro Socket DNP
JTAG Programming Port
X1
Abracon 32.768kHz ABS07-32.768KHZ-7-T SMD 7pF 20ppm
C10 6p2
C11 6p2
J12
PDE Plug-of-Nails Connector Mates With Tag-Connect TC2050-IDC-NL-430 DNP
JTAG Programming Port (In-field use)
TP1
TP2
TP3
TP4
TP5
TP6
TP7
TP8
TP9
TP10
TP11
TP12
TP13
TP14
TP15
TP16
TP17
TP18
TP19
TP20
TP21
TP22
TP23
TP24
TP25
TP26
TP27
TP28
TP29
TP30
Keystone 5011 Black
MT1 0.125" Hole, 0.230" Pad
MT2 0.125" Hole, 0.230" Pad
MT3 0.125" Hole, 0.230" Pad
MT4 0.125" Hole, 0.230" Pad
C1 100n
C9 100n
U1
U3
TI TS3A4751RUCR Quad SPST Analog Switch 14-microQFN 0.9 ohm
SENSOR0_EN
SENSOR1_EN
SENSOR2_EN
SENSOR3_EN
SENSOR4_EN
SENSOR5_EN
SENSOR6_EN
SENSOR7_EN
SENSOR0_PWR
SENSOR1_PWR
SENSOR2_PWR
SENSOR3_PWR
SENSOR4_PWR
SENSOR5_PWR
SENSOR6_PWR
SENSOR7_PWR
J1
J2
J3
J4
J5
J6
J7
J8
Phoenix 1770898 Screw Terminal Block 3 Pos, R/A, T/H, 2.5mm Pitch 6A, 63V, 20-26 AWG
** All Resistors are 0402, 5%, 1/10W unless otherwise specified
** All Capacitors are 0402, 10%, 50V unless otherwise specified
Pacific Design Engineering
Title
Johns Hopkins - Analog Sensor Board - Microcontroller/Connectors
Size
Document Number SC-5402-002
Rev 2.01
Date: Tuesday, January 24, 2012
Sheet 2 of 2

Mini-Toast Schematic
Alex Szalay 2013



Bibliography

[1] Abracon ABS07 SMD Low Profile Crystal Datasheet. From http://www.abracon.com/Resonators/ABS07.pdf.

[2] Advanticsys Sistemas y Servicios, S.L. From http://www.advanticsys.com/shop/asxm1000-p-24.html.

[3] Hamming(7,4). From http://en.wikipedia.org/wiki/Hamming(7,4).

[4] Afar Communications, inc. Radio (RF) Link Budget Calculator . Available at http://www.afar.net/rf-link-budget-calculator/.

[5] N. Baccour, A. Koubâa, L. Mottola, M.A. Zuniga, H. Youssef, C.A. Boano, and M. Alves. Radio link quality estimation in wireless sensor networks: A survey. *ACM Trans. Sen. Netw.*, 8(4):34:1–34:33, September 2012.

[6] A. Bachir, M. Dohler, T. Watteyne, and K.K. Leung. Mac essentials for wireless sensor networks. *Communications Surveys Tutorials, IEEE*, 12(2):222 –248, 2010.

[7] M. Buettner, G. Yee, E. Anderson, and R. Han. X-MAC: A Short Preamble MAC Protocol for Duty-Cycled Wireless Sensor Networks. In *Proceedings of the 4^{th} ACM SenSys Conference*, 2006.

[8] D. Carlson, M. Chang, A. Terzis, Yin Chen, and O. Gnawali. Forwarder selection in multi-transmitter networks. In *Distributed Computing in Sensor Systems (DCOSS), 2013 IEEE International Conference on*, pages 1–10, 2013.

[9] D. Carlson and A. Terzis. Flip-mac: A density-adaptive contention-reduction protocol for efficient any-to-one communication. In *Distributed Computing in Sensor Systems and Workshops (DCOSS), 2011 International Conference on*, pages 1 –8, June 2011.

[10] Doug Carlson, Jayant Gupchup, Rob Fatland, and Andreas Terzis. K2: A system for campaign deployments of wireless sensor networks. In Pedro Marron, Thiemo Voigt, Peter Corke, and Luca Mottola, editors, *Real-World Wireless Sensor Networks*, volume 6511 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin / Heidelberg, 2010.

[11] Doug Carlson, Jayant Gupchup, Andreas Terzis, and Omprakash Gnawali. Challenges of faults in long-term sensor network deployments. Technical report, Johns Hopkins University, August 2010.

[12] Douglas S. J. De Couto, Daniel Aguayo, John Bicket, and Robert Morris. A High-Throughput Path Metric for Multi-Hop Wireless Routing. In *Proceedings of the 9^{th} ACM International Conference on Mobile Computing and Networking (MobiCom 2003)*, September 2003.

[13] Crossbow Corporation. Stargate Gateway (SPB400). Available at: `http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/Stargate_Datasheet.pdf`, 2004.

[14] M. Demirbas, O. Soysal, and M. Hussain. A singlehop collaborative feedback primitive for wireless sensor networks. In *Proceedings of the 27^{th} IEEE Conference on Computer Communications (INFOCOM)*, 2008.

[15] Adam Dunkels, Bjorn Gronvall, and Thiemo Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Proceedings of the First IEEE Workshop on Embedded Networked Sensors (Emnets-I)*, November 2004.

[16] Simon Duquennoy, Olaf Landsiedel, and Thiemo Voigt. Let the tree bloom: scalable opportunistic routing with orpl. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, SenSys '13, pages 2:1–2:14, New York, NY, USA, 2013. ACM.

[17] Prabal Dutta, Stephen Dawson-Haggerty, Yin Chen, Chieh-Jan Mike Liang, and Andreas Terzis. Design and Evaluation of a Versatile and Efficient Receiver-Initiated Link Layer for Low-Power Wireless. In *Proceedings of the 8^{th} ACM Conference on Embedded Networked Sensor Systems (Sensys)*, pages 1–14, Nov 2010.

[18] Prabal Dutta, Steven Dawson-Haggerty, Yin Chen, Chieh-Jan Liang, and Andreas Terzis. A-MAC: A versatile and efficient receiver-initiated link layer for low-power wireless. *ACM Transactions on Sensor Networks (TOSN)*, 8(4):1–30, September 2012.

[19] Prabal Dutta, Răzvan Musăloiu-E., Ion Stoica, and Andreas Terzis. Wireless ACK collisions not considered harmful. In *Proceedings of the 7^{th} Workshop on Hot Topics in Networks (HotNets-VII)*, pages 19–24, Oct 2008.

[20] Sinem Coleri Ergen and Pravin Varaiya. Tdma scheduling algorithms for wireless sensor networks. *Wireless Networks*, 16(4):985–997, 2010.

[21] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele. Low-power wireless bus. In *10th ACM Conf. on Embedded Networked Sensor Systems (SenSys 12)*, 2012.

[22] Federico Ferrari, Marco Zimmerling, Lothar Thiele, and Olga Saukh. Efficient network flooding and time synchronization with glossy. In *Proceedings of the 10th International Conference on Information Processing in Sensor Networks (IPSN)*, Chicago, IL, USA, 2011. ACM/IEEE.

[23] David Gay, Phil Levis, Rob von Behren, Matt Welsh, Eric Brewer, and David Culler. The nesC Language: A Holistic Approach to Networked Embedded Systems. In *Proceedings of Programming Language Design and Implementation (PLDI) 2003*, June 2003.

[24] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection Tree Protocol. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 1–14, Nov 2009.

[25] MATLAB Users Guide. The mathworks. *Inc., Natick, MA*, 5, 1998.

[26] J. Gupchup, D. Carlson, R. Musaloiu-E., A. Szalay, and A. Terzis. Phoenix: An epidemic approach to time reconstruction. In *Proceedings of the Seventh European Conference on Wireless Sensor Networks (EWSN)*, pages 17–32, February 2010.

[27] Jayant Gupchup. *Data Management In Environmental Monitoring Sensor Networks*. PhD thesis, Johns Hopkins University, 2012.

[28] Yuan He, Lufeng Mo, Jiliang Wang, Wei Dong, Wei Xi, Tao Chen, Xingfa Shen, Yunhao Liu, Jizhong Zhao, Xiangyang Li, and Guojun Dai. Poster: Why Are Long-Term Large-Scale Sensor Networks Difficult? Lessons Learned from GreenOrbs. In *Proceedings of ACM MobiCom*, 2009.

[29] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System Architecture Directions for Networked Sensors. In *Proceedings of ASPLOS 2000*, November 2000.

[30] P. Huang, H. Oki, Y. Wang, M. Martonosi, L. Peh, and D. Rubenstein. Energy-efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet. In *Proceedings of the Tenth International Conferece on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X)*, October 2002.

[31] Jonathan Hui and David Culler. IP is dead, long live IP for wireless sensor networks. In *Proceedings of the 6th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2008.

[32] François Ingelrest, Guillermo Barrenetxea, Gunnar Schaefer, Martin Vetterli, Olivier Couach, and Marc Parlange. Sensorscope: Application-specific sensor network for environmental monitoring. *ACM Trans. Sen. Netw.*, 6(2):17:1–17:32, March 2010.

[33] J. Flint K. Leentvaar. The Capture Effect in FM Receivers. *IEEE Transactions on Communications*, 24(5):531–539, 1976.

[34] JeongGil Ko, Qiang Wang, Thomas Schmid, Wanja Hofer, Prabal Dutta, and Andreas Terzis. Egs: A cortex m3-based mote platform. In *IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communication and Networks (SECON)*, pages 1–3, Jun 2010.

[35] B. Kusy, C. Richter, Wen Hu, M. Afanasyev, R. Jurdak, M. Brunig, D. Abbott, Cong Huynh, and D. Ostry. Radio diversity for reliable communication in wsns. In *Information Processing in Sensor Networks (IPSN), 2011 10th International Conference on*, pages 270 –281, april 2011.

[36] K. Langendoen, A. Baggio, and O. Visser. Murphy loves potatoes: experiences from a pilot sensor network deployment in precision agriculture. In *Proceedings of the Parallel and Distributed Processing Symposium (IPDPS)*, April 2006.

[37] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, et al. Tinyos: An operating system for sensor networks. *Ambient intelligence*, 35, 2005.

[38] Philip Levis and David Gay. *TinyOS programming*. Cambridge University Press, 2009.

[39] Jie Liu, Bodhi Priyantha, Feng Zhao, Chieh-Jan Mike Liang, Qiang Wang, and Sean James. Towards Fine-Grained Data Center Cooling Monitoring using RACNet. In *HotEmNets*, 2008.

[40] Yunhao Liu, Yuan He, Mo Li, Jiliang Wang, Kebin Liu, Lufeng Mo, Wei Dong, Zheng Yang, Min Xi, Jizhong Zhao, and Xiang yang Li. Does wireless sensor network scale? a

measurement study on greenorbs. In *INFOCOM, 2011 Proceedings IEEE*, pages 873–881, 2011.

[41] Jiakang Lu and Kamin Whitehouse. Exploiting the capture effect for low-latency flooding in wireless sensor networks. In *INFOCOM*, 2009.

[42] Scott Moeller, Avinash Sridharan, Bhaskar Krishnamachari, and Omprakash Gnawali. Routing without routes: the backpressure collection protocol. In *IPSN '10: Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 279–290, New York, NY, USA, 2010. ACM.

[43] MoteIV Corporation. Tmote Sky. Available at: `http://www.sentilla.com/moteiv-endoflife.html`.

[44] Răzvan Musăloiu-E., Chieh-Jan Liang, and Andreas Terzis. Koala: Ultra-low power data retrieval in wireless sensor networks. In *Proceedings of the 7^{th} international symposium on information processing in sensor networks (IPSN)*, pages 421–432, April 2008.

[45] Vinod Namboodiri and Abtin Keshavarzian. Alert: An adaptive low-latency event-driven mac protocol for wireless sensor networks. In *Proceedings of the 7th international conference on Information processing in sensor networks*, IPSN '08, pages 159–170, Washington, DC, USA, 2008. IEEE Computer Society.

[46] F. Österlind, L. Mottola, T. Voigt, N. Tsiftes, and A. Dunkels. Strawman: resolving collisions in bursty low-power wireless networks. In *Proceedings of the 11th international conference on Information Processing in Sensor Networks*, pages 161–172. ACM, 2012.

[47] Fredrik Osterlind, Niklas Wirstrom, Niclas Finne Nicolas Tsiftes, Thiemo Voigt, and Adam Dunkels. Strawman: Making sudden traffic surges graceful in low-power wireless networks. In *HotEmNets*, 2010.

[48] Michael Owens. Embedding an sql database with sqlite. *Linux J.*, 2003(110):2–, June 2003.

[49] J. Paek, B. Greenstein, O. Gnawali, K.-Y. Jang, A. Joki, M. Vieira, J. Hicks, D. Estrin, R. Govindan, and E. Kohler. The tenet architecture for tiered sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 6(4), 2010.

[50] Joseph Polastre, Robert Szewczyk, and David Culler. Telos: Enabling Ultra-Low Power Wireless Research. In *Proceedings of the 4^{th} International Conference on Information Processing in Sensor Networks: Special track on Platform Tools and Design Methods for Network Embedded Sensors (IPSN/SPOTS)*, April 2005.

[51] Nithya Ramanathan, Thomas Schoellhammer, Eddie Kohler, Kamin Whitehouse, Thomas Harmon, and Deborah Estrin. Suelo: human-assisted sensing for exploratory soil monitoring studies. In *Proceedings of the 7^{th} ACM Conference on Embedded Networked Sensor Systems*, pages 197–210, 2009.

[52] Thomas Schmid, Roy Shea, Mani B. Srivastava, and Prabal Dutta. Disentangling wireless sensing from mesh networking. In *Proceedings of HotEmNets*, 2010.

[53] L. Selavo, A. Wood, Q. Cao, A. Srinivasan, H. Liu, T. Sookoor, and J. Stankovic. Luster: Wireless Sensor Network for Environmental Research. In *Proceedings of the 5^{th} ACM Conference on Embedded Networked Sensor Systems (SenSys)*, November 2007.

[54] SmartLabs, Inc. Insteon: The details. Available from: `http://www.insteon.net/pdf/insteondetails.pdf`, 2005.

[55] Dongjin Son, Bhaskar Krishnamachari, and John Heidemann. Experimental study of concurrent transmission in wireless sensor networks. In *Proceedings of the ACM Sensys*, November 2006.

[56] Kannan Srinivasan, Maria Kazandijeva, Saatvik Agarwal, and Philip Levis. The β-factor: Measuring Wireless Link Burstiness. In *Proceedings of the 6^{th} ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2008.

[57] Yanjun Sun, Omer Gurewitz, and David B. Johnson. RI-MAC: a receiver-initiated asynchronous duty cycle mac protocol for dynamic traffic loads in wireless sensor networks. In *Proceedings of the 6th International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 1–14, Nov 2008.

[58] Robert Szewczyk, Alan Mainwaring, Joseph Polastre, John Anderson, and David Culler. An analysis of a large scale habitat monitoring application. In *Proceedings of the 2nd international conference on Embedded networked sensor systems (SenSys)*, 2004.

[59] A. Terzis, R. Musaloiu-E., J. Cogan, K. Szlavecz, A. Szalay, J. Gray, S. Ozer, C.-J. M. Liang, J. Gupchup, and R. Burns. Wireless Sensor Networks for Soil Science. *International Journal on Sensor Networks on Environmental Sensor Networks*, 7(1/2):53 – 70, 2010.

[60] Andreas Terzis, Razvan Musaloiu-E, Josh Cogan, Katalin Szlavecz, Alex Szalay, Jim Gray, Stewart Ozer, Chieh-Jan Mike Liang, Jayant Gupchup, and Randal Burns. Wireless sensor networks for soil science. *International Journal on Sensor Networks*, In press, 2009.

[61] Texas Instruments. 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver. Available at `http://www.chipcon.com/files/CC2420_Data_Sheet_1_3.pdf`, 2006.

[62] Texas Instruments. CC2420: 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver. Available at `http://www.ti.com/lit/gpn/cc2420`, 2006.

[63] Texas Instruments. MSP430x1xx Family User's Guide (Rev. F). Available at `http://www.ti.com/litv/pdf/slau049f`, 2006.

[64] Texas Instruments. CC430 Family User's Guide. Available at `http://www.ti.com/lit/ug/slau259e/slau259e.pdf`, 2010.

[65] Texas Instruments. MSP430 SoC with RF Core. Available at `http://www.ti.com/product/cc430f5137/`, 2010.

[66] Texas Instruments. CC1101 Low-Power Sub-1 GHz RF Transceiver (Rev. H). Available at `http://www.ti.com/product/cc1101`, 2012.

[67] Texas Instruments. CC1190 850 - 950MHz RF Front End. Available at `http://www.ti.com/product/cc1190`, 2013.

[68] Yin Wang, Yuan He, Xufei Mao, Yunhao Liu, Zhiyu Huang, and Xiangyang Li. Exploiting constructive interference for scalable flooding in wireless networks. In *INFOCOM, 2012 Proceedings IEEE*, pages 2104–2112, march 2012.

[69] K. Whitehouse, A. Woo, F. Jiang, J. Polastre, and D. Culler. Exploiting the capture effect for collision detection and recovery. In *Proceedings of the 2nd IEEE workshop on Embedded Networked Sensors (EmNets)*, 2005.

[70] T. Winter, P. Thubert, and RPL Author Team. Rpl: Ipv6 routing protocol for low power and lossy networks. Internet Draft, IETF, 2010.

[71] Alec Woo, Terence Tong, and David Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *SenSys*, 2003.

[72] M. Yarvis, N. Kushalnagar, H. Singh, A. Rangarajan, Y. Liu, and S. Singh. Exploiting heterogeneity in sensor networks. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 2, pages 878 – 890 vol. 2, march 2005.

[73] Lohit Yerva, Brad Campbell, Apoorva Bansal, Thomas Schmid, and Prabal Dutta. Grafting energy-harvesting leaves onto the sensornet tree. In *Proceedings of the 11th international conference on Information Processing in Sensor Networks*, IPSN '12, pages 197–208, New York, NY, USA, 2012. ACM.

Vita

Douglas Carlson received his Bachelor of Science degree in Computer Science at Duke University in 2004. In 2008, he left a brief career in web application development and business process management consulting to join Dr. Andreas Terzis's Hopkins InterNetworking Research Group at Johns Hopkins University. He received his Master of Science in Engineering Degree in 2010. His research interests are wireless sensor networks for long-term environmental monitoring and low-power communication protocols that leverage non-destructive radio interference.

Upon completing his Ph.D. in 2014, Douglas will join the Embedded Systems group at Hillcrest Laboratories, a consumer electronics company in Rockville, MD.