Washington University in St. Louis
Washington University Open Scholarship

All Theses and Dissertations (ETDs)

Summer 9-1-2014

# Wireless Sensor Networking in Challenging Environments

Mo Sha
*Washington University in St. Louis*

Follow this and additional works at: http://openscholarship.wustl.edu/etd

Recommended Citation
Sha, Mo, "Wireless Sensor Networking in Challenging Environments" (2014). *All Theses and Dissertations (ETDs)*. 1344.
http://openscholarship.wustl.edu/etd/1344

This Dissertation is brought to you for free and open access by Washington University Open Scholarship. It has been accepted for inclusion in All Theses and Dissertations (ETDs) by an authorized administrator of Washington University Open Scholarship. For more information, please contact digital@wumail.wustl.edu.

WASHINGTON UNIVERSITY IN ST. LOUIS

School of Engineering and Applied Science

Department of Computer Science and Engineering

Dissertation Examination Committee:
Chenyang Lu, Chair
Christopher Gill
Humberto Gonzalez
Raj Jain
Jonathan Turner
Guoliang Xing

# Wireless Sensor Networking in Challenging Environments

by

Mo Sha

A dissertation presented to the Graduate School of Arts and Sciences
of Washington University in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

August 2014
Saint Louis, Missouri

© 2014, Mo Sha

# Contents

# List of Figures

# List of Tables

# Acknowledgments

This dissertation would not have been possible without the support and collaboration of my many colleagues. I would like to thank my advisor Dr.Chenyang Lu for his guidance throughout my doctoral research, and my committee members Dr.Christopher Gill, Dr.Humberto Gonzalez, Dr.Raj Jain, Dr.Jonathan Turner, Dr.Guoliang Xing for their guidance and feedback on my dissertation research. I would also like to thank Dr.Greg Hackmann, Rahav Dor, Chengjie Wu, Abu Sayeed Saifullah, Dolvara Gunatilaka, Dr.Tae-Suk Kim, and Dr.Taerim Park, who all played an important part in the work described here.

Last but not least, I would like to give my thanks to my family: my dear mom, dad, and my wife for encouraging me, supporting me, and being proud of me all the time. It is the foundation and ideals they instilled in me that made all of this come true.

Mo Sha

*Washington University in Saint Louis*
*August 2014*

Dedicated to my parents.

ABSTRACT OF THE DISSERTATION

Wireless Sensor Networking in Challenging Environments

by

Mo Sha

Doctor of Philosophy in Computer Science

Washington University in St. Louis, 2014

Professor Chenyang Lu, Chair

Recent years have witnessed growing interest in deploying wireless sensing applications in real-world environments. For example, home automation systems provide fine-grained metering and control of home appliances in residential settings. Similarly, assisted living applications employ wireless sensors to provide continuous health and wellness monitoring in homes. However, real deployments of Wireless Sensor Networks (WSNs) pose significant challenges due to their low-power radios and uncontrolled ambient environments. Our empirical study in over 15 real-world apartments shows that low-power WSNs based on the IEEE 802.15.4 standard are highly susceptible to external interference beyond user control, such as Wi-Fi access points, Bluetooth peripherals, cordless phones, and numerous other devices prevalent in residential environments that share the unlicensed 2.4 GHz ISM band with IEEE 802.15.4 radios.

To address these real-world challenges, we developed two practical wireless network protocols including the Adaptive and Robust Channel Hopping (ARCH) protocol and the Adaptive

Energy Detection Protocol (AEDP). ARCH enhances network reliability through opportunistically changing radio's frequency to avoid interference and environmental noise and AEDP reduces false wakeups in noisy wireless environments by dynamically adjusting the wakeup threshold of low-power radios.

Another major trend in WSNs is the convergence with smart phones. To deal with the dynamic wireless conditions and varying application requirements of mobile users, we developed the Self-Adapting MAC Layer (SAML) to support adaptive communication between smart phones and wireless sensors. SAML dynamically selects and switches Medium Access Control protocols to accommodate changes in ambient conditions and application requirements.

Compared with the residential and personal wireless systems, industrial applications pose unique challenges due to their critical demands on reliability and real-time performance. We developed an experimental testbed by realizing key network mechanisms of industrial Wireless Sensor and Actuator Networks (WSANs) and conducted an empirical study that revealed the limitations and potential enhancements of those mechanisms. Our study shows that graph routing is more resilient to interference and its backup routes may be heavily used in noisy environments, which demonstrate the necessity of path diversity for reliable WSANs. Our study also suggests that combining channel diversity with retransmission may effectively reduce the burstiness of transmission failures and judicious allocation of multiple transmissions in a shared slot can effectively improve network capacity without significantly impacting reliability.

# Chapter 1

# Introduction

Recent years have witnessed growing interest in deploying wireless sensing applications in real-world environments. For example, home automation systems provide fine-grained metering and control of home appliances in residential settings. Similarly, assisted living applications employ wireless sensors to provide continuous health and wellness monitoring in homes. However, real deployments of Wireless Sensor Networks (WSNs) pose significant challenges due to their low-power radios and uncontrolled ambient environments. We performed two in-depth empirical studies on wireless channels in real-world residential environments, providing key design guidelines for meeting the reliable wireless communication constraints of residential sensing applications. The *spectrum study* analyzes spectrum usage in the 2.4 GHz band where WSNs based on the IEEE 802.15.4 standard must coexist with existing wireless devices. We characterize the ambient wireless environment in six apartments through passive spectrum analysis across the entire 2.4 GHz band over seven days in each apartment. The *multi-channel link study* measures the reliability of different 802.15.4 channels through active probing with motes in ten apartments. The empirical studies show that low-power WSNs based on the IEEE 802.15.4 standard are highly susceptible to external interference beyond user control, such as Wi-Fi access points, Bluetooth peripherals, cordless phones, and numerous other devices prevalent in residential environments that share the unlicensed 2.4 GHz ISM band with IEEE 802.15.4 radios [104] [106] [101].

To address these real-world challenges, we developed two practical wireless network protocols including the *Adaptive and Robust Channel Hopping (ARCH)* protocol [103] and the *Adaptive Energy Detection Protocol (AEDP)* [105]. ARCH enhances network reliability through channel diversity: devices opportunistically change their radio's frequency in order to avoid

adverse channel conditions such as interference and environmental noise. ARCH has several key features. First, ARCH is an *adaptive* protocol that channel-hops based on changes in channel quality observed in real time. Second, ARCH is a *distributed* protocol that selects channels on a per-link basis, due to the large link-to-link variations in channel quality observed under empirical study. Third, ARCH is designed to be *robust and lightweight*. ARCH uses a practical handshaking approach to handle channel desynchronization and an efficient sliding-window scheme that does not involve expensive calculations or modeling, and can be reasonably implemented on memory-constrained wireless sensor platforms. Fourth, ARCH introduces *minimal communication overhead* for applications where packet acknowledgements are already enabled. We evaluate our approach through real deployment in real-life apartments with residents' daily activity. Our experimental results demonstrate that ARCH can effectively reduce packet retransmissions and improve delivery rate on the unreliable links with relatively few channel hops per day.

Low Power Listening (LPL) is a common Medium Access Control (MAC) layer technique for reducing energy consumption in WSNs, where nodes periodically wake up to sample the wireless channel to detect activity. However, LPL is highly susceptible to *false wakeups* caused by environmental noise being detected as activity on the channel, causing nodes to spuriously wake up in order to receive nonexistent transmissions. In our empirical studies in residential environments, we observe that the false wakeup problem can significantly increase a node's duty cycle, compromising the benefit of LPL. We also find that the energy-level threshold used by the Clear Channel Assessment (CCA) mechanism to detect channel activity has a significant impact on the false wakeup rate. We then design AEDP, an adaptive energy detection protocol for LPL, which dynamically adjust a node's CCA threshold to meet application-specified bounds on network reliability and duty cycle. Empirical experiments in both controlled tests and real-world environments show that AEDP can effectively mitigate the impact of noise on radio duty cycles, while maintaining satisfactory link reliability.

The other major trend in WSNs is the convergence with smart phones. To deal with the dynamic wireless conditions and varying application requirements of mobile users, we developed the *Self-Adapting MAC Layer (SAML)* [102] to support adaptive communication between smart phones and wireless sensors. SAML dynamically selects and switches MAC protocols to changes in ambient conditions and application requirements. SAML comprises (1) a *Reconfigurable MAC Architecture (RMA)* that can switch to different MAC protocols

at run time and (2) a *learning-based MAC Selection Engine* that selects the protocol most suitable for the current condition and requirements. To the application SAML appears as a traditional MAC layer and realizes its benefits through a simple API for the mobile applications. We have implemented SAML in TinyOS 2.x and built three prototypes containing up to five MACs. We evaluate the system in controlled tests and real-world environments using a new gateway device that integrates a 802.15.4 radio with Android phones. Our experimental results show that SAML can effectively adapt MAC layer behavior to meet varying application requirements in dynamic environments through judicious selection and efficient switching of MAC protocols.

Compared with the residential and personal wireless systems, industrial applications such as process automation pose unique challenges due to their critical demands on reliability and real-time performance. We developed an experimental testbed by realizing key network mechanisms of industrial Wireless Sensor-Actuator Networks (WSANs) including multi-channel TDMA with shared slots at the MAC layer and reliable graph routing. We then performed an in-depth empirical study on the reliability, latency, and energy consumption of variant solutions under clean, noisy, and stress testing conditions, providing key insights for meeting the reliable and real-time constraints of industrial applications. Our study shows that graph routing is more resilient to interference and its backup routes may be heavily used in noisy environments, which demonstrate the necessity of path diversity for reliable WSANs. Our study also suggests that combining channel diversity with retransmission may effectively reduce the burstiness of transmission failures and judicious allocation of multiple transmissions in a shared slot can effectively improve network capacity without significantly impacting reliability.

The rest of the dissertation is organized as follows. Chapter 2 introduces our empirical studies on wireless channels in real-world residential environments. Chapter 3 presents our ARCH protocol and Chapter 4 shows our AEDP protocol. Chapter 5 discusses our design of SAML and Chapter 6 presents our experimental testbed of industrial WSANs and our empirical study. Chapter 7 concludes this dissertation.

# Chapter 2

# Empirical Studies of Home-Area Sensor Networks

## 2.1 Introduction

In recent years, there has been growing interest in various wireless sensing applications in residential environments. For example, smart energy systems provide fine-grained metering and control of home appliances in residential settings. Similarly, assisted living applications such as vital sign monitoring and fall detection leverage wireless sensors to provide continuous health monitoring in homes. Wireless sensor networks offer a promising platform for home automation applications because they do not require a fixed wired infrastructure. Hence, home area networks (HANs) based on wireless sensor network technology can be used to easily and inexpensively retrofit existing apartments and households without the need to run dedicated cabling for communication and power [1] [50]. HAN applications have increasingly adopted the IEEE 802.15.4 wireless personal area network standard to provide wireless communication among sensors and actuators. 802.15.4 radios are designed to operate at a low data rate and be inexpensively manufactured, making them a good fit for residential applications where energy consumption and manufacturing costs are often at a premium. Industry standards such as ZigBee Smart Energy have adopted 802.15.4 technology for use in residential automation applications. The IETF has promoted efforts to standardize IPv6 on top of 802.15.4 for integrating wireless sensors into the Internet.

[1] Previous study shows that power becomes a scarce resource once the number of sensors exceeded the number of 120V wall sockets in each home (typically 20-30). Furthermore, wall-powered nodes were 2.3x more likely to lose power than battery-powered nodes [50]

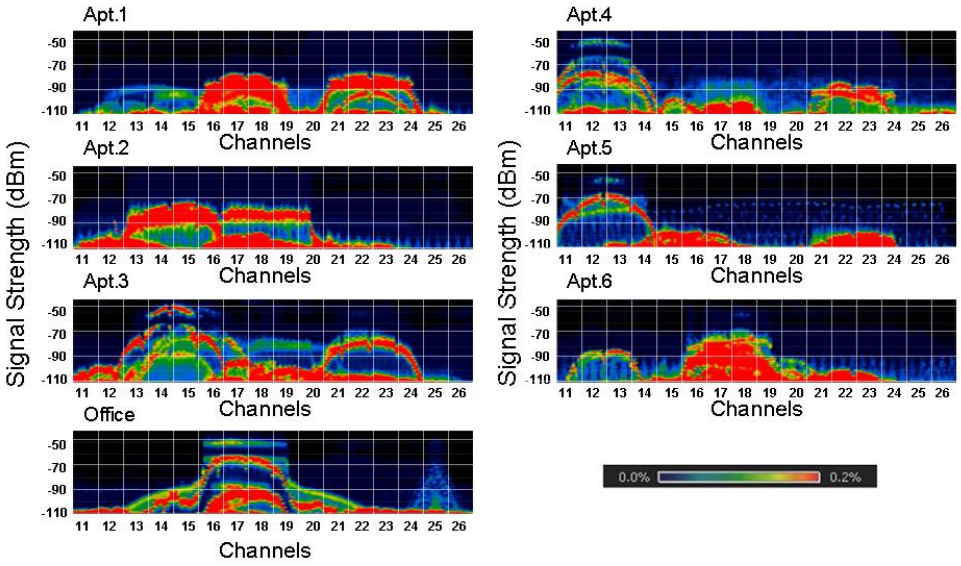



Figure 2.1: Histogram over 7 days' raw energy traces. X axis indicates 802.15.4 channels, Y axis indicates power, and color indicates how often a signal was detected at $x$ GHz with an energy level of $y$ dBm.

However, HANs pose unique challenges in wireless communication due to their low-power radios and uncontrolled residential environments. HANs typically feature low data rates but require high network reliability in uncontrolled residential environments. Our study shows that low-power IEEE 802.15.4 channels are highly susceptible to external interference beyond user control, such as Wi-Fi access points, Bluetooth peripherals, cordless phones, and numerous other devices prevalent in residential environments that share the unlicensed 2.4 GHz ISM band with IEEE 802.15.4 radios.

Figure 2.1 illustrates this challenge with raw spectrum usage traces collected from the 2.4 GHz spectrum in six apartments and an office building (described in more detail in Section 2.3). The office environment provides a relatively clean and predictable wireless environment, with only two major sources of noise: a campus-wide 802.11g network in the middle of the spectrum, and a 802.15.4 sensor network testbed at the upper end. In contrast, the

residential settings present a much noisier and more varied environment; for example, apartments 4 and 5 show sporadic interference across the entire 2.4 GHz spectrum (represented by blue shapes spanning nearly the entire X axis) which could complicate finding a persistently reliable communication channel. These results highlight a fundamental challenge of residential deployments: while the wireless devices in industrial and office settings are typically centrally managed, resulting in more predictable noise patterns, residential settings present numerous sources of environmental noise due to a lack of spectrum management. This challenge is compounded by the fact that wireless signals may traverse multiple neighboring residences, subjecting neighbors' networks to interference beyond their control. For example, in just one apartment in our dataset, a deployed laptop was able to decode beacons from 28 distinct Wi-Fi access points.

In this chapter, we present a two-part empirical study which aims to characterize the real-world network performance of HANs, focusing specifically on devices based on the 802.15.4 standard. Our study is divided into two major parts. First, we carry out an analysis over spectrum analyzer traces collected in six apartments. This spectrum study of ambient wireless conditions in homes illustrates the challenge of finding a "clean" part of the shared 2.4 GHz spectrum in such settings. Our analysis demonstrates that the wireless environments in these apartments are much more crowded and more variable than an office setting. Moreover, while 802.11 WLANs contribute a significant fraction of the spectrum usage, we also identified signals across the 2.4 GHz band indicating non-negligible noise from non-802.11 devices.

Second, we explore how these challenging environments may directly affect applications' QoS, through an active probing study of wireless link reliability across all 16 channels in ten apartments. This second study focuses on packet reception ratio (PRR), which is both a direct indicator of link reliability and closely related to other important QoS metrics such as latency and energy consumption. From this active study, we make several more key observations which could greatly impact the QoS of wireless sensor networks deployed in residential environments: (1) Link reliability varies significantly from channel to channel and over time. (2) In a typical apartment environment, there may not be a single channel which is persistently reliable for 24 hours. (3) Retransmissions alone are insufficient for HANs due to the burstiness of packet losses. (4) Exploiting channel diversity by occasional channel hopping at runtime can effectively maintain long-term reliable communication. (5)

Channel conditions are not cyclic. (6) Reliability is strongly correlated across adjacent channels; channel-hopping should move as far away as possible from a failing channel. (7) Increasing transmission power may be effective for maintaining channel reliability, but is potentially expensive. Combining channel diversity with transmission power control is a promising strategy for controlling energy consumption while maintaining network reliability.

These findings reveal the characteristics of wireless channels and 2.4 GHz spectrum in residential environment, highlight the importance of channel diversity in managing HANs, and provide ground truth and findings as a foundation for developing reliable wireless communication approaches for HANs. For example, it highlights the importance of dynamic channel selection in managing HANs. Devices cannot be deployed with a factory-set default channel as no channel can consistently achieve long-term reliability in all the apartments we studied. Neither will a channel selected based on measurements at deployment time suffice either because of the time-varying nature of channel conditions. On the other hand, sustained reliability can be achieved by changing the channel only a few times a day. This observation motivates the design of HAN management tools with dynamic channel management functions that are not typically needed in Wi-Fi network management. Our study also provides insights for managing the co-existence of HANs with other wireless technology such as Wi-Fi. While co-existence of HANs and Wi-Fi has received attention in the literature [70], we found that other devices can also be non-negligible sources of interference. Therefore, co-existence solutions tailored specifically for Wi-Fi may not be effective in all residential environments. Instead, general solutions agonistic to specific co-existing wireless technology will be more effective in residential environments with diverse sources of interference.

The rest of the chapter is organized as follows. Section 2.2 reviews related work. Section 2.3 discusses the findings of our passive spectral study. Section 2.4 then presents our active probing study. Finally, we conclude in Section 2.5 by highlighting the implications of our findings on HAN design.

## 2.2 Related Work

Several recent studies have aimed to characterize the impact of interference on wireless networks through controlled experiments [51, 60, 110, 112, 134]. [89, 111, 127] present theoretical

analysis based on simulation study. Gummadi et al. [43] presents an empirical study on the impact of ZigBee and other interferers' impact on 802.11 links, proposing to alleviate interference with rapid channel-hopping in conjunction with 802.11b's existing support for Direct-Sequence Spread Spectrum (DSSS). Srinivasan et al. [115] examines the packet delivery behavior of two 802.15.4-based mote platforms, including the impact of interference from 802.11 and Bluetooth. Liang et al. [70] measures the impact of interference from 802.11 networks on 802.15.4 links, proposing the use of redundant headers and forward error correction to alleviate packet corruption. In contrast to these controlled studies, our own study examines the performance of HANs subject to normal residential activities and diverse interference sources. Due to the co-existence of diverse interference sources in these uncontrolled environments, our study considers ambient wireless conditions as a whole, rather than analyzing specific sources of interference. For example, our spectrum study showed that, while Wi-Fi is a significant source of interference in residential environments, non-Wi-Fi devices can also be non-negligible sources of interference. This result indicates that solutions tailored specifically for one type of co-existing wireless technology may not be effective in all residential environments.

Bahl et al. [14] presents a study of UHF white space networking, while Chen et al. [26] presents a large-scale spectrum measurement study followed by a 2-dimensional frequent pattern mining algorithm for channel prediction. These studies focus on supporting wide-area networks based on white space networking and the GSM band, respectively. Our own study focuses on the reliability of static, indoor wireless sensor networks designed for home environments, and on the unlicensed 2.4 GHz band used by IEEE 802.15.4 and shared by other wireless devices prevalent in residential environments. Accordingly, our study provides new insights into the reliability of HANs, including the high variability of residential wireless environments, the lack of persistently reliable wireless channels, the diverse sources of interference (including the non-negligible impact of non-Wi-Fi devices), and the effectiveness of occasional channel hopping in maintaining link reliability.

Papagiannaki et al. [85] performed an empirical study of home networks based on 802.11 technology. Our study considers devices based on the 802.15.4 standard, which operate at a much lower transmission power than 802.11 devices and hence are significantly more susceptible to interference. Our study therefore leads to a different set of observations that underscores the impact of spectrum usage on these low-power 802.15.4 networks.

Ortiz et al. evaluates the multi-channel behavior of 802.15.4 networks in a machine room, a computer room, and an office testbed. Ortiz's study finds path diversity to be an effective strategy to ensure reliability. Our own study in residential environments provides many different insights on low wireless characteristics compared with what is observed in Ortiz's study. The residential settings in our study exhibit more complex noise patterns and higher variability than the environments studied by Ortiz. This difference may be attributed to homes being open environments with no centralized control on spectrum usage; many 2.4 GHz devices are used in homes, and the physical proximity of some residences means that strong interferers (such as 802.11 APs, Bluetooth devices, and cordless phones) may even affect the wireless conditions in other homes. Accordingly, our active study in Section IV finds exploiting channel diversity to be an attractive strategy for ensuring reliability in residential environments. We note that channel and path diversity are orthogonal strategies; the two could be used together in particularly challenging wireless environments.

Hauer et al. [47] discusses a multi-channel measurement of Body Area Networks (BANs) and proposes a noise floor-triggered channel hopping scheme to detect and mitigate the effects of interference. Hauer's study features controlled indoor experiments along with outdoor experiments carried out during normal urban activity. Shah el al. [108] performed a controlled experiment to study the effect of the human body on BANs. Shah's study measures the effects of various activities (sitting, standing, and walking) and node placements (ear, chest, waist, knee, and ankle) on 802.15.4 radio performance. Instead of body-area networks, our own study focuses on HANs designed for smart energy, which feature significantly different setups and wireless properties. Moreover, our study is performed under normal home activities, providing a realistic setting to evaluate HAN performance.

## 2.3 Wireless Spectrum Study

In this section, we present a study of the ambient wireless conditions in real-world residential environments. For this study, we collected 7 days' energy traces in the 2.4 GHz spectrum from six apartments in different neighborhoods. A detailed description of the experimental settings may be found in Table 2.1.

| Name | Begin Date | End Date |
|---|---|---|
| Apt. 1 | 2:00pm, Apr. 4, 2010 | 3:30pm, Apr. 19, 2010 |
| Apt. 2 | 6:50pm, June 30, 2010 | 6:50pm, July 7, 2010 |
| Apt. 3 | 9:05pm, May 12, 2010 | 11:29pm, May 20, 2010 |
| Apt. 4 | 11:40am, June 6, 2010 | 12:40pm, June 13, 2010 |
| Apt. 5 | 12:25pm, Apr. 20, 2010 | 10:50am, Apr. 28, 2010 |
| Apt. 6 | 7:00pm, July 7, 2010 | 9:00pm, July 14, 2010 |
| Office | 1:15pm, July 16, 2010 | 1:20pm, July 23, 2010 |

Table 2.1: The settings and dates where the spectrum data was collected.

As a baseline for comparison, we also collected energy traces from an office in Bryan Hall at Washington University in St. Louis. We note that this baseline is meant to illustrate how controlled testbed settings within an office environment may potentially be very different from real home environments; it is not meant to be a comprehensive study of office environments.

Specifically, this study addresses the following questions. (1) Is there a common area of the 2.4 GHz spectrum which is free in all apartments? (2) Does spectrum usage change with time? (3) Do residential settings have similar spectrum usage properties as office settings? (4) Is Channel Occupancy Temporally Correlated? (5) Is 802.11 the dominant interferer in residential environments?

### 2.3.1 Experimental Methodology

We are primarily interested in the spectrum usage between 2.400 GHz and 2.495 GHz, which are the parts of the spectrum used by the 802.15.4 standard for wireless sensor networks. To analyze this part of the spectrum, we collected energy traces using a laptop equipped with a Wi-Spy 2.4x spectrum analyzer [1]. The Wi-Spy sweeps across the 2.4 GHz spectrum approximately once every 40 ms, returning a signal strength reading (in dBm) for each of 254 discrete frequencies. We continuously collected energy traces for 7 days in each apartment during the residents' normal daily activities, as well as in an office in Bryan Hall. The resulting traces contained 15,120,000 readings for each of the 254 frequencies, resulting in a data set of approximately 2.5 GB per location. Figure 2.1 presents a histogram of the raw spectrum usage data in all seven datasets.

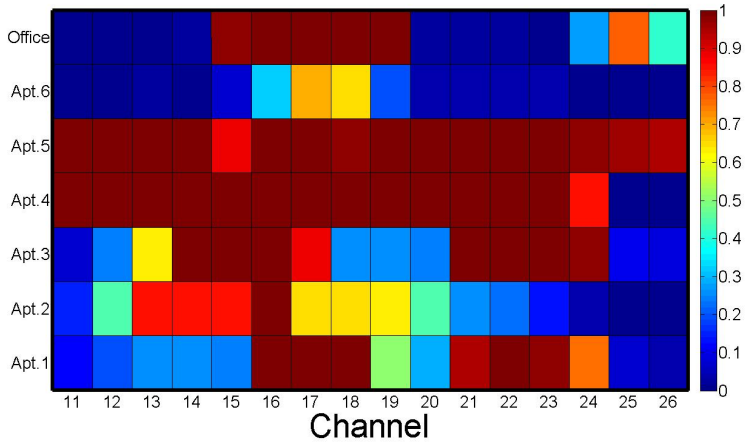



Figure 2.2: Channel occupancy rate. X axis designates channels, Y axis designates experimental settings, and color represents the proportion of readings above the occupancy threshold.

For the purposes of analysis, we apply a thresholding process like that employed in [26] to convert signal strength readings into binary values, with 0 denoting a channel being idle and 1 denoting a channel being busy. We found experimentally that a receive signal strength of $-80$ dBm is needed to create a high-quality link between a pair of Chipcon CC2420 radios; however, a noise level of $-85$ dBm or higher would be enough to induce packet drops on such a link [106]. Hence, throughout our analysis, we use $-85$ dBm as our threshold value to denote a busy channel. Using a constant threshold allows for a fair comparison across different apartments. While the specific numerical results of our analysis are dependent on the threshold, the trends and observations we make from these results should generally apply to other threshold values.

To assess the impact of ambient wireless signals on HANs, we aggregate the data from the Wi-Spy's 254 channels into the 16 channels used by the 802.15.4 standard; i.e., an 802.15.4 channel is deemed busy if any of its corresponding Wi-Spy channels are busy.

### 2.3.2 Is There a Common Idle Channel in Different Homes?

We first considered whether any 802.15.4 channel can be considered "clean" in all the tested residences. If such a channel exists, it could be used as a default, factory preset channel for

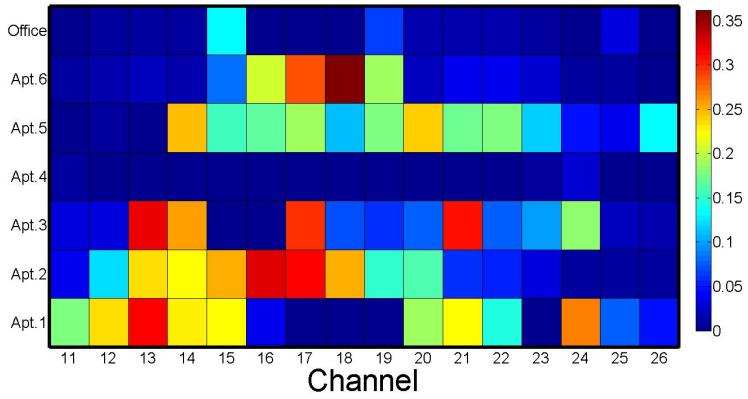
HANs. For example, channel 26 is often assumed as a good default channel, because it does not overlap with the spectrum used by 802.11 in North America.

To determine this, we calculate the channel occupancy rate — i.e., the proportion of samples that exceeded the −85 dBm threshold — over all channels in the six apartments and the office building. High occupancy rates correspond to a large proportion of samples where interference could have caused packet loss on an otherwise high-quality link.

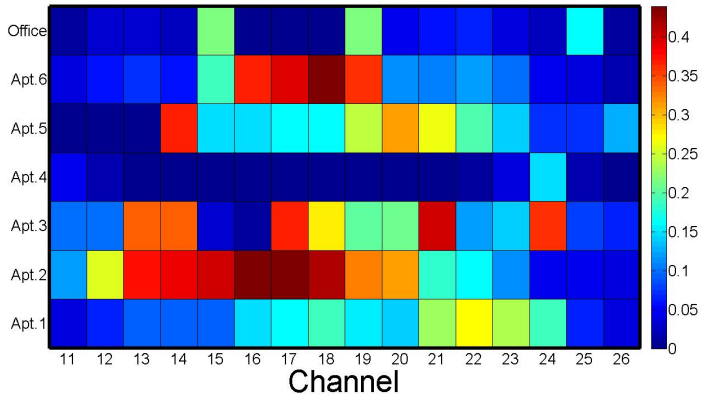
Figure 2.2 plots the occupancy rate of each channel in each location. If we compare Figures 2.1 and 2.2, we can note various phenomena that prevent finding a common idle channel. For example, apartment 5 has a channel occupancy rate above 95% for 15 of its 16 channels. Notably, even channel 26 has a channel occupancy rate as high as 95.04%, contradicting the commonly-held assumption that channel 26 will be open. The uniformly high occupancy rate across channels is likely caused by a relatively high-power spread-spectrum signal across the whole 2.4 GHz spectrum, which appears in Figure 2.1 as a series of thin blue arches. Devices with such wireless footprints include Bluetooth transmitters, baby monitors, wireless speaker systems, and game controllers [2]. (Unfortunately, by the very nature of residential environments lacking central management of wireless devices, there is no way to be certain about the sources of some of these phenomena.)

The only channel in apartment 5 with an occupancy rate below 95% is channel 15, which in contrast has an occupancy rate of 100.0% in apartments 3 and 4; thus, there is no common good channel in these apartments. In the case of apartment 3, channel 15 is unusable due to it intersecting with the middle of multiple 802.11 APs, represented as superimposed arcs on the left side of apartment 3's energy trace. For apartment 4, we see that only channels 25 and 26 have low occupancy rates; this phenomena is likely caused by the tall blue shape across most of apartment 4's energy trace, corresponding to some sporadic but high-power interferer.

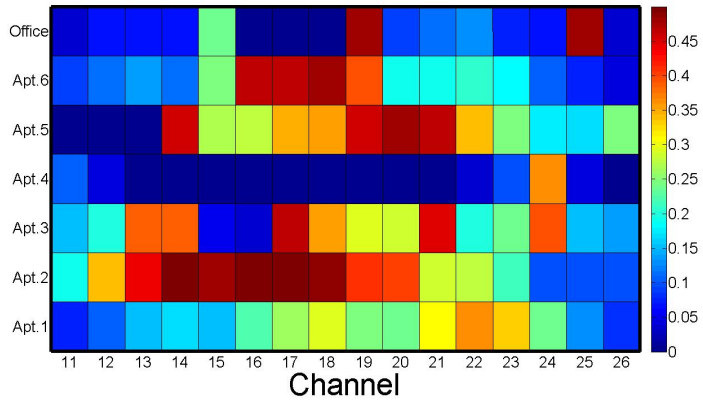
***Observation S1****: There may not exist a common idle channel across different homes, due to significant diversity in their spectrum usage patterns.*




(a) Daily standard deviation




(b) Hourly standard deviation




(c) 5-minute standard deviation

Figure 2.3: The standard deviation in channel occupancy rate at different timescales.

### 2.3.3 Does Spectrum Usage Change with Time?

We next explored whether the spectrum was stable in these residential settings. If spectrum is stable within a given apartment, it would be possible for a technician to pick a single "best" channel for the HAN at deployment time and expect it to work well over a long time period.

To determine this, we calculated the standard deviation in occupancy ($\sigma$) for each apartment and each channel. Figure 2.3 plots the standard deviation from day-to-day, from hour-to-hour, and for every 5 minutes. We see that channel conditions in most apartments can be quite variable, regardless of the timescale used. Except for apartment 4, $\sigma$ ranges from 24.0%–36.2% for the worst channel at a daily timescale, from 27.4%–43.9% at an hourly timescale, and 36.4%–50.0% at a 5-minute timescale. Apartment 4 is stable across the spectrum on a day-to-day basis, with $\sigma \leq 2.5\%$ for all channels. However, even for this apartment, some variability emerges at shorter timescales, with channel 24 featuring a $\sigma = 14.9\%$ on an hourly timescale and $\sigma = 36.0\%$ at a 5-minute timescale.

We also note that the office had much lower variability than all but apartment 4. For example, at a daily timescale, 10 of the 16 channels had $\sigma < 1.0\%$, and the most highly-variable channel had $\sigma$ of only 13.7%. Indeed, even at a 5-minute timescale, only three channels reveal significant variability; these three channels are at the edge of the campus 802.11g network (15), at the center of the same network (19), and at the center of the building's 802.15.4 testbed (25).

***Observation S2****: Spectrum occupancy in homes can exhibit significant variability over time, whether looking at timescales of days, hours, or minutes.*

### 2.3.4 Is Channel Occupancy Temporally Correlated?

Although channel occupancy is highly variable even on a timescale of minutes, there may nevertheless be temporal correlations in channel usage on even shorter time scales (e.g., packet-to-packet). To determine if such a correlation exists, we computed the conditional channel usage function ($CCUF$) for each channel in each apartment. For $k > 0$, $CCUF(k)$

is the conditional probability that $k$ consecutive busy readings are followed by another busy reading; for $k < 0$, $CCUF(k)$ is the conditional probability that $|k|$ consecutive idle readings are followed by another idle reading.

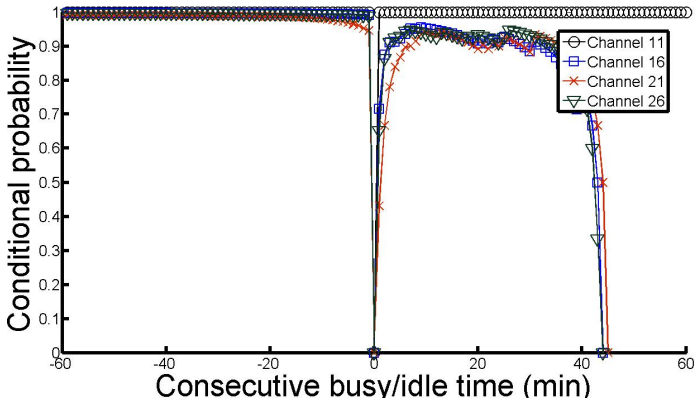
Figure 2.4 plots the $CCUF$ for three apartments and four channels; results for other apartments and other channels are similar but omitted for space. For all channels and all apartments, $CCUF$ rapidly stabilizes to $\geq 80\%$ within 10 minutes, indicating that a small channel-assessment window is sufficient to estimate channel condition with high probability. Moreover, the CCUF curve remains relatively flat after increasing to $\geq 80\%$. This indicates that longer windows (of 20 to 40 minutes) have minimal benefit for predicting channel conditions.

***Observation S3****: A short ($\leq$ 10 minute) channel assessment window is sufficient for estimating channel conditions with high probability; larger time windows provide minimal benefit.*

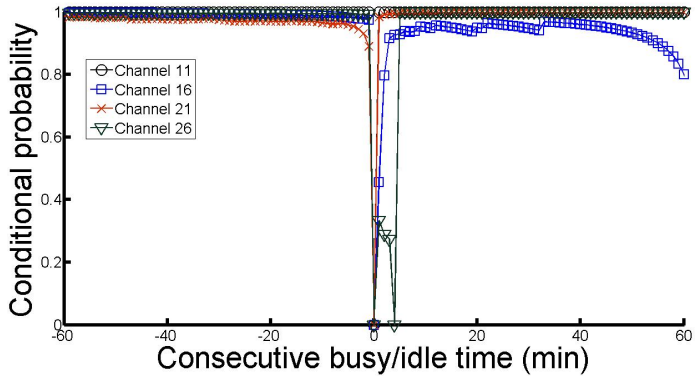
### 2.3.5 Is Wi-Fi the Dominant Source of Spectrum Usage?

Because of Wi-Fi's ubiquity and relatively high transmission power, it is often treated as a dominant interferer. Thus, our final analysis of our passive spectrum data is to identify whether there are other significant sources of interference. If Wi-Fi is indeed the dominant interferer in residential settings, then HANs could leverage solutions which are specifically designed to avoid interference from Wi-Fi networks (e.g., [70]).

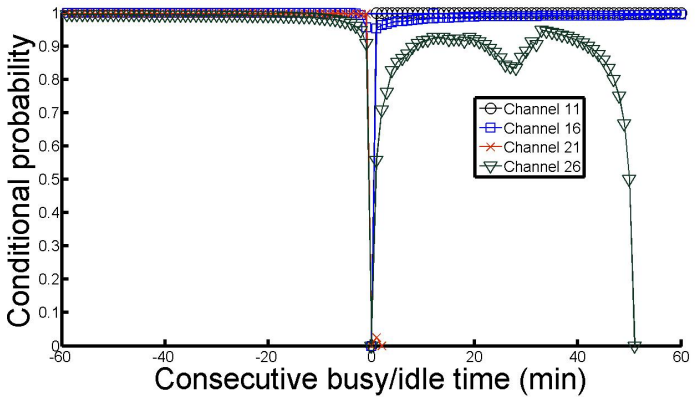
A visual inspection of Figures 2.1 and 2.2 suggests other important interferers besides Wi-Fi. Wi-Fi APs have a distinctive radiation pattern that manifests in Figure 2.1 as arcs the width of several 802.15.4 channels. For example, the energy traces for apartment 3 show two distinct arcs that are likely caused primarily by 802.11 APs configured to two different channels. Referring to Figure 2.2, we see that these areas of the spectrum are indeed highly occupied. However, looking at the energy trace for apartment 5, we see evidence of Wi-Fi APs on only part of the spectrum; nevertheless, the channel occupancy rate is above 95% for nearly the entire spectrum. This phenomena can be explained by the series of blue arcs across the 2.4 GHz spectrum, which indicate sporadic but high-powered spread-spectrum




(a) Apartment 1




(b) Apartment 3




(c) Apartment 5

Figure 2.4: Conditional channel usage functions ($CCUF$s) in three different apartments. The $X$ axis indicates consecutive busy or idle readings, where negative values represent consecutive idle readings and positive values represent consecutive busy readings. The $Y$ axis provides the probability that the channel is currently idle/busy given $x$ prior time slots which were all idle/busy.

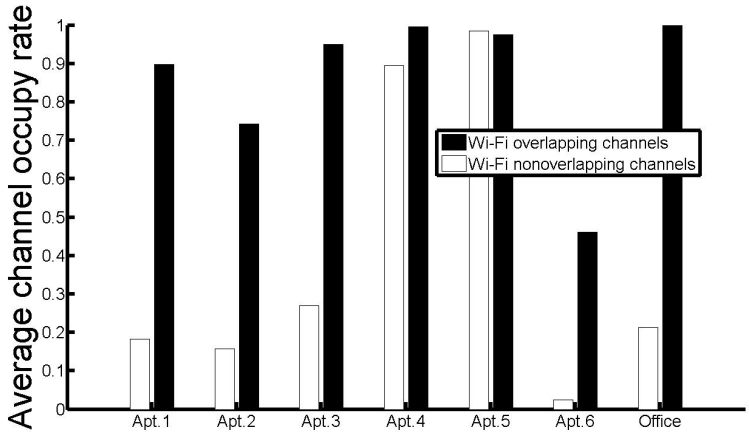



Figure 2.5: A comparison of the average channel occupancy rate between channels that overlap with Wi-Fi and channels that do not.

transmissions. (Again, by the nature of the environment, we cannot be certain about the source of this noise pattern.)

To quantify the relative impact of Wi-Fi, we leverage a feature of the Wi-Spy which logs the service set identifier (SSID) and 802.11 channel of all visible 802.11 access points (APs)[2]. Based on this data, we are able to divide the 802.15.4 channels in each apartment into two groups: those that overlap with 802.11 APs detectable from the corresponding apartment, and those that do not. We then calculated the average channel occupancy rate for each of the two groups in each apartment, as shown in Figure 2.5.

In most of the apartments, there is a clear distinction between the overlapping and non-overlapping channels. For example, apartment 1 has an average occupancy rate of 89.7% for the overlapping channels compared to 18.3% for the non-overlapping ones. But strikingly, we find that the non-overlapping channels are not *always* significantly more idle than those which overlap with Wi-Fi APs. In apartments 4 and 5, the channel occupancy rates of the non-overlapping channels are similar to the overlapping ones; indeed, in apartment 5, the non-overlapping channels are slightly more occupied on average than the overlapping ones. This observation can have important implications on the design of HANs, in that solutions

[2]Although many APs may be configured not to broadcast their SSID, we have observed that the Wi-Spy software can still identify these "hidden" access points in practice.

specifically designed to deal with Wi-Fi interference may not be effective in all residential environments.

***Observation S4****: While Wi-Fi is an important source of interference in residential environments, other interferers can also be non-negligible contributors to spectrum occupancy.*

## 2.4 Multi-Channel Link Study

In this section, we present a multi-channel link study in homes. The spectrum study presented in Section 2.3 focuses on characterizing the ambient wireless environment in homes. While link quality can be significantly influenced by interference from existing wireless signals, other factors such as signal attenuation and multi-path fading due to human activities can also impact the reliability of low-power wireless links. Our link study *directly* evaluates the multi-channel behavior of HANs by actively sending packets between motes equipped with 802.15.4 radios.

Specifically, this study addresses the following questions. (1) Can a HAN find a single persistently reliable channel for wireless communication? (2) If a good channel cannot be found, are packet retransmissions sufficient to deal with packet loss? (3) If no single channel can be used for reliable operation, can the network exploit channel diversity to achieve reliability? (4) Do channel conditions exhibit cyclic behavior over time? (5) Is reliability strongly correlated among different channels? (6) How effective is increasing transmission power for improving link reliability?

### 2.4.1 Experimental Methodology

For this active study, we carried out a series of experiments in ten real-world apartments in different neighborhoods, as listed in Table 2.2. (Due to the participating residents moving, only four of the apartments in this study are the same as those instrumented in the spectrum study.) Figure 2.6 shows an example floor plan of one of the apartments used in the study; a similar topology was deployed in the other apartments. Each experiment was carried out continuously for 24 hours with the residents' normal daily activities.

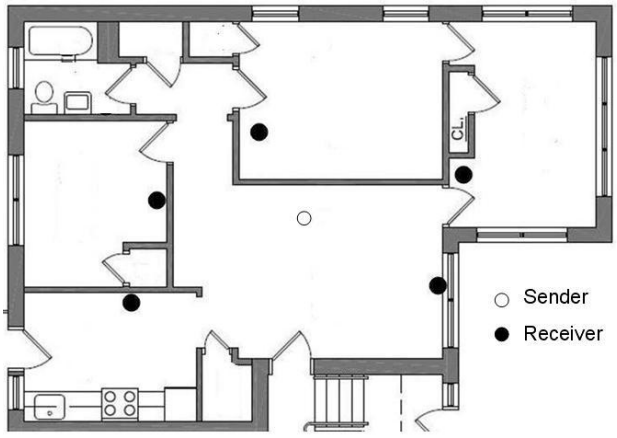



Figure 2.6: Floor plan of an apartment used in the study.

| | **Begin Date** | **End Date** |
|---|---|---|
| Apt. 1 | Sept. 30, 2009 | Oct. 1, 2009 |
| Apt. 2 | Sept. 30, 2009 | Oct. 1, 2009 |
| Apt. 3 | Oct. 3, 2009 | Oct. 4, 2009 |
| Apt. 4 | Oct. 3, 2009 | Oct. 4, 2009 |
| Apt. 5 | Sept. 30, 2009 | Oct. 1, 2009 |
| Apt. 6 | Sept. 12, 2009 | Sept. 13, 2009 |
| Apt. 7 | Oct. 3, 2009 | Oct. 4, 2009 |
| Apt. 8 | Sept. 18, 2009 | Sept. 19, 2009 |
| Apt. 9 | Oct. 6, 2009 | Oct. 7, 2009 |
| Apt. 10 | Oct. 6, 2009 | Oct. 7, 2009 |

Table 2.2: The settings and dates where the link data was collected.

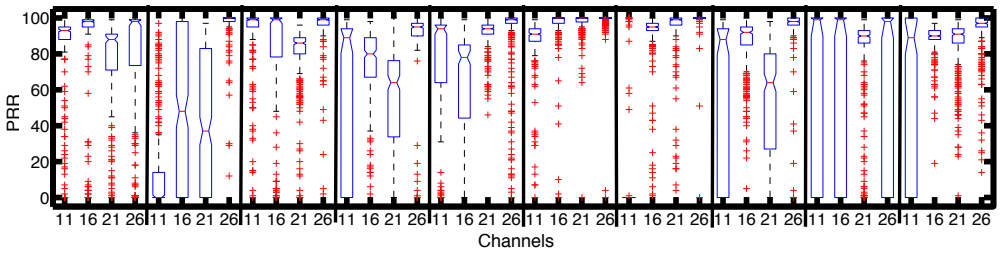



Figure 2.7: Box plot of the PRR for four channels in all ten apartments, calculated over 5-minute windows. Central mark in box indicates median; bottom and top of box represent the 25th percentile ($q_1$) and 75th percentile ($q_2$); crosses indicate outliers ($x > q_2 + 1.5 \cdot (q_2 - q_1)$ or $x < q_1 - 1.5 \cdot (q_2 - q_1)$); whiskers indicate range excluding outliers. Vertical lines delineate apartments.

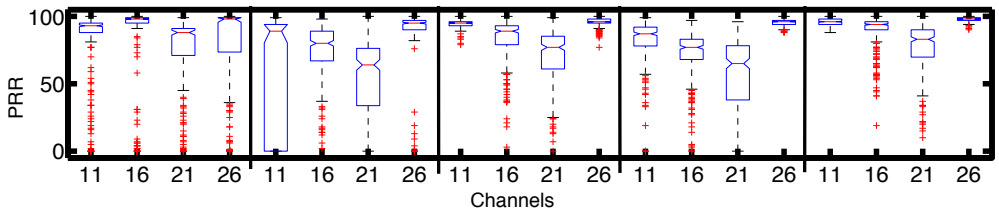



Figure 2.8: Box plot of the PRR of five different links in the same apartment on four channels, calculated over 5-minute windows. Vertical lines delineate links.

Our experiments were carried out using networks of Tmote Sky and TelosB [88] motes. Each mote is equipped with an IEEE 802.15.4 compliant Chipcon CC2420 radio [3]. IEEE 802.15.4 radios like the CC2420 can be programmed to operate on 16 channels (numbered 11 to 26) in 5 MHz steps. We leverage the CC2420's Received Signal Strength (RSS) indicator in our experiments to measure the signal power of environmental noise. Our experiments are written on top of the TinyOS 2.1 operating system [4] using the CC2420 driver's default CSMA/CA MAC layer.

We measure the packet reception ratio (PRR), defined as the fraction of transmitted packets successfully received by the receiver. PRR is not only a direct indicator of link reliability, but also closely related to other important QoS metrics such as latency and energy consumption. To measure the PRR of all channels at a fine granularity, we deployed a single transmitter node in each apartment which broadcast packets over each of the 16 channels. Specifically, the transmitter sent a batch of 100 consecutive packets to the broadcast address using a single wireless channel, then proceeded to the next channel in a round robin fashion. The process of sending 16 batches of 100 packets repeated every 5 minutes. The recipient nodes record the PRR over each batch of packets into their onboard flash memory. The use of a single sender and multiple recipients allowed us to test multiple links simultaneously while avoiding interference between senders. (Inter-link interference is not a major concern in many HANs due to the low data rates that are typically employed; for example, 1 temperature reading every 5 minutes is sufficient for an HVAC system to control ambient temperature.)

It is worth noting that HAN applications such as smart energy require persistent, long-term reliability. Transient link failures are non-negligible — these failures represent periods where parts of a household may experience sporadic service or no service at all (e.g., changing the thermostat may have no effect until a wireless link is restored minutes or hours later).

Hence, our study looks not just at the average PRR of each link but at its entire range of performance, including those outliers that indicate temporary failures.

In [115], links with a PRR below 10% were found to be poor-quality, and links with a PRR between 10% and 90% to be bursty. Accordingly, we use a PRR of 90% throughout this section as a threshold to designate links as "good" or "reliable".

### 2.4.2 Is There a Persistently Good Channel?

We first analyzed our data from the perspective of finding a single, persistently good channel across all of the tested apartments. Again, if a common good channel exists across all apartments, then it could be used as a preset default channel for HANs. For this analysis, we grouped the data from all links in all apartments together and then subdivided it by channel. Figure 2.7 presents a box plot of the PRR in 4 channels in all the apartments, where the PRR has been calculated over 5-minute windows. (The remaining 12 channels exhibit similar behavior and are omitted for reasons of clarity.) From this figure, we see significant variations in PRR on the same channel when moving from apartment to apartment. For example, channel 11 achieves a median PRR $> 90\%$ in apartments 1, 3, and 9, albeit with many outliers; however, the same channel has a near-zero median PRR in apartment 2. Only channel 26 has a median PRR above the 90% threshold in all apartments.

We also see significant variations in PRR from channel to channel, even in the same apartment. Strikingly, these variations even affect channel 26, which is often considered an open channel since it is nominally outside the 802.11 spectrum in North America. Although channel 26 achieves uniformly high *median* PRR in all apartments, there are numerous points during the experiment where the PRR falls much lower. For example, apartment 9 has a 25th percentile PRR of 0.0%, indicating a substantial portion of the experiment where the channel experienced total link failure.

Further analysis showed that there is not likely to be a single good channel across multiple links in the *same* apartment. We regrouped the PRR data, this time looking at the performance of each link/channel pair individually. Figure 2.8 presents a box-plot of the PRR for all five links within one apartment; again, for reasons of clarity, we present the data from

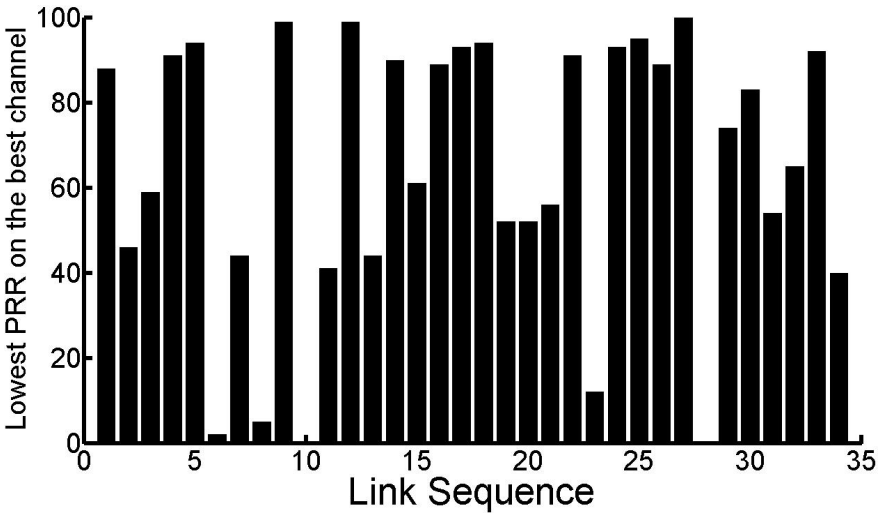



Figure 2.9: The lowest PRR observed on each link's most reliable channel.

only 4 of the 16 channels. We observe that the median PRR on a given channel varies greatly across links, particularly for outlier points. Again, this variation even affects channel 26: all five links have at least one outlier below the 90% threshold, and four links have numerous outliers below the threshold. Link 1 shows particularly high variance on channel 26, with a 25th-percentile PRR of only 73.5% in spite of a 98.0% median PRR. We also note that *all four* channels had numerous outliers below a PRR of 10%; that is, any single channel selection would have led to at least one link experiencing near-total disconnection at some point during the day.

Notably, each link had at least one channel with a high median PRR and low variance. For instance, as shown in Figure 8, link 1 shows a particularly high quality on channel 16 with a 99.3% median PRR and a variance less than 10%, while this link presents a high variance on channel 26, with a 25th-percentile PRR of only 73.5% in spite of a 98.0% median PRR. This indicates that all the links in our study are relevant to HAN applications given proper selection of channels.

***Observation L1****: Link reliability varies greatly from channel to channel.*

Looking at the entire dataset across all apartments, we found that few links were able to achieve a consistently high PRR, even on their most reliable channels. Figure 2.9 plots the lowest PRR observed on each link's most reliable channel: i.e., for the channel which achieves

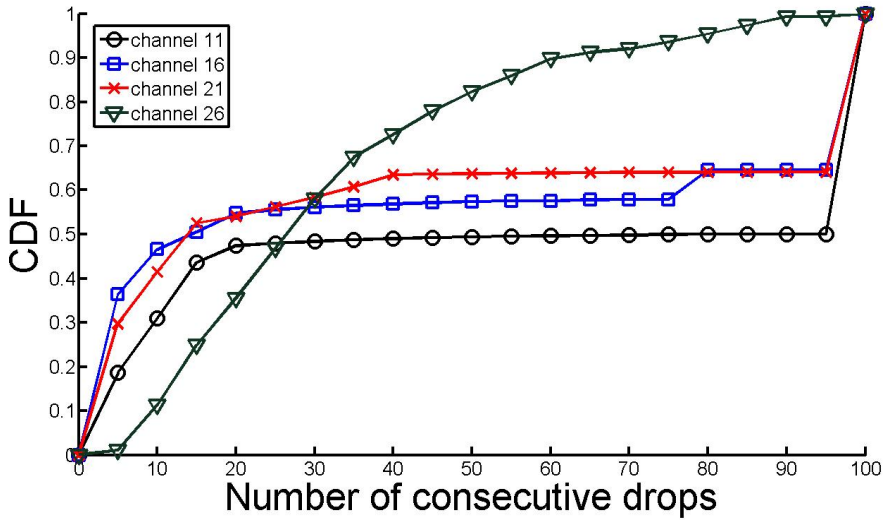



Figure 2.10: CDF of number of consecutive drops.

the highest average PRR over 24 hours, we plot the worst PRR out of all the 100-packet batches. Notably, only 12 of the 34 links in our dataset are able to persistently reach the 90% PRR threshold on even their best channel. Indeed, even lowering the threshold to 70%, more than half the links in our dataset would still have no persistently good channel.

***Observation L2****: Link reliability varies greatly over time, even within the same channel. Hence, even when selecting channels on a per-link basis, there is not always a single persistently reliable channel.*

### 2.4.3 Is Retransmission Sufficient?

Because retransmissions are effective in alleviating transient link failures, we next analyze whether it would be effective in alleviating the link failures observed in our experimental traces. However, we found that retransmissions alone are insufficient in residential environments, due to the bursty nature of the packet losses.

Figure 2.10 illustrates this problem with the cumulative probability density (CDF) of consecutive packet drops for all links on four channels. Specifically, we measured consecutive packet losses *within each batch* of 100 packets; we did not include inter-batch losses due to

the 5-minute gap between batches. Even on the best channel (channel 26), up to 85 consecutive packet drops were observed, and 10% of link failures lasted for more than 60 consecutive packets. On the remaining three channels, bursts of more than 95 consecutive packet drops were observed.

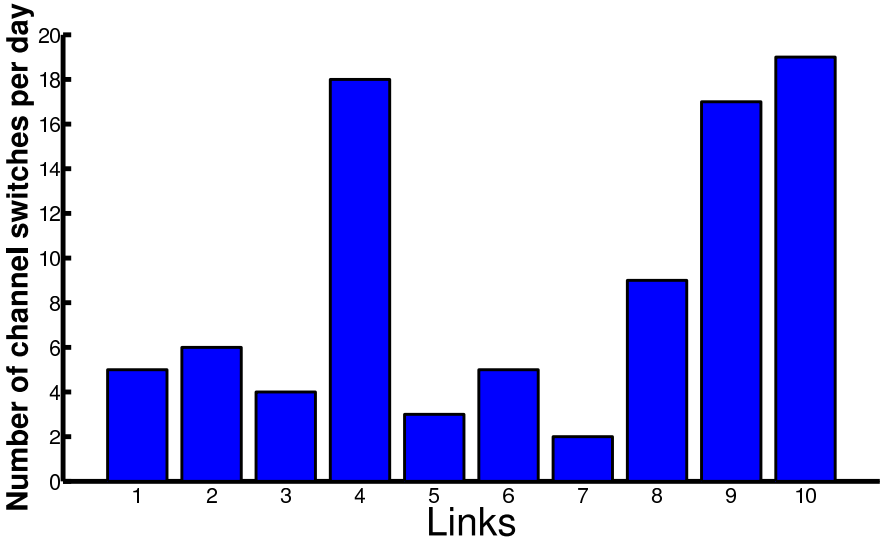
***Observation L3****: Retransmissions alone are insufficient for HANs due to the burstiness of packet losses.*

### 2.4.4 Is Channel Diversity Effective?

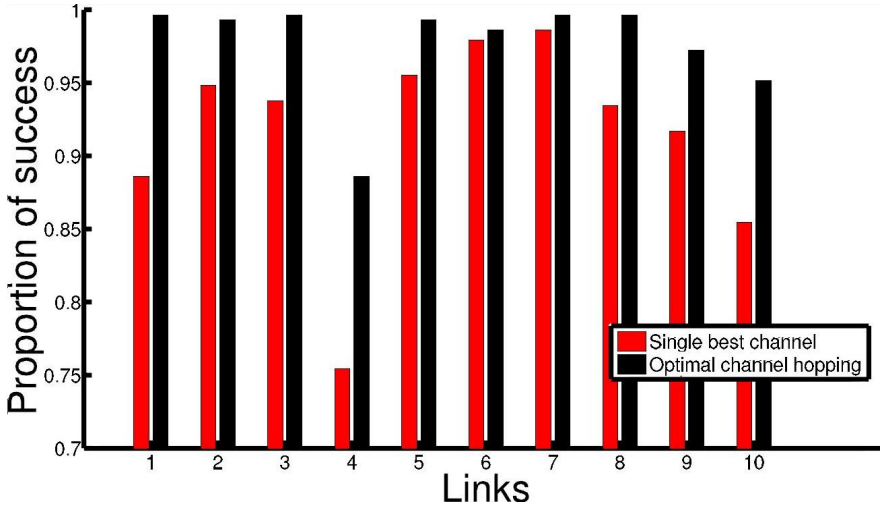
Our analysis above indicates that using a single channel is often not acceptable when long-term reliability must be maintained. Thus, a natural question to ask is whether it is feasible to exploit channel diversity to achieve reliability in situations where single channel assignments are not practical.

To understand the potential for channel hopping, we retrospectively processed our dataset to find the minimum number of channel hops needed to maintain a 90% PRR threshold using a greedy algorithm. We prove the optimality of the algorithm in [106]. Figure 2.11(a) plots the number of channel hops required for 10 links in the dataset, one randomly selected from each apartment. We find that relatively few channel hops are needed to maintain link reliability; in no case is more than 20 hops required per day.

We note that there are periods where none of the 16 channels meet the PRR threshold, and hence no channel hopping occurs during these times. Nevertheless, channel-hopping can significantly reduce the number of link failures compared to picking the single "best" channel (i.e., that with the highest average PRR). Figure 2.11(b) compares the proportion of windows which meet the 90% threshold under two retrospective strategies: an ideal channel-hopping strategy that maintains the PRR threshold with the minimum number of channel hops, and a strategy that fixes each link to its single "best" channel with the highest average PRR. (Note that both strategies make decisions based on the entire data trace retrospectively, and hence cannot be employed at run time; they are chosen here to analyze the *potential* benefit of channel hopping.) In some cases, the improvements achieved by channel hopping are modest. For example, links 6 and 7 only achieve a 0.7% and 1.0% higher success rate




(a) Minimum number of channel hops required; one link randomly selected per apartment.




(b) The proportion of windows where the PRR threshold was met.

Figure 2.11: Retrospective channel-hopping analysis in different apartments.

under channel hopping, largely because their success rates were already high without channel hopping. However, in most cases, we find notable improvements in link success. For example, 6 out of the 10 links experience at least 5% fewer failures with channel hopping than with their single best channel; and links 1 (11.0%) and 4 (13.1%) have substantially higher success rates with channel hopping.

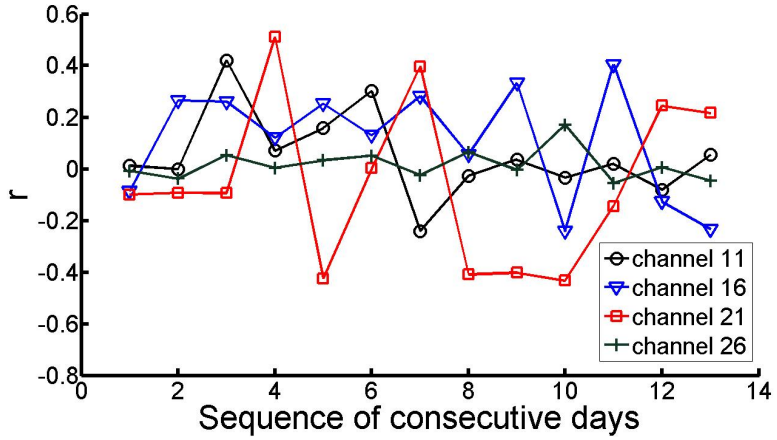
Channel hopping has been proposed in industry standards as a means for improving wireless link reliability, including established standards like Bluetooth's AFH [5] and newer standards such as WirelessHART's TSMP [6] and the forthcoming IEEE 802.15.4e [10]. The results of our analysis confirm that this feature is indeed beneficial for maintaining link reliability in challenging residential environments.

***Observation L4****: Channel hopping is effective in alleviating packet loss due to channel degradation. Occasional channel hopping can effectively maintain reliable communication.*

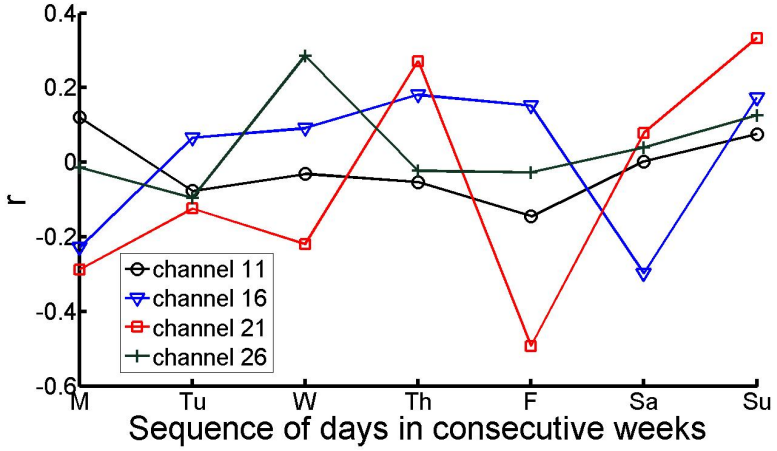
### 2.4.5 Can Hopping be Scheduled Statically?

Because channel quality varies over time, we next explored whether it exhibits cyclic properties (e.g., due to recurrent human activities and schedules). If so, then channel-hopping could be implemented in a lightweight fashion by generating a static channel schedule for each environment. To perform this comparison, we carried out an extended experiment using same setup in one apartment over a period of 14 days. We then calculated the Pearson product-moment correlation coefficient (PMCC) [117], a common measure of dependence between two quantities, as $r$. Intuitively, $r$ values near $-1$ or 1 indicate strong correlation, while values near 0 indicate independence.

Figure 2.12(a) plots $r$ for PRRs calculated at the same times on subsequent days (e.g., 4 PM on Monday vs. 4 PM on Tuesday). Figure 2.12(b) compares the PRR during the same time in consecutive weeks (e.g., 4 PM on Monday vs. 4 PM on the next Monday). $|r|$ is almost always smaller than 0.4, regardless of the channel used; this indicates that there is no obvious correlation between consecutive days or consecutive weeks. Therefore, channel-hopping decisions must be made *dynamically* based on channel conditions observed at runtime.




(a) PMCC of PRRs during the same time on consecutive days.




(b) PMCC of PRRs during the same time in consecutive weeks.

Figure 2.12: The Pearson's product correlation coefficient (PMCC) comparing the PRR at the same time on consecutive days or weeks.

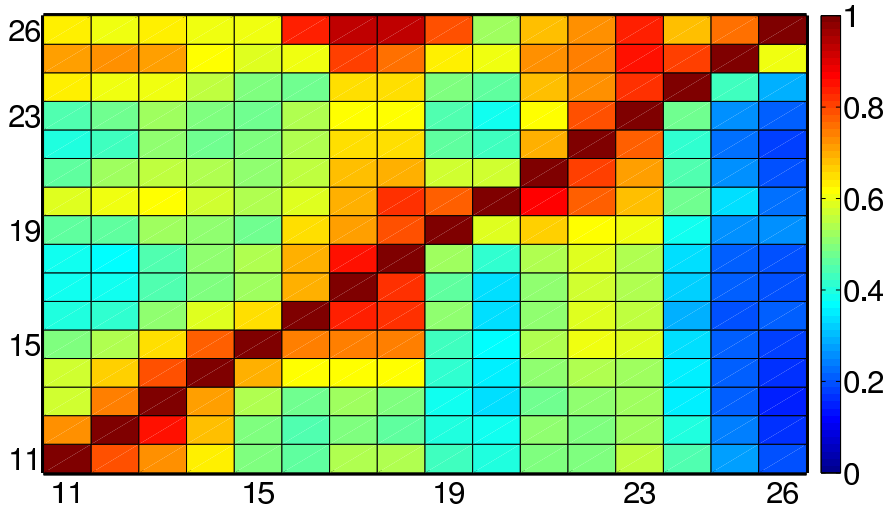



Figure 2.13: Correlation of channel reliability. The X and Y axes indicate channels; the color indicates the probability that channel $x$'s PRR $< 90\%$ when channel $y$'s PRR $< 90\%$.

***Observation L5****: Channel conditions are not cyclic, so channel-hopping decisions must be made dynamically.*

### 2.4.6 How Should New Channels be Selected?

Since channel-hopping must be performed dynamically, it is important to pick a good strategy for selecting new channels when the current channel has degraded beyond use. For the purposes of this analysis, we studied the effect of *channel distance* (the absolute difference between channel indices) on the *conditional probability* of channel failure (the probability that channel $x$ is below the PRR threshold when channel $y$ is also below the threshold).

We observe that not all channels are equally good candidates for channel hopping: from Figure 2.13, we can see that performance is strongly correlated across adjacent channels. For instance, when channel 20 has poor PRR ($< 90\%$), there is a probability greater than 76.8% that channels 18, 19, 21, and 22 also suffer from poor PRR. In Figure 2.14, we plot the conditional probability of link failure as a function of channel distance. We observe that this probability can be as high as 70% between neighboring channels and 60% between every other channel, but drops off as channel distance increases. When facing a failing channel, a probabilistic approach on new channel selection should be used to avoid jamming the

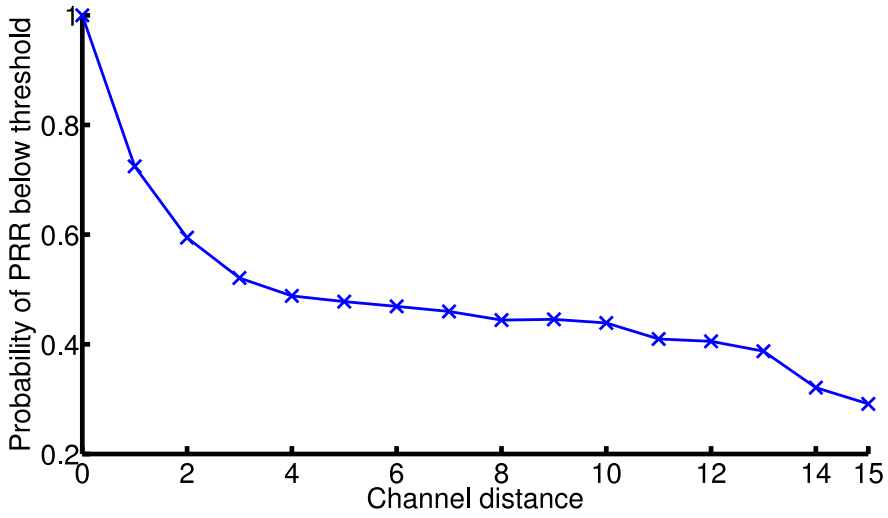



Figure 2.14: Correlation of channel reliability as a function of channel distance.

new channel. Designing a channel selection algorithm is out of the scope of this chapter. The focus of this chapter is on the empirical studies that provide ground truth and insights for designing and managing HANs. We have since developed a practical channel selection scheme [103] based on the findings presented in this chapter.

***Observation L6****: Reliability is strongly correlated among adjacent channels; a device should probabilistically select a new channel that is at least three channels away from the failing channel.*

### 2.4.7 How effective is increasing transmission power for improving link reliability?

As an orthogonal approach to channel hopping, transmission power control [73] [44] aims to maintain link quality by dynamically adjusting transmission power. We evaluate transmission power control's potential for maintaining channel reliability through a microbenchmark experiment. For this evaluation, we repeat the same experimental setup used in the previous experiments, except using multiple transmission powers. Specifically, the transmitting node was configured to send 100 consecutive packets at a given transmission power; this was repeated over 29 of the CC2420's 31 distinct power settings in a round-robin fashion.

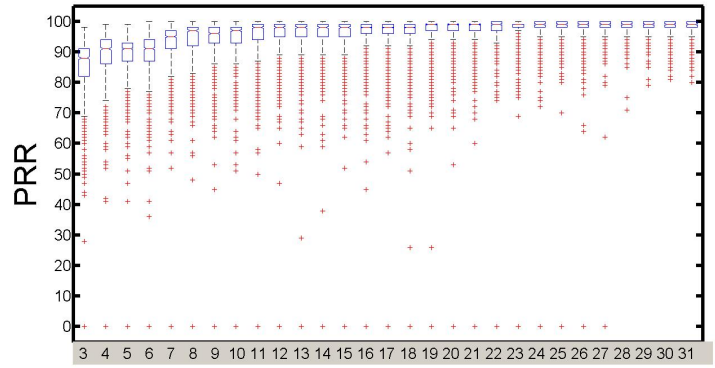
(The two lowest power settings were excluded from this experiment, as the manufacturer has indicated that the CC2420's output power is unstable at these settings [3].)

Figure 2.15 plots the PRR on three different channels in one apartment; results for other apartments and other channels are similar but omitted for space. We observe that adjusting transmission power can indeed be effective at improving link quality. Figure 2.15(b) presents the PRR from the worst channel (18): on this channel, the median PRR increases from 68% to 91% when the transmission power level increases from 4 to 11, and further increases to 95% at the maximum transmission power (level 31). Nevertheless, the impact of switching channels may be even more pronounced, as seen by comparing Figure 2.15(a) through 2.15(c). By changing to channel 26, a link on channel 11 or 18 could have achieved a comparable increase in PRR while remaining at power level 3. Moreover, switching channels can be significantly less expensive than increasing transmission power: for example, on the CC2420, increasing the transmission power can increase the radio's current consumption from as low as 8.5 mA to as high as 17.4 mA [3]. Hence, leveraging channel diversity *in conjunction* with transmission power control can potentially result in significant energy savings.

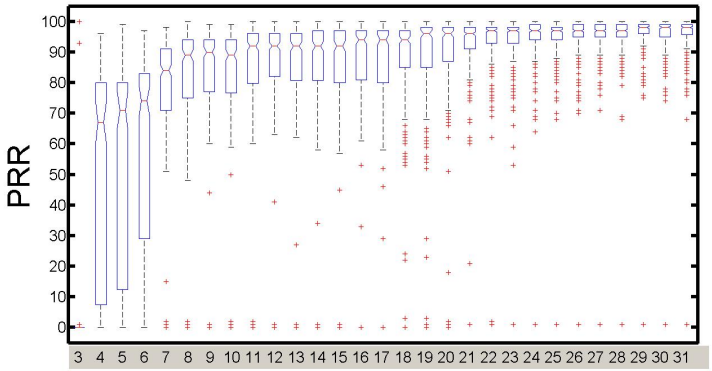
***Observation L7****: Increasing transmission power may be effective for maintaining channel reliability, but is potentially expensive. Combining channel diversity with transmission power control is a promising strategy for controlling energy consumption while maintaining network reliability.*

## 2.5 Conclusion

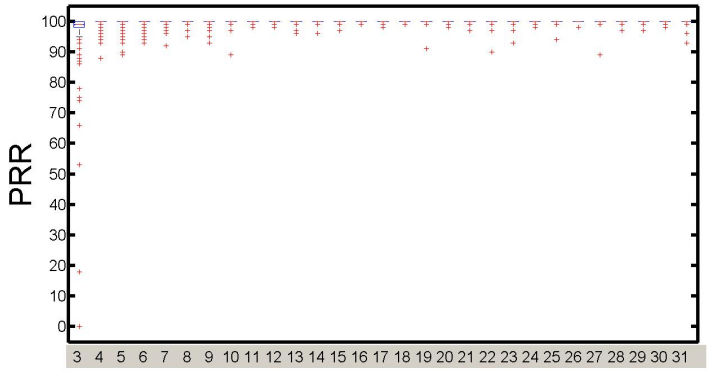
HANs based on wireless sensor network technology represent a promising communication platform for emerging home automation applications such as smart energy. These emerging applications often impose stringent wireless communication requirements in terms of network reliability, which are made challenging by the complex and highly variable wireless environments in typical residential environments. This chapter presents an empirical study on the performance of HANs in real-life apartments, looking both at passive spectrum analysis traces and an active probing link study. The observations made in our study highlight the significant challenges that face HAN applications for achieving reliable wireless communication in residential settings. Nevertheless, our observations also suggest that these challenges




(a) Channel 11




(b) Channel 18




(c) Channel 26

Figure 2.15: Box plot of the PRR of a link over 29 different transmission power levels.

may be tamed through the judicious use of channel diversity. Specifically, we may distill our findings into set of key design guidelines for developing reliable HANs:

1. Channel selection can have a profound impact on HAN reliability. Channel selection cannot be simply relegated a static channel assignment, whether made at the factory or at deployment time. (S1, L1, L2)
2. Retransmissions alone cannot always compensate for a poor-quality channel. (L3)
3. Short time channel assessment is effective in estimating channel condition, since larger time window of measurement cannot bring more benefit. (S3)
4. Although Wi-Fi is a major source of channel usage, other wireless technologies may also contribute significantly to channel usage. Solutions which target a single interfering technology are not always sufficient in residential environments. (S4)
5. Reliable communication can be maintained through occasional channel hopping. (L4)
6. Channel hopping cannot be performed based on a static, cyclic schedule. (L5) Instead, channel-hopping decisions should be made dynamically based on conditions observed at runtime. (S2, L2)
7. A device should probabilistically select a new channel that is at least three channels away from the failing channel. (L6)
8. Increasing transmission power may be effective for maintaining channel reliability, but is potentially expensive. Combining channel diversity with transmission power control is a promising strategy for controlling energy consumption while maintaining network reliability. (L7)

We believe that our findings and insights will provide general design guidelines and impact the development of HANs that are gaining increasing importance with the emergence of smart energy as the "killer app" for wireless sensor networks.

# Chapter 3

# ARCH: Adaptive and Robust Channel Hopping for Home-Area Sensor Networks

## 3.1 Introduction

Home automation technologies aim to provide households with a high degree of control and monitoring of common household devices. Such technologies form an integral part of emerging "smart home" applications ranging from energy metering to assisted living. Low-power wireless sensor networks (WSNs) represent an attractive technology for retrofitting existing residences with home automation applications. Such home area network (HAN) devices can be operated without the need for fixed power or communication infrastructure, alleviating the need to install a wired power and communication infrastructure. However, the very fact that these HANs do not depend on a fixed infrastructure also poses key deployment challenges. Unlike traditional wired technologies, HANs depend on low-power wireless communication susceptible to interference in the free 2.4 GHz ISM band.

We recently performed an in-depth empirical study into the reliability of HANs in real-world apartment buildings [106]. Our study demonstrated the need for dynamic channel hopping in maintaining reliable links: in an apartment, there is usually no single channel which is persistently reliable for 24 hours at a time. Moreover, we observed that many individual links suffered long-lived disconnections on a particular channel; these bursty losses meant that retransmissions alone were insufficient to maintain a target link quality. We also found that

channel reliability does not exhibit cyclic behavior, requiring that channel-hopping decisions be made based on conditions observed at runtime. Nevertheless, switching channels even a few times at runtime could effectively maintain reliable communication.

In this chapter, we draw on these insights to develop the *Adaptive and Robust Channel Hopping* (ARCH) protocol. ARCH enhances network reliability through *channel diversity*: devices opportunistically change their radio's frequency in order to avoid adverse channel conditions such as interference and environmental noise. ARCH has the following salient features that distinguish it from existing channel diversity schemes:

1. ARCH *adaptively* selects channels based on runtime conditions, hopping channels only when channel conditions have degraded. ARCH achieves consistent reliability on existing 802.15.4 radio hardware with minimal channel-switching overhead.
2. ARCH is *distributed* and selects channels on a per-link basis, rather than synchronizing hops across the entire network. Hence, ARCH's coordination policy is simple, and nodes can avoid localized phenomena.
3. ARCH is *lightweight and robust*, allowing it to be feasibly implemented on constrained WSN hardware.
4. ARCH also introduces *minimal communication overhead*, leveraging existing packet acknowledgments when available.

We evaluate our approach through trace-driven simulations and through real deployment in real-life apartments with residents' daily activity. Our results in a single-hop scenario demonstrate that ARCH can reduce the number of packet retransmissions by a median of 42.3% compared to using a single, fixed wireless channel, and can enable up to a 2.2× improvement in delivery rate on the most unreliable links in our experiment. Under a multi-hop scenario, ARCH achieved an average 31.6% reduction in radio usage, by reducing the ETX of each link by up to 83.6%. Due to ARCH's lightweight reactive design, this improvement in reliability is achieved with an average of 10 or fewer channel hops per link per day. ARCH's lightweight design also allows it to be reasonably deployed even on constrained WSN hardware: adding ARCH to a multi-hop data collection application introduced an overhead of only 480 bytes of program ROM and 26 bytes of RAM.

The rest of the chapter is organized as follows. Section 3.2 reviews related work. Section 3.3 discusses the design of ARCH channel-hopping protocol. Section 3.4 presents a series of simulator-driven and real-world experiments which illustrate ARCH's efficiency in alleviating packet loss due to poor channel conditions. Finally, we conclude in Section 3.5.

## 3.2 Related Work

In recent years, there has been increasing interest in using channel hopping to enhance MAC layer performance. SSCH [15] aims to improve network capacity by using channel hopping to prevent interference among simultaneous transmissions. [83] proposes a rapid channel hopping scheme to protect from jamming attacks in the 802.11a band. Other multi-channel protocols [62, 67, 68, 124, 126, 133] have been proposed for WSNs with their limited resources in mind. Our work is distinguished from these protocols in two key ways. First, these protocols focus on enhancing throughput, while our own work aims for enhanced reliability. Second, these works deal primarily with in-network interference, while ARCH is designed to avoid external sources of interference and other environmental noise. These differences in design goals reflect the specific requirements of typical HAN applications: real-life HANs typically feature applications with low data rate requirements, but are subject to strong external interference and environmental impacts.

Hauer et al. [47] discusses a multi-channel measurement of Body Area Networks (BANs). Hauer's study features controlled indoor experiments along with outdoor experiments carried out during normal urban activity, and concludes that channel hopping schemes may use noise-floor measurements to effectively detect and mitigate the effects of interference. In contrast, ARCH's design is based on empirical study of the multi-channel properties in residential environments, which often exhibit highly complex behavior. Accordingly, ARCH evaluates channel conditions using direct Estimated Transmission Count (ETX) measurements.

Several industry standards, such as WirelessHART's TSMP [6], Bluetooth's AFH [5], and ZigBee 2007's optional frequency agility [134], leverage channel diversity to improve link reliability. While both TSMP and our approach are based on the 802.15.4 standard, ARCH employs a simpler reactive channel-hopping mechanism in contrast to TSMP's automatic pseudorandom channel-hopping scheme. Because WirelessHART is targeted to industrial

applications with stringent reliability requirements (e.g., safety-critical monitoring and control systems), it uses sophisticated TDMA techniques and a complex centralized network controller to ensure channel reliability even in harsh environments. ARCH's relative simplicity makes it a more cost-effective and easier-to-deploy solution for home automation applications, where reliability requirements are less stringent. Bluetooth, particularly the emerging low-power Bluetooth standard, represents another potential approach to HANs; like TSMP, Bluetooth's AFH avoids persistent interference by constantly hopping pseudo-randomly across channels. ARCH serves as an alternative approach based on the 802.15.4 standard, where radio chips are typically not designed to accommodate AFH's aggressive channel-hopping schedules. ZigBee 2007's frequency agility uses a centralized channel manager node to synchronize channel usage across the whole network; based on the results of our empirical study, ARCH instead selects channels on a per-link basis in order to avoid localized effects of environmental noise. Moreover, ZigBee 2007 explicitly leaves key portions of the frequency agility scheme (such as the mechanism for selecting new channels) unspecified, providing only general suggestions for how these components could be implemented. In contrast, ARCH represents a complete instantiation of a practical, lightweight channel hopping algorithm.

## 3.3 Protocol Design

In this section, we present the design of our ARCH protocol. ARCH is designed based on the key observations in our empirical study and has the following salient features. First, ARCH is an *adaptive* protocol that channel-hops based on changes in channel quality (specifically, ETX) observed in real time. We use ETX rather than RSSI/LQI to indicate link quality because RSSI/LQI are not sufficiently robust in complex indoor environments [44]. Second, ARCH is a *distributed* protocol that hops channels on a per-link basis, based on the observation that channel conditions can vary greatly from link to link even within the same network. Third, ARCH is designed to be *robust and lightweight*. ARCH uses a practical hand-shaking approach to handle channel desynchronization and an efficient sliding-window scheme to predict channel deterioration, and can be reasonably implemented on memory-constrained wireless sensor platforms. Fourth, ARCH introduces *minimal communication overhead* for applications where packet acknowledgements are already enabled.

We will begin by discussing the design insights based on the key findings in our previous empirical study, and then present the ARCH algorithm in outline. We will then describe several important subcomponents of ARCH — channel condition estimation, opportunistic channel selection, and coordination across nodes — in more detail. Finally, we will discuss mechanisms in ARCH for detecting and handling channel desynchronization errors.

### 3.3.1 Design Insights

To investigate the multi-channel wireless properties of residential environments, we carried out a series of experiments in ten real-world apartments constructed by different housing companies. Several key insights derived from our study formed the basis for ARCH's design; we summarize the relevant findings in this subsection. More details on the study may be found in [106].

#### Is Channel Diversity Effective?

Our study found that there was usually no single channel which was persistently reliable for 24 hours at a time. Hence, we considered channel diversity as a means to achieve long-term link reliability. After retrospectively analyzing our experimental traces to find an optimal channel hopping schedule, we found that relatively few channel hops are needed to maintain a target link quality. Only 5, 8, and 36 hops were needed per day to meet target packet reception rate (PRR) thresholds of 80%, 90%, and 95%, respectively.

***Insight 1****: Link reliability can be achieved through relatively occasional channel hopping.*

#### Can Hopping be Scheduled Statically?

If channel quality exhibits cyclic properties, then channel-hopping could be implemented in a lightweight fashion by generating a static channel schedule for each environment. However, our study found no obvious cyclic, predictable schedule of interference patterns.

***Insight 2**: Channel-hopping decisions must be made dynamically based on channel conditions observed at runtime.*

**How Should New Channels be Selected?**

Since channel-hopping must be performed dynamically, it is important to pick a good strategy for selecting new channels when the current channel has degraded beyond use. Our analysis found that channel quality is often correlated among spatially nearby channels. Hence, channel selection should favor new channels which are further away from the current channel.

***Insight 3**: It is more beneficial to switch to a further-away channel when the current channel degrades beyond use.*

### 3.3.2 ARCH Protocol Outline

Based on the above findings, we now outline our design for ARCH. ARCH is a receiver-oriented protocol; i.e., receivers select the communication channel for all incoming links, and senders switch to the recipient's channel when they wish to transmit a packet. Each link is initially set to use some predefined $Default\ Channel$ out of a provided $Channel\ Pool$. This pool specifies the channels which the application is allowed to use; this could be selected at design time to include all 16 channels or some subset (e.g., 4 orthogonal channels).

As a packet arrives, the channel's reliability (represented as ETX) is updated, as discussed in more detail in Section 3.3.3. When the ETX exceeds a specified $ETX\ Threshold$, the receiver node will select a new channel from the channel pool (see Section 3.3.4) and initiate a channel hop. The receiver then notifies all of its senders of this channel hop using the mechanism discussed later in Section 3.3.5.

To avoid the bursty packet loss observed in [106], ARCH blacklists bad channels so that they will not be used again for at least a short time period. ARCH ensures that enough candidate channels are available by un-blacklisting the entire channel pool when the number of candidate channels drops below a specified $Standby\ Channel\ Threshold$.

### 3.3.3 Channel Estimation

Insights 1 and 2 highlight the importance of reactively hopping channels based on runtime channel quality data. A key component of this approach is an agile *channel estimation* scheme that can quickly and accurately detect channel degradation at runtime. Estimating the reliability of a wireless link or channel is a challenging topic which has garnered significant interest in the research community. One common quality metric is ETX, which represents link quality as the number of (re)transmissions required for a successful reception. ETX is particularly compelling for home automation applications because it can be estimated from sequence numbers embedded in existing packets. Thus, there is no need for expensive active probing.

We note that ARCH does *not* perform a moving average over multiple ETX values, as in e.g. TinyOS's four-bit link estimator [36]. Instead, ARCH maintains a sliding window of ETX values for the last $m$ packets; a channel is predicted to be unreliable if all $m$ ETX values exceed some threshold value. Our trace study in Section 3.4.1 demonstrates that this approach can predict channel reliability with sufficient accuracy using as little as 15 minutes' worth of history.

### 3.3.4 Opportunistic Channel Selection

As noted in Insight 3, link quality is often strongly correlated among adjacent channels. Hence, using a fixed channel hopping sequence is therefore neither safe nor robust: we wish to avoid channels which are spatially close to the current, poor-quality channel. Likewise, we do not wish to continuously monitor all channels in order to support channel selection decisions; while effective, this would incur unreasonable overhead.

Instead, ARCH uses a probabilistic scheme to select new channels. When hopping channels, ARCH generates a random number $q \in [0, k]$ for each non-blacklisted channel in the *Channel Pool*, starting from the furthest-away channel to the closest. If $q$ falls into the range $[0, c_i k]$ ($c_i < 1$), then channel $i$ will be selected. $c_i$ is weighted according to the spectral distance away from the currently-used channel: the larger the distance, the more likely that a channel is selected.

### 3.3.5 Coordinated Channel Hopping

In [106], we observed that channel quality may vary significantly even within a network. Hence, ARCH does not hop channels in lockstep across the entire network. Instead, ARCH uses a receiver-oriented approach to channel selection. In effect, each node specifies which channel they wish to use to receive packets. Whenever a node detects channel degradation on its incoming links, it selects a new channel using the policy described in Section 3.3.4.

A simple coordination policy allows ARCH to transparently support both single-hop and multi-hop routing. Upon selecting a new channel, nodes notify their neighbors of this change (using a mechanism described below), who then record this information in their neighbor tables. Nodes stay on their own (receiving) channel as often as possible. When a node transmits data, it temporarily switches channels to match its recipient, then switches back after waiting long enough to receive an ACK. Thus, the node can continue to receive packets from other nodes further upstream; the only times a node leaves it own channel is when it transmits data downstream, when it could not have received data anyway.

Two strategies exist to notify neighbors of channel hops. First, a node may notify its upstream neighbors one-by-one. In the interest of minimizing overhead, this notification may be embedded in ACK packets the next time the node receives a packet from an upstream neighbor. Second, the node may broadcast an explicit channel hopping message to all neighbors in range. The first approach introduces the lowest overhead, but may delay the channel hop for excessively long periods of time and cannot handle situations where the node has not yet discovered a neighbor. The second approach requires an additional control packet and may not work for asymmetric links (since broadcasts are unreliable), but allows a node to coordinate with undiscovered neighbors.

Based on these tradeoffs, ARCH implements a hybrid policy which combines the two forms of notification, shown in Algorithm 1. Additional measures are employed to handle coordination failures, as described in the next subsection.

Because a node's neighbors may reside on different channels, broadcast traffic patterns may be handled by transmitting unicast packets to each of a node's neighbors on their respective

**Algorithm 1**

```
1: if received data packet then
2:   if detected channel degradation and Pending = 0 then
3:     Select a new channel;
4:     if only has one neighbor then
5:       Send back ACK with immediate channel hop;
6:       Set new receiving channel.
7:     else
8:       Send back ACK with pending channel hop;
9:       Set Pending = 1;
10:      Save new receiving channel.
11:    end if
12:  else if Pending = 1 then
13:    if the sender is its last neighbor then
14:      Send back ACK with channel hopping decision;
15:      Set new receiving channel and Pending = 0;
16:    else
17:      Send back ACK with pending channel hop;
18:    end if
19:  else
20:    Send back ACK;
21:  end if
22: end if
23: if received immediate channel hop then
24:   Set new sending channel and set flag = 1;
25: end if
26: if received pending channel hop then
27:   Save new sending channel and set flag = 0;
28: end if
29: if transmitting packet then
30:   if flag = 1 then
31:     Send packet using recipient's channel and wait a small time for ACK;
32:   else
33:     Send packet using recipient's new channel and wait a small time for ACK;
34:     if ack received then
35:       Set flag = 1;
36:     else
37:       Resend packet using recipient's last channel and wait a small time for ACK;
38:     end if
39:   end if
40: end if
```

channels. While this approach introduces some overhead compared to a single-channel protocol, we expect this overhead to be low in practice: by its nature, HAN traffic will often be dominated by (unicast) data collection and actuation packets.

### 3.3.6 Handling Channel Desynchronization

When channel conditions degrade, reliability may drop so far that the coordination messages described above are lost. Under this situation, a node and one or more of its senders may become desynchronized. ARCH uses two thresholds to detect these conditions: $T_1$ on the receiver side, which denotes the maximum waiting time between two packets; and $N$ on the sender side, which denotes the maximum number of allowed packet retransmissions. $T_1$ and $N$ are selected so that the receiver's timeout is longer than the sender's timeout, for reasons discussed below.

Based on these thresholds, ARCH uses the following procedure to detect and handle desynchronization. Let $t$ denote the time since the receiver received its last packet and $n$ denote the number of times the sender has retransmitted the current packet. When either threshold is exceeded ($t > T_1$ or $n > N$), the node reverts to the default channel[3]. Because the receiver has the longer timeout, the sender will already have reverted to the default channel by the time the receiver arrives. The receiver may then initiate resynchronization with the sender.

A subtle complication is that desynchronization may be falsely detected. It is possible that the two nodes indeed switched to the same channel; however, this new channel was too noisy for communication, and hence the nodes falsely believed that they were desynchronized. Thus, ARCH has a policy that nodes exchange their previous channels when resynchronizing. If the channels do not match, then there was indeed a channel synchronization problem, and the nodes proceed to resynchronize on the receiver's previously-selected channel. However, if the channels match, then the nodes did successfully resynchronize on the new channel but were unable to communicate. In this case, the receiver selects an entirely new channel (since the previous channel was too unreliable) and repeats the channel-hopping procedure.

[3]If multiple default channels are specified, the node reverts to the channel spatially furthest from its last successful synchronization.

A salient feature of this scheme is that it provides an upper bound on disconnection time. This feature is important to home automation applications where, for example, extended disconnections in a thermal stack could cause a room to reach uncomfortable temperatures.

## 3.4 Evaluation

To validate the efficiency of ARCH in alleviating packet loss through channel-hopping, we performed a series of simulation-driven and real-world experiments. First, we carried out two *simulator-based microbenchmarks* driven by data traces collected in ten different apartments. These microbenchmarks measure the efficacy of ARCH's opportunistic channel selection scheme and ETX-based link estimator, respectively. We then measured ARCH's performance through a series of *real-world macrobenchmarks*. For these experiments, we deployed an implementation of ARCH on top of the TinyOS 2.1 operating system [4], and measured its performance in real-world apartments under various application scenarios. These scenarios range in complexity from a basic always-on, single-hop network to a multi-hop network deployed with a low-power listening MAC layer.

### 3.4.1 Simulator-Based Microbenchmarks

Our microbenchmark experiments were carried out in a C++ simulator environment, and are driven by two sets of link quality data previously collected for [106]. Both data sets were collected in 10 real-world apartments using networks of Tmote Sky and TelosB [88] motes. A single transmitter node deployed in each apartment broadcast packets to multiple recipient nodes, which recorded the sequence number of all successfully decoded packets. Every 5 minutes, the transmitter node cycled over each of the 16 channels defined by the IEEE 802.15.4 standard. This process repeated for 24 hours in each apartment during the residents' normal activity.

For the first data set, the transmitter used a data rate of 100 packets per channel every 5 minutes and no packet retransmissions. This provided high-granularity PRR data, which we used to evaluate ARCH's channel selection policy. For the second data set, the transmitter

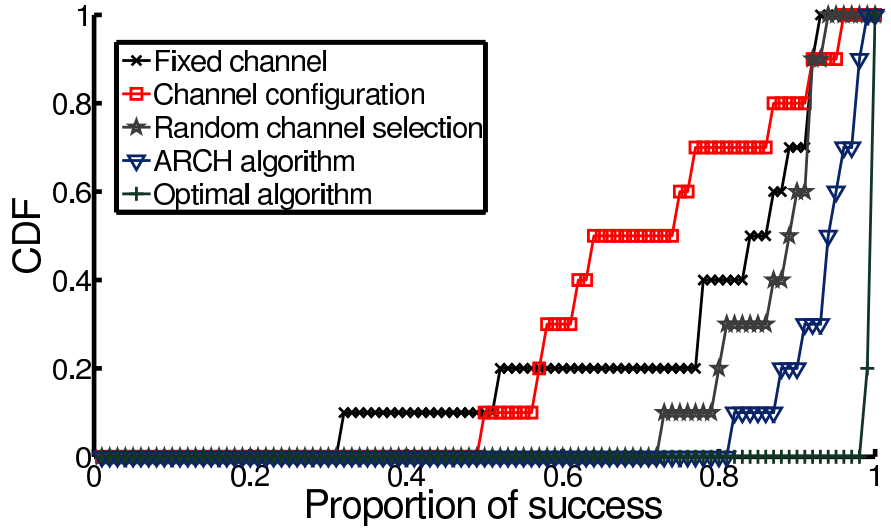
used a reduced data rate of 1 packet per channel every 5 minutes with packet retransmissions. This data set gave us a direct measurement of ETX, which we used to validate ARCH's link estimation scheme. We will now describe both microbenchmarks in detail.

### Channel Selection Scheme

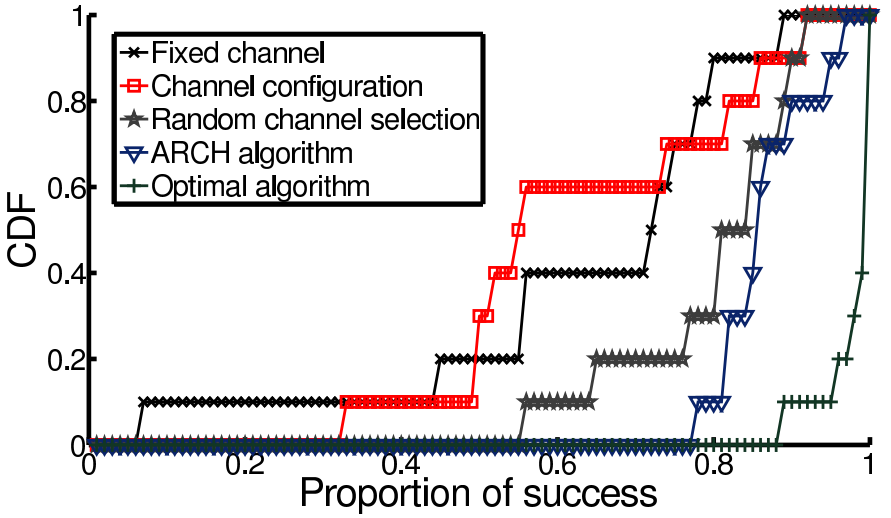
Our first group of simulations isolates the performance of ARCH's opportunistic channel selection scheme by comparing ARCH against two widely-used channel diversity schemes. First, the *fixed channel* scheme uses the default channel of 15 (which had the highest average PRR of all links in our data traces) for all links in all apartments. Second, the *channel configuration* scheme selects the channel with the best performance during the first 30 minutes of the empirical study (emulating a protocol which collects extensive link quality while bootstrapping). To further isolate the performance of ARCH's channel selection scheme from its channel estimation routines, we also performed a series of experiments using a *random channel-hopping* variant of ARCH. This variant detects channel degradation in the same way as the unmodified ARCH, but responds to degradation by hopping to random channels. Finally, we compare ARCH against an *optimal channel-hopping* algorithm. This optimal scheme retroactively processes the entire dataset to find the best possible channel hopping decisions. By its nature, the optimal scheme provides an upper bound on performance, but cannot actually be implemented online.

The simulations were configured as follows. ARCH's *Channel Pool* was set to use all 16 channels, with the default channel set to 15. For the probabilistic channel selection, we set $k = 1$ and selected $c_i$ to be the difference between the two channels' numbers divided by 100. We conducted two sets of experiments with different PRR thresholds: 80% and 90%. To rule out the effects of the channel estimator, we replaced ARCH's ETX estimator with ground-truth PRR data over 5 minute windows.

Figure 3.1(a) plots the CDF of the nodes' success, defined as the proportion of time that the node met the PRR threshold of 80%. On average, ARCH achieves 18% higher success than the fixed channel and channel configuration schemes. In addition, ARCH's channel selection and blacklisting schemes allow it to improve on the random channel selection scheme by




(a) CDF of proportion of time an 80% PRR threshold was met.




(b) CDF of proportion of time a 90% PRR threshold was met.

Figure 3.1: A comparison of node success (PRR > threshold) under various channel selection schemes. Results were measured under simulation using experimentally-collected PRR traces from 10 apartments.

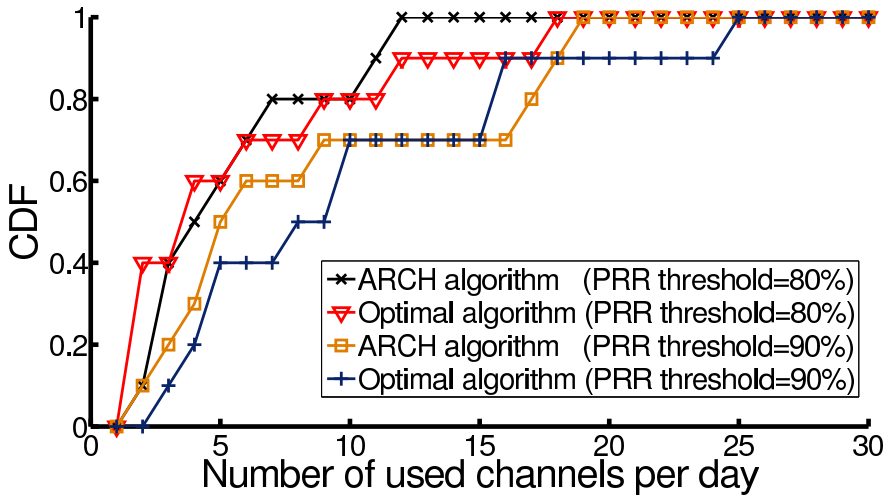



Figure 3.2: CDF of number of channel hops per day under ARCH and optimal channel selection schemes. Results were measured under simulation using experimentally-collected PRR traces from 10 apartments.

9%. Indeed, we note that ARCH comes within 6% of upper bound provided by the optimal scheme.

Increasing the PRR threshold to 90% provides similar results, as shown in Figure 3.1(b). Under ARCH, the links have a median success rate of 88%; in contrast, under the fixed channel and channel configuration schemes, the median success rates are 72% and 56%, respectively. Again, ARCH improves on the random channel selection scheme by 8%, coming within 12% of the optimal scheme's upper bound.

As shown in Figure 3.2, ARCH achieves this degree of reliability with relatively few channel hops. At most 25 channel switches are needed per link per day to meet the 90% PRR requirement, with a median of fewer than 10.

### Channel Quality Estimator

Next, we wished to explore the ability of ARCH's channel estimator to accurately predict long-term channel conditions. For these experiments, we used the second set of data traces, which were collected at a reduced data rate of 1 packet/5 minutes and retransmissions enabled. We processed these traces through ARCH's estimator to produce a series of binary

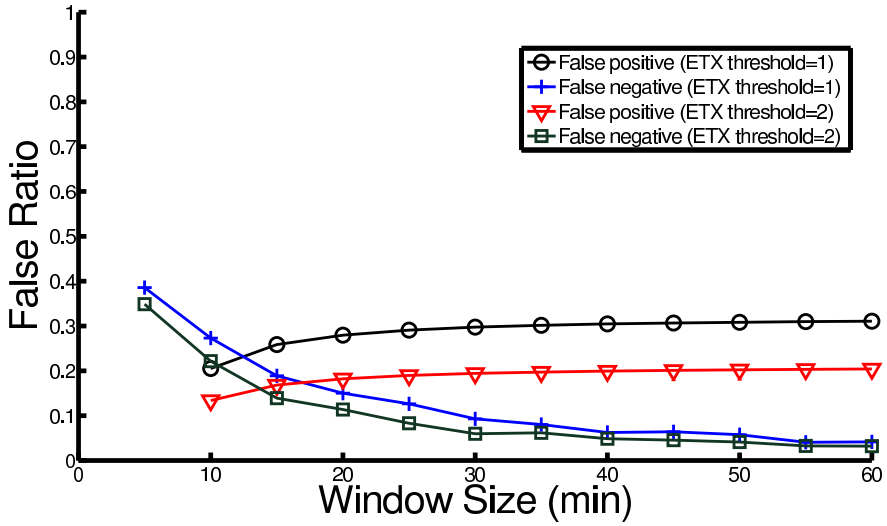
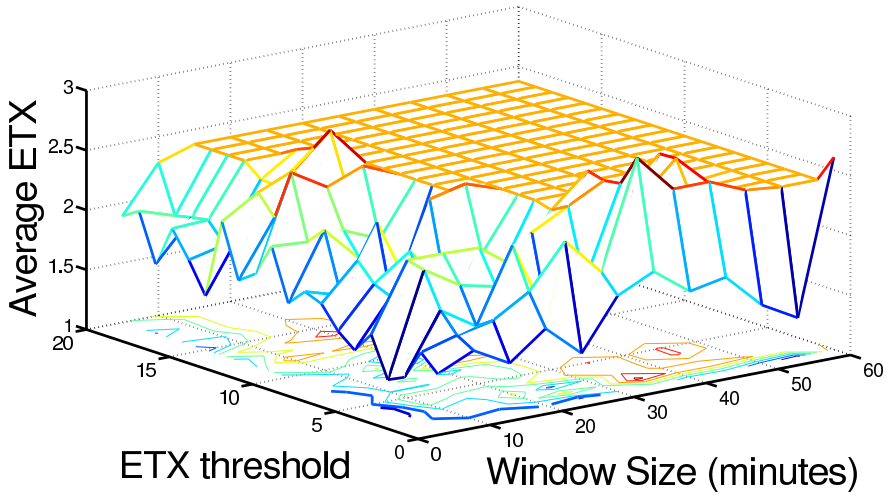



Figure 3.3: False-positive and false-negative rates of ARCH's channel estimation scheme. Results were measured under simulation using experimentally-collected ETX traces from 10 apartments.

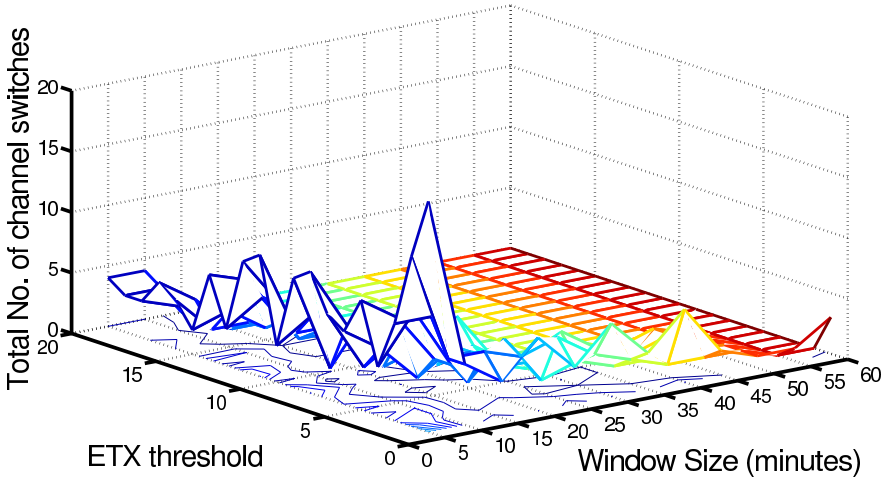
channel quality predictions. Specifically, given a sliding window of $m$ minutes' worth of traces, the estimator produced a series of binary decisions indicating whether the channel has failed. We then generated a series of ground-truth data by comparing the ETX traces against numerous fixed thresholds.

Figure 3.3 compares ARCH's predictions for channel reliability against the ground truth data. Specifically, the figure plots the false positive (i.e., channel failure predicted when no failure actually occurred) and false negative (i.e., no channel failure predicted when the channel had actually failed) rates with various ETX thresholds and window sizes. We observe that an ETX threshold of 2 and window size of 15 minutes (i.e., 3 packets) achieves false positive and negative rates below 20%.

Figure 3.4 confirms that these parameters are ideal, even over a wider range of thresholds and window sizes. A threshold of 2 and window size of 15 minutes achieved the lowest ETX (an average of 1.66 transmissions) and total channel switches (5). For comparison, a fixed-channel scheme run over the same data trace produced an average ETX of 2.38 transmissions.




(a) Average ETX with different ETX thresholds and window sizes.




(b) Number of channel switches with different ETX thresholds and window sizes.

Figure 3.4: The effect of various thresholds and window sizes on the performance of ARCH's channel estimation scheme. Results were measured under simulation using experimentally-collected ETX traces from 10 apartments.

| | ROM (bytes) | RAM (bytes) |
|---|---|---|
| Fixed channel | 23776 | 1218 |
| ARCH | 24256 | 1244 |

Table 3.1: ROM and RAM usage comparison of our multi-hop macrobenchmark application.

### 3.4.2 Real-World Macrobenchmarks

To evaluate ARCH's real-world performance, we performed a series of data collection macrobenchmarks in real-world apartments. The experimental setup is similar to the experiments used to gather the data traces above. However, rather than collecting data on all channels, we deployed a full TinyOS implementation of ARCH below our application logic and allowed ARCH to automatically select the channel usage.

Owing to its lightweight design, ARCH introduces very little code size overhead. Table 5.1 shows the program ROM and RAM usage for the benchmark application described in Section 3.4.2, as reported by the TinyOS toolchain. Compiling the application with ARCH enabled only consumes 480 extra bytes of ROM and 26 bytes of RAM compared to the same application compiled with a fixed channel assignment. (For comparison, the MSP430F1611 MCU used by the TelosB and Tmote Sky motes provides 48 kilobytes of flash ROM and 10 kilobytes of RAM.)

Per the previous simulation results, we configured ARCH's channel quality predictor to use an ETX threshold of 2 and window size of 3 packets. As with the second set of simulations, we use a data rate of 1 packet/5 minutes, and enabled packet retransmissions. This configuration emulates the behavior of typical HAN applications, which feature relatively low data rates but require reliable data delivery. For example, a wireless HVAC system typically requires 1 valid temperature reading from each sensor every 5 minutes to maintain a comfortable temperature level. Each experimental run lasted 24 hours.

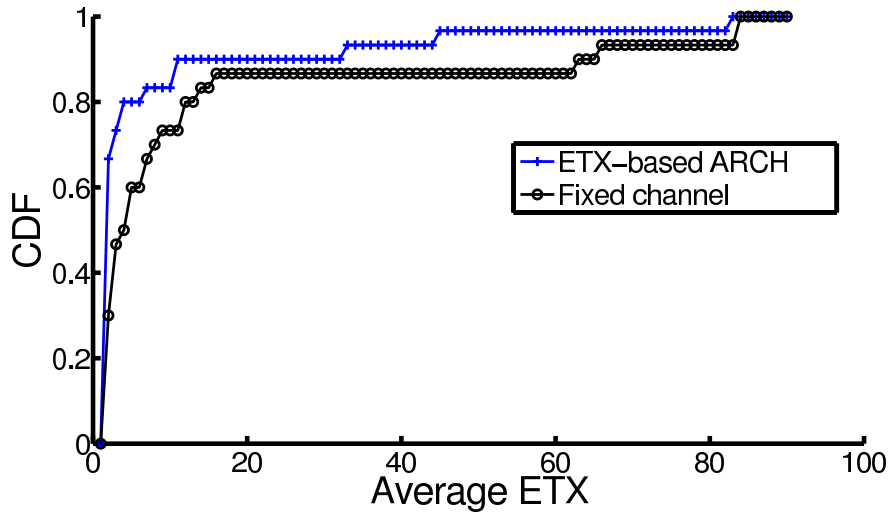
### Single-Hop Data Collection

Our first experiment measured ARCH's performance under a simple single-hop data collection application. In this scenario, one node was designated as a data sink. The remaining nodes produced packets at a rate of 1 packet/5 minutes and attempted to deliver the packet directly to the sink. For the purposes of this experiment, we used TinyOS's default CSMA/CA MAC layer (i.e., no duty cycling). As a baseline, we also carried out the experiment with every node fixed to channel 26.

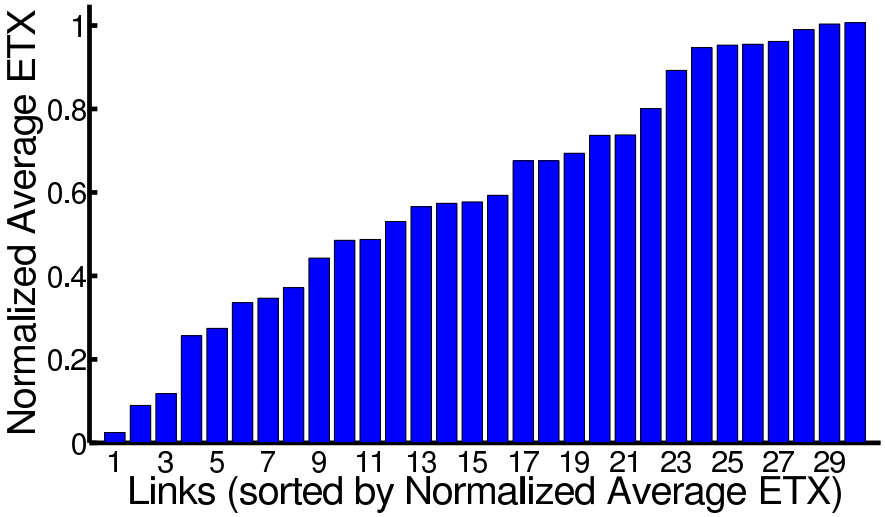
Figure 3.5(a) plots the CDF of the average ETX for each of the experiment's 30 links. ARCH reduces the ETX by a median of 42.3% compared to a fixed-channel scheme. Figure 3.5(b) breaks down ARCH's improvements on a per-link basis. In many cases, the improvements are quite notable; ARCH reduced the transmissions by more than half for 11 of the 30 links, and in one extreme case reduced transmissions by 97.5%. Even in the worst case, ARCH performs comparably with the fixed-channel scheme, with a slight ETX increase of 0.7%.

Figure 3.6(a) compares the delivery rate (i.e., the proportion of packets successfully delivered after any number of retransmissions) of ARCH and the fixed-channel scheme. While ARCH does not achieve 100% delivery under all links, it does so for 26 of the 30 links. In comparison, the fixed-channel scheme achieves a 100% delivery rate for only 21 links. ARCH also achieves a much higher minimum delivery rate (54.2% vs. 17.0%) than the fixed-channel scheme.

Figure 3.6(b) illustrates the number of channel desynchronizations detected for each corresponding link in Figure 3.6(a). (The links are sorted in the same order as in Figure 3.6(a) so that a direct comparison may be made.) Although some of the links experience many desynchronization events, ARCH is still able to maintain a high delivery rate. For example, link 4 experienced over 100 desynchronizations during the 24-hour experimental run, but nevertheless achieved a delivery rate of close to 100%. This indicates that ARCH's desynchronization-handling mechanism, as described in Section 3.3.6, is indeed effective at resolving these events. We note an outlier in link 1, which desynchronized more than 200 times throughout the experiment and achieved a delivery rate of only 54.2%. These statistics reflect the fact that the link was under such harsh, persistent interference that the recipient struggled to locate a single good channel. Nevertheless, as noted above, ARCH is still able to achieve a 2.2× improvement in delivery rate on this link over the fixed-channel scheme.

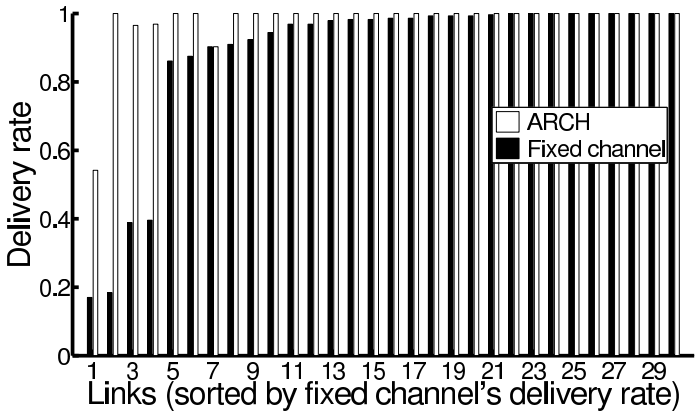



(a) CDF of average ETX of 30 links.

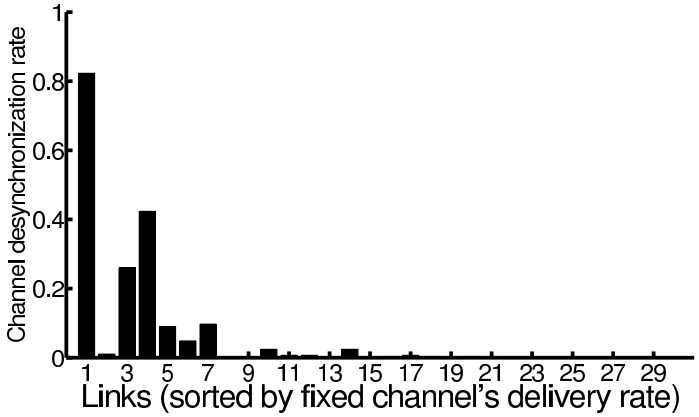



(b) Normalized average ETX (ARCH's divided by fixed-channel's ETX) of all links.

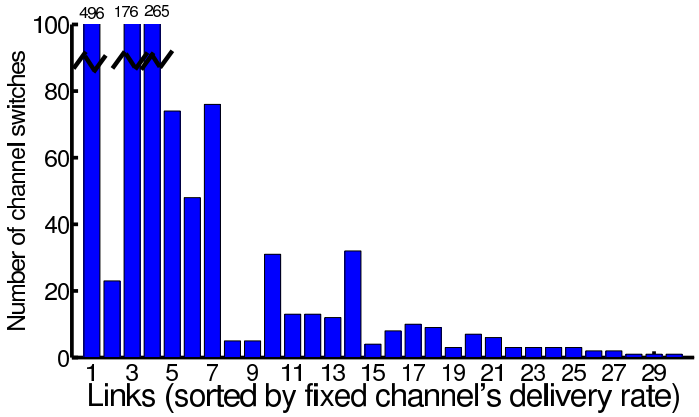
Figure 3.5: Comparison of ETX during single-hop data collection experiments under ARCH and fixed-channel schemes.




(a) A comparison of each link's delivery rate.




(b) The number of channel desynchronization events for each link.



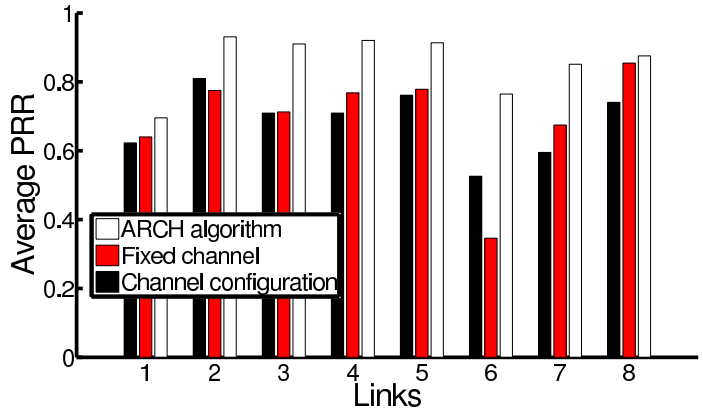

(c) Overhead of ARCH in terms of channel switches.

Figure 3.6: A per-link breakdown of the performance under a single-hop data collection experiment. For comparison, all results are sorted by the link's delivery rate under a fixed-channel scheme.

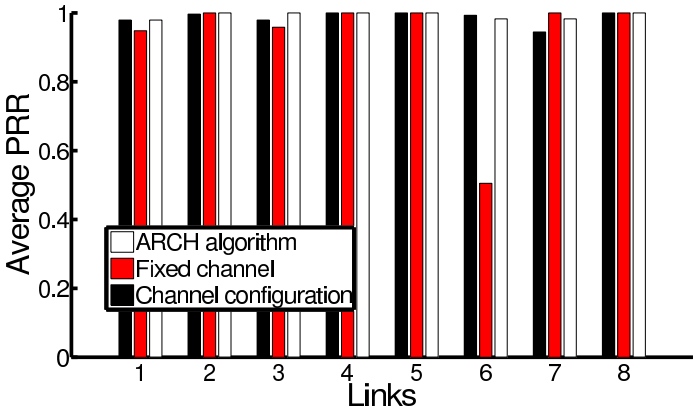
Figure 3.6(c) presents the overhead of ARCH in terms of channel switches. As with the simulator experiments, we observe that the number of channel switches is quite low. 18 links in the experiment require 10 or fewer switches per day to maintain reliability. Three links require more than 100 switches per day; this was due to repeated channel desynchronization events caused by strong interference at the receivers. Comparing Figures 3.6(a) and 3.6(c), we note that ARCH is effective even on these links with severe interference. On these three links, ARCH improved the delivery rate from 2.5×–5.6× compared to the fixed-channel scheme.

To further explore ARCH's real-world performance, we repeated the experiment using the BoX-MAC-2 [81] low power listening MAC layer. BoX-MAC-2 is a commonly-used protocol which automatically duty-cycles the radio in order to reduce nodes' idle listening cost. In order to directly measure the effect of automatic packet retransmissions, we performed these experiments twice: once with retransmissions disabled, and once with retransmissions enabled. In addition to the previously used fixed channel baseline, we added the channel configuration baseline described in Section 3.4.1. Due to time constraints, the experiment was reduced in size from 30 links among 10 apartments to 8 links in one apartment.

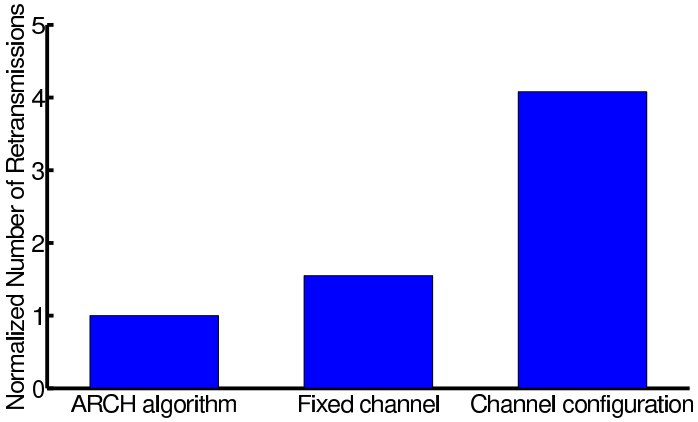
The results are plotted in Figures 3.7(a) and 3.7(b), respectively. Without retransmissions, we see that ARCH has uniformly higher delivery rates than either of the baselines, achieving an average delivery rate of 85%. With retransmissions, the delivery rate increases to 99%. Similar gains are seen for the fixed-channel and channel configuration schemes when retransmissions is enabled; however, as shown in Figure 3.7(c), they pay a much higher retransmission cost to achieve this level of reliability. Over the 24 hour experiment, the fixed-channel and channel configuration schemes need to transmit 1.6× and 4.1× as many packets as under ARCH, respectively, to maintain high reliability. Moreover, from Figure 3.7(b), we see that link 6 failed under the fixed channel scheme: even with retransmissions enabled, it achieved a PRR of only 51%. Again, this observation illustrates that retransmission alone is not sufficient for long-term reliable operation under significant interference and environmental dynamics.




(a) Delivery rate of 8 links without retransmissions.




(b) Delivery rate of 8 links with retransmissions.




(c) Number of transmitted packets with retransmissions.

Figure 3.7: Performance comparison of ARCH, fixed channel, and channel configuration schemes under single-hop data collection when using BoX-MAC-2.

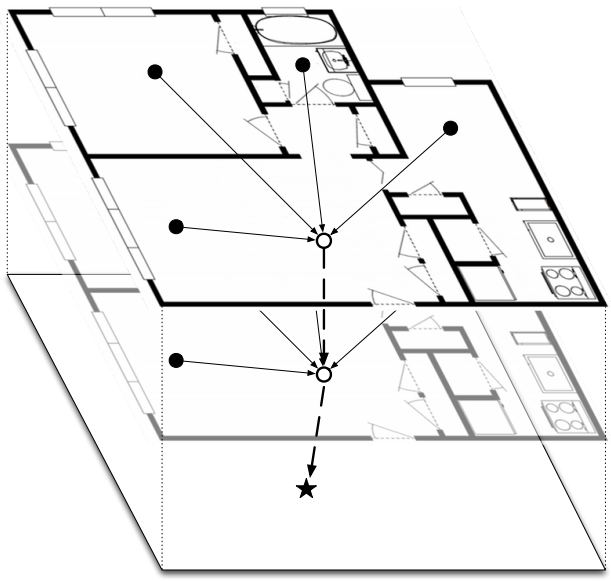


Figure 3.8: 3D diagram of sensor placement and collection tree for the multi-hop experiment. Nodes (circles) are laid out similarly on two floors, with a sink (star) in the basement. Hollow circles indicate relays.

### Multi-Hop Data Collection

Finally, we measured ARCH's performance under a multi-hop routing scenario. For this experiment, we deployed 10 sensor nodes between two floors of an apartment, as shown in Figure 3.8; an 11th node was deployed in the building's basement as the data sink. In order to isolate ARCH's performance from the decisions of the routing layer, our experiment used a fixed routing topology. 8 nodes were designated as leaf nodes and attempted to deliver 1 packet/5 minutes through fixed paths to the sink. 2 additional nodes in the network's interior acted as relays only, and did not produce data packets of their own.

In order to understand ARCH's effect on energy usage at a lower level, we instrumented the nodes' CC2420 radio stack to keep a cumulative count of how long the radio hardware was powered on. As with the previous experiment, retransmissions and BoX-MAC-2 were enabled, and a second run using a fixed-channel (channel 26) scheme was performed as a baseline. Each experimental run was carried out for 24 hours.

Figure 3.9 compares the end-to-end delivery rate under ARCH and the fixed-channel scheme. As with the previous benchmark, the use of retransmissions enables uniformly high delivery

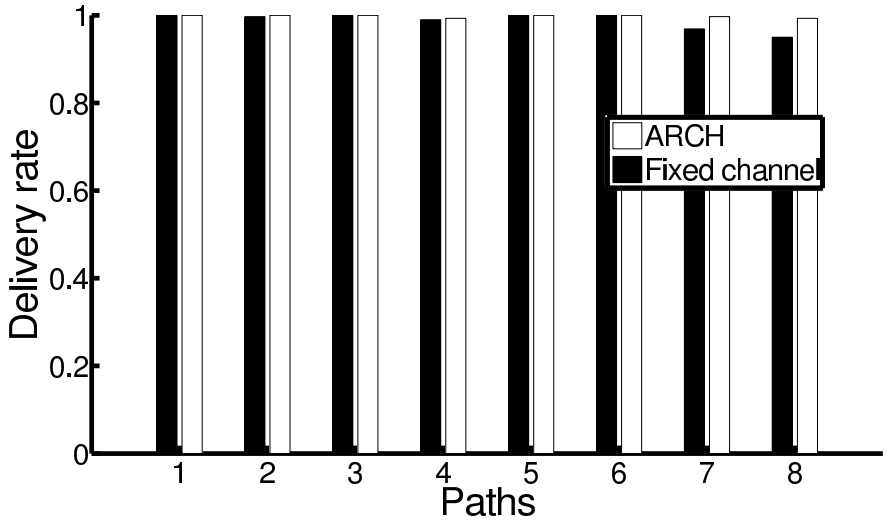
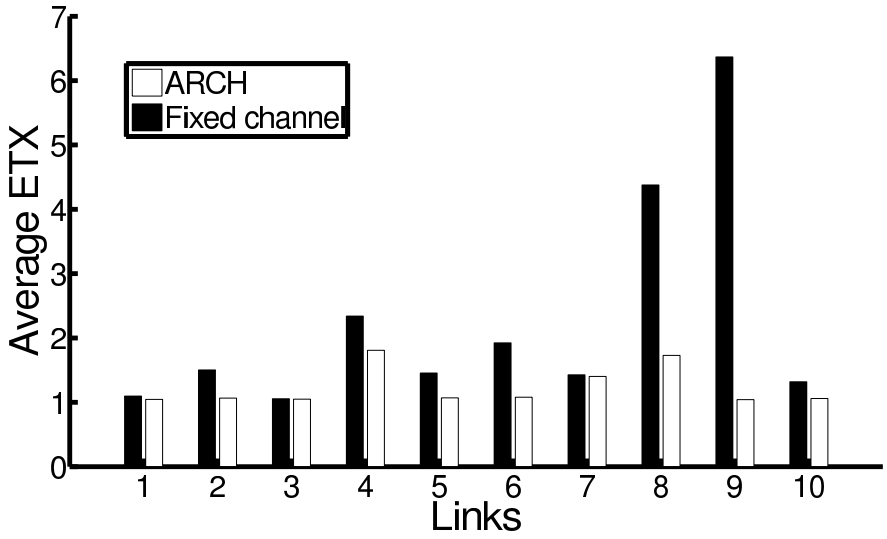



Figure 3.9: The delivery rate of all 8 multi-hop paths.

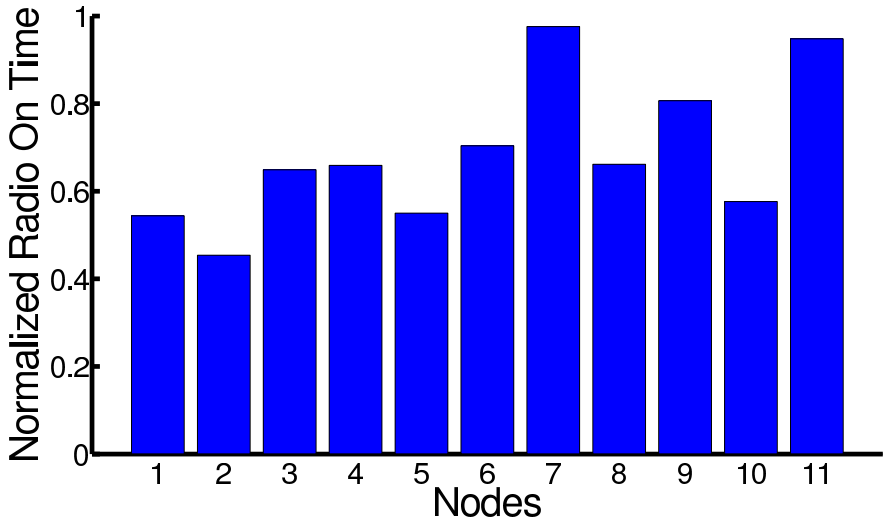
rates under both schemes. ARCH and fixed-channel delivered a minimum of 99.3% and 95.0% of their packets over each path, respectively.

However, as shown in Figure 3.10, the fixed-power scheme incurred a much higher energy burden to achieve this degree of reliability. Figure 3.10(a) compares the average ETX of each of the 10 links in the network. We observe that the fixed-power scheme required an average of 1.8× as many transmissions as ARCH, and up to 6.1× in the most extreme case. Indeed, *no* link performed worse with ARCH than the fixed-channel scheme. Consequently, ARCH proved to be significantly more energy-efficient. For each node, Figure 3.10(b) plots the ratio of radio usage between the two experimental runs. ARCH reduced the amount of time the radio was powered on by an average of 31.6%, representing a significant savings in energy consumption.

As shown in Figure 3.11, ARCH achieved this improvement with low overhead. Each link required only 2–11 channel hops during the 24-hour experiment, with a mean of 9.6 hops.




(a) The average ETX of all 10 links.




(b) The normalized radio usage of all 11 nodes under ARCH.

Figure 3.10: A comparison of energy efficiency between ARCH and fixed-power under multi-hop data collection.

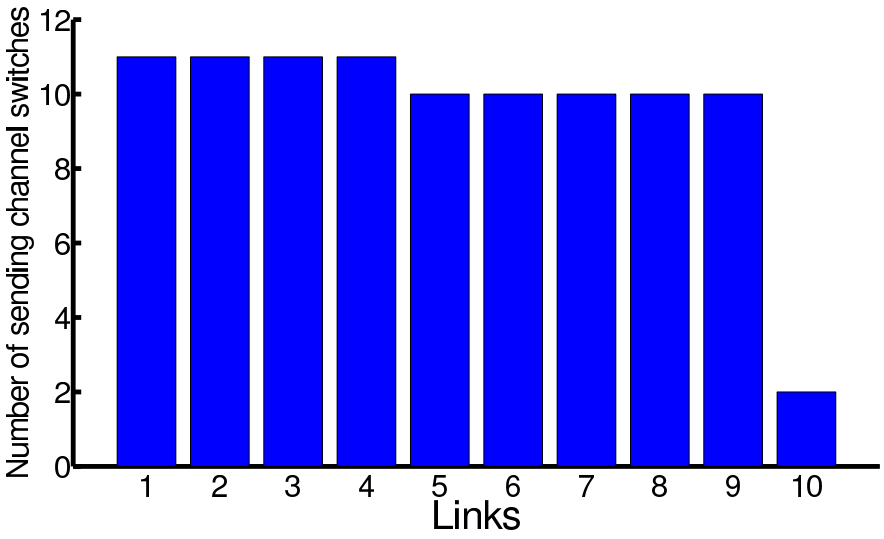



Figure 3.11: ARCH's overhead in terms of channel switches under multi-hop data collection.

## 3.5 Conclusion

Achieving reliable HAN performance in real-world residential settings can be challenging, due to their highly complex and dynamic wireless environments. Based on empirical study of these environments, we proposed the Adaptive and Robust Channel Hopping (ARCH) protocol: a lightweight yet effective channel hopping protocol that can handle the dynamics of channel conditions in apartments using a handful of channel hops per link per day. ARCH has several key features. First, ARCH is an *adaptive* protocol that channel-hops based on changes in channel quality (specifically, the Estimated Transmission Count, or ETX) observed in real time. Second, ARCH is a *distributed* protocol that hops channel on a per-link basis in order to avoid localized sources of channel failure. Third, ARCH is designed to be *robust and lightweight*. ARCH uses a practical hand-shaking approach to handle channel desynchronization and an efficient sliding-window scheme that does not involve expensive calculations or modeling and can be reasonably implemented on memory-constrained wireless sensor platforms. In a multi-hop data collection application, ARCH introduced an overhead of only 480 bytes of ROM and 26 bytes of RAM. Fourth, ARCH introduces *minimal communication overhead* for applications where packet acknowledgements are already enabled. Trace-driven simulations and real-world macrobenchmarks demonstrate the efficacy of ARCH's design. Single-hop experiments reveal a median decrease in packet

retransmissions of 42.3% and a 17% increase in the proportion of links with perfect delivery rates. ARCH provides even greater benefit for the most challenging of links, increasing the minimum delivery rate in our experiments by a factor of 2.2×. Further multi-hop experiments revealed a 31.6% average reduction in radio usage, representing a significant savings in energy. ARCH's lightweight design enables these dramatic reliability improvements with relatively few channel hops; most links required 10 or fewer channel hops per day.

# Chapter 4

# Energy-Efficient Low Power Listening for Wireless Sensor Networks in Noisy Environments

## 4.1 Introduction

Clear Channel Assessment (CCA) is a fundamental mechanism in MAC protocols for wireless networks. A CCA check [4] samples the energy level in the wireless channel and considers the channel busy if the energy level is above a threshold, or idle otherwise. CCA has been commonly used for two important (and orthogonal) purposes. First, it has been used by CSMA/CA protocols to avoid collisions on shared wireless channels, by sampling the channel for activity just before transmission. Second, CCA has been used in Low Power Listening (LPL), a popular MAC-layer approach that enables the radio to operate at low duty cycles. Under LPL, every node periodically wakes up to perform CCA. It then stays awake to receive packets if the CCA check detects activity in the wireless channel, or goes back to sleep immediately otherwise. Due to its simplicity and effectiveness, LPL has been a popular approach to energy-efficient MAC protocols in Wireless Sensor Networks (WSNs). A multitude of LPL-based MAC protocols has been developed in recent years [22, 81, 87], and LPL has been implemented by many radio drivers inside sensor operating systems such as TinyOS [4] and Contiki [7].

[4] CCA, carrier sense and energy detection are used as synonymous in this Chapter, as supported by [91].

While the effect of CCA on collision avoidance has been well studied in the literature, its impact on LPL, particularly in noisy environments such as residential and office environments, has received relatively little attention. Applications deployed in noisy wireless conditions are susceptible to frequent *false wakeups*: noise may be detected as legitimate activity on the channel, causing the node to remain awake even when no transmissions occur. False wakeups may significantly increase the duty cycle and energy consumption of the nodes, as shown by our empirical studies in residential environments (see Section 4.4). This limitation of LPL protocols is becoming increasingly significant as more and more WSNs are being deployed in residential environments, where co-existing wireless devices and electromagnetic equipments cause prevalent and highly variable noise.

To address this important problem, we propose a novel approach that dynamically adjusts the *CCA threshold*, i.e., the energy level threshold used to decide if a channel is active. This approach is motivated by the key observation that nodes may effectively reduce false wakeups by choosing a threshold above the background noise level, but below the level of real transmissions. Specifically, the main contributions of this work are three-fold:

1. An empirical study in residential environments that demonstrates the potential benefits of adaptive CCA control based on both normal channel conditions and controlled 802.11n traffic;

2. Adaptive Energy Detection Protocol (AEDP), an adaptive protocol that dynamically adjusts a node's CCA threshold to improve network reliability and duty cycle based on application-specified bounds;

3. Discovery of significant shortcomings in the implementation of CCA checks in TinyOS 2.1.1 [5] caused by improper selections of key radio parameters, and a systematic methodology to tune these parameters in order to enable efficient CCA checks.

In contrast to previous studies on adjusting the CCA threshold to better avoid collisions in both 802.11 [21] and 802.15.4 [18, 19, 63, 129] networks, this Chapter investigates the CCA threshold's role in waking up nodes, with the goal to mitigate the false wakeup problem associated with LPL; to our knowledge, it represents the first systematic study of the CCA

[5] The shortcomings still exist in TinyOS 2.1.2 officially released on August 20, 2012.

threshold's role in the effectiveness of LPL. The uses of CCA for collision avoidance in CSMA/CA and wakeup in LPL are orthogonal and complementary to each other, as CCA is being used at different times for different goals. Indeed, both forms of CCA adjustment could be deployed simultaneously, by simply maintaining separate thresholds for collision avoidance and LPL.

The remainder of this Chapter is organized as follows. Section 4.2 compares our approach with related works. Section 4.3 describes an overview of LPL. Section 4.4 presents an empirical study into the effect of noise on LPL behavior, and explores the use of CCA thresholds to control the associated false wakeup problem. Section 4.5 details the AEDP protocol for dynamically adjusting a node's CCA threshold in order to minimize false wakeups. Section 4.6 describes the implementation of AEDP on the TelosB mote platform and analyzes the impact of radio parameters on the effectiveness of CCA. Section 4.7 presents an empirical evaluation of AEDP in both controlled tests and real-world environments. Finally, we conclude the Chapter in Section 4.8.

## 4.2 Related Work

Traditionally, CCA functionality has been used in CS-MA/CA MAC protocols to avoid collisions on shared wireless channels. A sender performs CCA before transmission. It proceeds with the transmission if the CCA check does not detect channel activity; otherwise it backs off to avoid colliding with an on-going transmission. Numerous studies have explored the impact of the CCA thresholds used for collision avoidance on both 802.11 networks and WSNs [18,19,21,63,129]. Bertocco et al. [18] shows that the CCA threshold is critical, as false negative channel activity detections result in collisions and false positives cause increased latency. Kiryushin et al. [63] studies the real-world impact of CCA thresholds in avoiding packet collisions. Chintalapudi et al. [27] shows that a poor energy detection scheme can lead to significant overhead for listening to the channel and switching the radio between send and receive modes, which may take hundreds of microseconds. Boano et al. [19] shows that tuning the CCA threshold at run time can improve the robustness of existing MAC protocols under interference. Yuan et al. [129] presents that dynamically adjusting CCA threshold can substantially reduce the amount of discarded packets due to channel access failures.

Brodsky et al. [21] presents an opposite conclusion based on theories of radio propagation and Shannon capacity and shows that it is possible to choose a fixed CCA threshold which performs well across a wide range of scenarios since carrier sense performance is surprisingly close to optimal for radios with adaptive bitrate. All these works focus on the impact of CCA on collision avoidance in transmissions rather than its use for wakeup in LPL-based MAC protocols.

In contrast to previous studies on CCA for channel avoidance, this Chapter investigates the CCA threshold's role in avoiding the false wakeup problem associated with LPL; to our knowledge, it represents the first systematic study of the CCA threshold's role in the effectiveness of LPL in achieving low duty cycles in WSNs, especially in noisy environments where traditional LPL protocols are vulnerable to false wakeup problems. Our work is therefore *orthogonal* and *complementary*.

Recently, receiver-initiated MAC protocols have been proposed to avoid the false wakeup problem. Receiver-initiated MAC protocols such as [32, 118] require recipients to transmit probe packets indicating that they are ready for packet reception. As our experiments presented in Section 4.7.5, AEDP is more energy efficient at low data rates than the state-of-the-art receiver-initiated MAC protocol A-MAC [32], as AEDP avoids the overhead of the probe packets. On the other hand, A-MAC is more energy efficient than AEDP for high data rate applications, where the cost of sending these probe packets are offset by reduced overhead for transmissions. Our work is therefore an alternative sender-initiated approach that is complementary to receiver-initiated MACs for applications with different data rates.

ContikiMAC [31] addresses the false wakeup problem with two targeted optimizations. First, it performs *two* CCA checks spaced slightly apart, allowing it to identify phenomenon too short to be an 802.15.4 transmission. Second, it performs a "fast sleep" optimization that reduces the cost of false wakeups, by detecting patterns of activity and silence which cannot belong to ContikiMAC transmissions. In our testing, we found that our approach's CCA-threshold-adjustment can effectively avoid false wakeups without these optimizations. Since our approach requires only a *single* CCA check, it induces lower energy cost in low duty-cycle cases where nodes rarely need to wakeup to receive packets. Nevertheless, these approaches are orthogonal, and in particularly challenging environments could be combined to reduce both the likelihood and the energy overhead of false wakeups.

There has been increasing interest in studying the impact of interference on WSNs and enhancing the robustness of MAC protocols in noisy environments. Srinivasan et al. [115] examines the packet delivery behavior of two 802.15.4-based mote platforms, including the impact of interference from 802.11 and Bluetooth. Liang et al. [70] measures the impact of interference from 802.11 networks on 802.15.4 links, proposing the use of redundant headers and forward error correction to alleviate packet corruption. These studies focus on improving the reliability of transmission and do not deal with the false wakeup problem to improve energy efficiency.

## 4.3 Overview of LPL

*Low power listening* (LPL) is a common MAC-layer technique for reducing energy consumption in WSNs [87]. Under LPL, nodes periodically wakeup to perform CCA, i.e., to briefly sample the wireless channel for activity. If energy is detected on the channel, the node remains awake in order to receive a packet (or until some timeout). Otherwise, the node quickly goes back to sleep. To minimize overhead when the network is idle, these periodic wakeups are not synchronized across nodes: that is, the recipient knows the recipient's wakeup interval but not its wakeup time. Accordingly, before transmitting a packet, the transmitter sends a preamble stream at least as long as the recipient's wakeup interval; this ensures that the recipient will sample the channel during the preamble. After the preamble, the sender and recipient exchange data packets.

Later LPL-based MAC layers such as X-MAC [22] modify this approach by inserting destination address information and periodic gaps in the preamble stream. When a node wakes up, it may decode the destination address and see if it is the packet's intended recipient. If so, it uses the gaps in the preamble to send an acknowledgment to the sender, which will in turn immediately transmit the payload. If not, the node may go back to sleep immediately. These enhancements significantly reduce the cost of waking up for a packet intended for another node, while also reducing the average cost of unicast packet transmissions by half. BoX-MAC-2 [81] further refines this approach by transmitting the entire data packet in place of the destination address, eliminating the need to explicitly exchange the payload after the recipient has ACKed the preamble.

Quickly and accurately assessing whether the channel is active is a critical component of a LPL-based MAC layer. Modern radios, including all 802.15.4-compliant hardware [57], provide CCA functionality that assists with this procedure. A common method for radios to implement CCA is to provide a digital readout (often a dedicated pin) indicating whether the channel's energy level currently exceeds some threshold. This particular implementation, known as *energy detection*, is commonly found in low-power radios such as the Chipcon CC2420 and has been identified as a critical feature for WSN hardware design [33]. After waking up the radio, the microcontroller may sample the CCA pin in a tight loop; the node remains awake for packet reception if some minimum number of samples are positive.

## 4.4 Empirical Study

This section describes a series of empirical studies that provide the motivation and insights for the design of AEDP. We first measure the false wakeup problem in office and residential environments, followed by a systematic study on the impact of CCA's energy detection threshold on wakeups in LPL.

### 4.4.1 Effects of Wireless Noise

Existing literature on LPL-based MAC layers emphasizes the ability to run applications at an extremely low duty cycle, sometimes as low as 1% [87], in exchange for moderately increasing the cost of packet transmissions. This tradeoff makes LPL well-suited for applications with low-to-moderate data rates. However, noise from other wireless devices can have a dramatic (and often unanticipated) impact on nodes' duty cycle, significantly reducing system lifetime.

Radios based on the 802.15.4 standard operate in the unlicensed 2.4 GHz band shared by many other devices. Energy detection simply looks for the presence of *some* signal on the wireless channel; it does not distinguish between the system's own traffic and noise from other devices. To illustrate how a false-negative wakeup can considerably increase the cost of a CCA check, we deployed a TelosB mote [30] running TinyOS 2.1.1 [4] in an office environment. The TelosB mote was configured to use the BoX-MAC-2 LPL-based MAC

layer, TinyOS's *de facto* standard LPL implementation. BoX-MAC-2 was in turn configured with a wakeup interval of 2 seconds: i.e., the motes sleep for 2 seconds between sampling the channel for activity. In order to capture the effects of wireless noise, we configured the CC2420 radio to use channel 18, which overlaps with a campus-wide 802.11g network. All other MAC layer and radio parameters were left to their respective defaults.

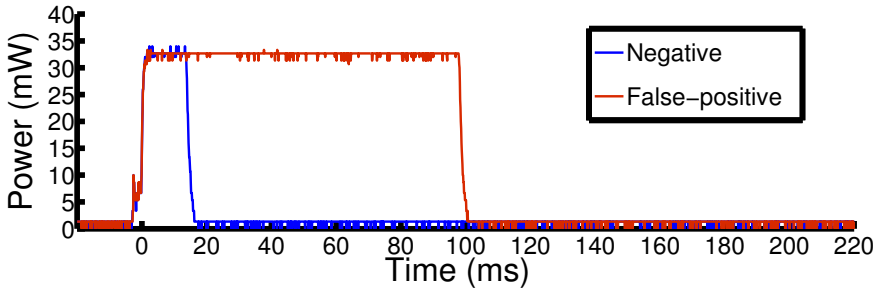



Figure 4.1: Oscilloscope traces comparing a TelosB node's energy consumption during a negative (idle) and false-positive (detected) energy detection check.

Figure 4.1 shows the energy consumption of this mote when performing a single energy detection check, as captured with an oscilloscope. When the channel is idle, the radio is powered on for 19.0 ms; in contrast, when the channel is occupied, the false wakeup causes the radio to remain powered on for 103.4 ms until it eventually times out. Similar results were observed in [32], which found that false wakeups increased the current consumption of a CCA check by 17.3×.

An equally important question is how often wireless noise will cause these false wakeups to occur in real-world environments. To measure this phenomenon, we deployed four pairs of TelosB motes on orthogonal channels (11, 16, 21, and 26, respectively) in five different apartments located in different neighborhoods in St. Louis. The motes were deployed for 24 hours in each apartment during the residents' normal activities. One mote in each pair was configured to transmit 1 packet every minute, and the BoX-MAC-2 MAC layer configured with a wakeup interval of 2 seconds. We augmented TinyOS's CC2420 radio stack to track the result of each CCA check and the radio "on time", i.e., the cumulative total time the radio was active during the entire experimental run. The latter data was in turn used to compute each mote's duty cycle. For the purposes of this experiment, the mote's onboard CC2420 was again configured with the hardware-default CCA behavior, setting its CCA pin based on an energy threshold of −77 dBm.

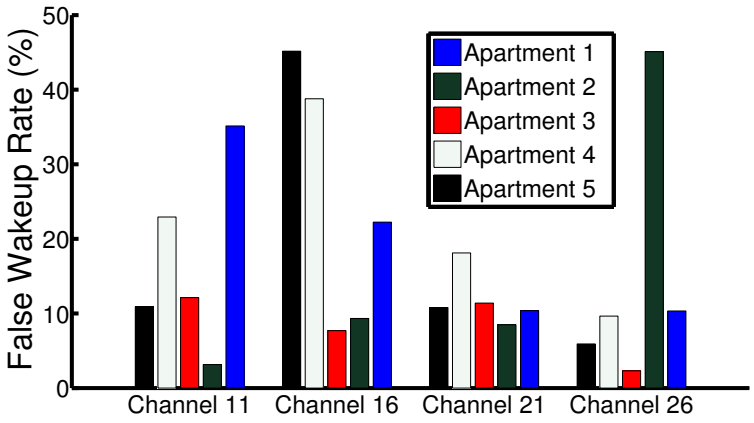
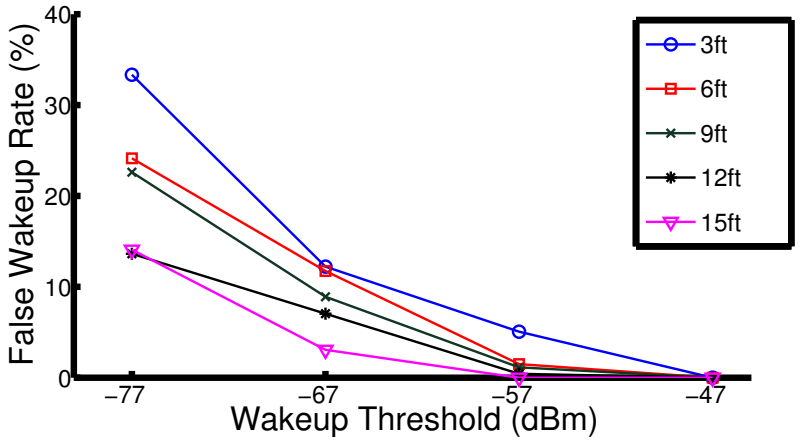



Figure 4.2: The false wakeup rate of each recipient mote in each apartment.

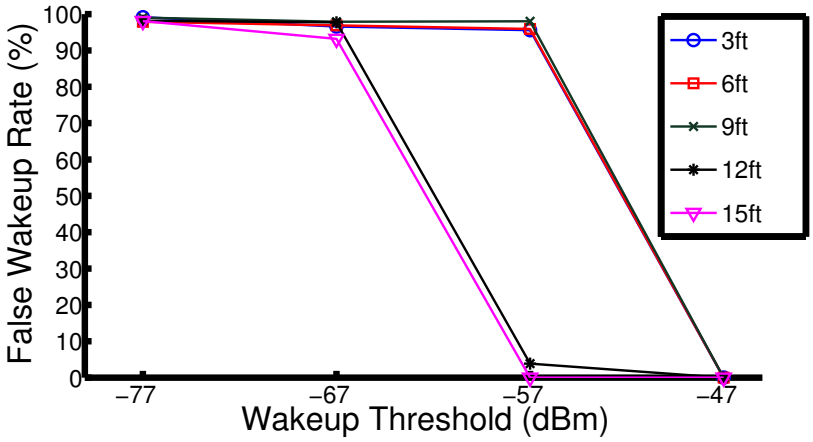
Figure 4.2 plots the false wakeup rate (the proportion of CCA checks resulting in wakeup but no packet reception) of each node in the experiment. From the receiver's wakeup interval and the sender's data rate, we expect a nominal duty cycle of 0.17%. However, the false wakeups caused by environmental noise result in substantially higher duty cycles, with an average duty cycle of 1.4% across all four tested channels in all five apartments. In the two worst cases — channel 16 in apartment 5 and channel 26 in apartment 2 — false wakeup rates of 45% resulted in greatly inflated duty cycles of 2.8%.

### 4.4.2 Effects of CCA Threshold

We propose to address the false wakeup problem by adjusting the CCA threshold: that is, the specific energy level used as a binary threshold to determine whether a node should remain awake. In the context of LPL, setting the CCA threshold too low will cause nodes to wakeup to receive non-existent packets. Setting the threshold too high may cause nodes not to wakeup during transmissions, forcing the sender to repeatedly retransmit. We note that adjusting the CCA threshold for LPL has *no* effect on the receiver's ability to decode packets, so long as the threshold is low enough to wakeup the receiver. Hence, link reliability will only be affected if the threshold is high enough to cause a false-negative energy detection (i.e., a node fails to stay awake to receive a legitimate packet).




(a) False wakeup rate under office occupants' normal activities.




(b) False wakeup rate under controlled (5 Mbps UDP) 802.11n traffic.

Figure 4.3: The effects of tuning the CC2420's wakeup threshold on the motes' false wakeup rate, subject to office occupants' normal activities and controlled 802.11n traffic. The motes were located 3–15 ft away from the 802.11n router, and were configured to use a threshold ranging from −77 to −47 dBm.

As discussed earlier, the CCA threshold also plays a role in the context of collision avoidance. However, adjusting the CCA threshold has a different effect in the context of collision avoidance, where it directly affects the *sender* rather than *receivers*. Setting the threshold too low encourages spurious backoffs, while setting the threshold too high may introduce packet losses from otherwise-avoidable collisions. To distinguish the CCA threshold used by the receiver for LPL from the CCA threshold used by the sender in CSMA/CA, we henceforth refer to the former in this Chapter as the *wakeup threshold*. This Chapter focuses on reducing false wakeups by manipulating the wakeup threshold used for LPL. We do not change the CCA threshold used for transmission, an important but orthogonal problem that has been well-studied in literature.

We perform a set of controlled tests in an office environment to investigate the potential energy savings from adjusting the wakeup threshold, we deployed five groups of four TelosB motes on channel 16 at varying distances (3–15 ft) from a pair of 802.11n devices (access point+MAC pro laptop) operating on 2.4 GHz band that overlaps with 802.15.4. Each experimental run was carried out for one hour; as before, BoX-MAC-2 was configured with a wakeup interval of 2 seconds. In contrast to the previous experiments, which used the radio-default CCA threshold of $-77$ dBm, each mote in a group was configured to use one of four different thresholds ($-77$, $-67$, $-57$, and $-47$ dBm). Signal generated by motes may become part of the background noise when its strength is lower than recipients' CCA threshold. We intentionally stop motes from generating real transmissions in this set of tests, thus we can treat the total wakeup rate as the false wakeup rate.

Figure 4.3(a) plots the recipients' false wakeup rate under the office occupants' normal activities in real-world environment. Figure 4.3(b) plots the false wakeup rate when using LanTraffic V2 [8] to generate a controlled stream of 5 Mbps UDP traffic through the pair of 802.11n devices. Two important conclusions may be drawn from these figures. First, tuning the wakeup threshold provides a powerful opportunity for conserving energy. We observe that the false wakeup rate drops dramatically when increasing the threshold from the radio default of $-77$ dBm. Under real-world activity as shown in Figure 4.3(a), the default threshold incurs a false wakeup rate of 14–33%. In comparison, this rate may be reduced to 3–12% by moderately increasing the threshold by 10 dBm, or to 0% by increasing the threshold by 30 dBm. The effects of tuning the threshold are even more pronounced under the higher-bandwidth controlled experiments, as shown in Figure 4.3(b). At a threshold

of −77 dBm, the nodes experience a false wakeup rate no lower than 97.8%, regardless of distance from the pair of 802.11n devices. This rate drops to 0–4% for two of the distances at a threshold of −57 dBm, and to 0% for all distances at a threshold of −47 dBm. Second, the "best" wakeup threshold is highly dependent on external factors such as the 802.15.4 nodes' vicinity to other devices, and the other devices' usage patterns and signal strength. Comparing Figures 4.3(a) and 4.3(b), we see that increasing the threshold from −77 dBm to −67 dBm significantly reduces the false wakeup rate under normal activities. However, under a sustained 5 Mbps UDP stream, a comparable threshold increase has virtually no impact on the false wakeup rate.

We also used motes to perform a series of measurements on signal strength of external interference generated by several real-world 802.11 applications as well as the LanTraffic V2 with various speeds. We observed that noise varies from application to application and over time for a given application depending on the distance from interference source.

Hence, picking an appropriate wakeup threshold is not simply a matter of choosing a more aggressive default setting. The minimum threshold needed to avoid noise varies from setting to setting, and even over time depending on the occupants' activities. Moreover, selecting too high of a threshold will intuitively cause the receiver to stop waking up for legitimate transmissions, decreasing network reliability.

## 4.5 Protocol Design

In this section, we present the design of our AEDP. At a high level, AEDP tries to meet application-specified constraints on network reliability and wakeup rate. The desired network reliability is specified by the desired ETX, $ETX_{threshold}$, where ETX is the expected number of transmissions needed to successfully send a packet to its destination. The desired wakeup rate, $WR_{threshold}$ can be determined based on the application data rate (and hence the corresponding *true* wakeup rate) plus a small margin for false wakeups allowed by the application. When it is not possible to meet both constraints, network reliability takes precedence, as it is typically more critical than lifetime constraints. We set a default value of $ETX_{threshold}$ to be 5 and a default value of $WR_{threshold}$ to be 5 times of data rate according to the typical low data rate home automation systems.

AEDP maintains three variables at run time: $ETX$, $WR$, and $WR_L$. $ETX$ is the average ETX value over a sliding window (default window size is 15 minutes). $WR$ is the wakeup rate within the same sliding window. $WR_L$ is the cumulative wakeup rate over the whole application lifetime. Note that $WR_L$ reflects the long-term wakeup rate that affects the battery life of the node.

At runtime, AEDP periodically updates these three variables $ETX$, $WR$, and $WR_L$ and compares their values against $ETX_{threshold}$ and $WR_{threshold}$. It then computes a new wakeup threshold $T$ based on four different cases, described below.

1. **Case 1:** $ETX$ exceeds $ETX_{threshold}$. AEDP attempts to quickly recover by significantly reducing the wakeup threshold. This policy reflects the fact that network reliability constraints are typically more critical than lifetime constraints.

2. **Case 2:** $ETX$ meets $ETX_{threshold}$ but $WR$ exceeds $WR_{threshold}$. This case indicates that the current wakeup threshold is too low to achieve the desired wakeup rate. AEDP increases the wakeup threshold by a small amount $\Delta T$ to try to meet the application's bound on wakeup rate. The default value of the tuning step $\Delta T$ is set to be 2 dBm.

3. **Case 3:** $ETX$, $WR$, and $WR_L$ all meet their respective constraints. This case indicates that the current wakeup threshold is meeting the application's constraints, both in this period and over the application's lifetime. AEDP aims to find the minimum threshold that does so, as lower wakeup thresholds are potentially more robust to changes in topology and signal strength. Hence, AEDP decreases the wakeup threshold by $\Delta T$.

4. **Case 4:** $ETX$ and $WR$ meet their constraints but $WR_L$ does not. Here, AEDP takes no action. Since $WR$ is below $WR_{threshold}$, the wakeup threshold is high enough to meet the application's wakeup rate constraint in the short term. However, $WR_L$ has still not met the application's constraint over the long term, so AEDP will not yet start to reduce the wakeup threshold.

In all cases, AEDP constrains the wakeup threshold $T$ to a range $[T_{min}, T_{max}]$. Reducing $T$ too much will cause the node to always be awake, while increasing $T$ too much will cause packet loss (increased ETX). AEDP sets $T_{min}$ to be the noise floor to avoid sustained wakeups, and

sets $T_{max}$ to be the minimum Received Signal Strength (RSS) of incoming links, since our experimental results have shown that link reliability degrades heavily when $T$ exceeds the RSS of incoming link [105]. To accommodate topology changes, AEDP periodically resets the wakeup threshold to $T_{min}$ for several periods (a default value of 5 wakeup intervals) enabling node to establish new incoming links with RSS lower than $T$.

AEDP has several key design features based on the observations in our empirical study. First, AEDP adaptively adjusts energy detection threshold based on changes in network reliability (specifically, ETX) observed at runtime. Second, AEDP performs its computations based solely on local state ($WR$, $WR_L$, and $ETX$), requiring no additional transmissions between sender and receiver. Third, AEDP is a lightweight protocol that only piggybacks a single byte (used to measure ETX) in each existing packet transmission, and introduces no other traffic of its own.

## 4.6 Implementation

In this section, we discuss our implementation of AEDP on TinyOS 2.1.1. We first describe the software architecture used by AEDP. We then discuss several key radio parameters that affected the energy efficiency of LPL, and present a methodology for picking these parameters appropriately.

### 4.6.1 AEDP Architecture

We implement the AEDP algorithm as a layer situated between the application and MAC layers. This layer consists of three important components. The *WakeupRateMonitor* component tracks the wakeup rate $WR$ and computes the cumulative wakeup rate $WR_L$. The *LinkEstimator* component measures the ETX of incoming packets using sequence numbers in each packet, and computes the average ETX value ($ETX$) over a sliding window. The LinkEstimator also measures the RSS of incoming packets, using the minimum average RSS value of all incoming links as the bound $T_{max}$. The *CCAControlEngine* component computes and sets the wakeup threshold based on the values $ETX$, $WR$ and $WR_L$.

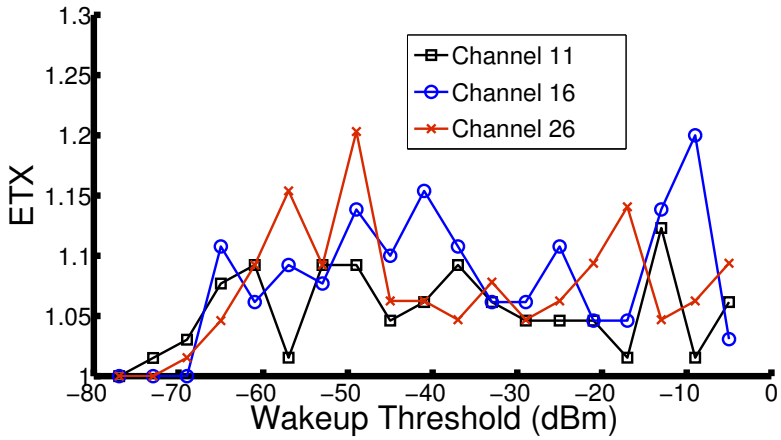



Figure 4.4: The relationship between wakeup threshold and ETX in the default TinyOS CC2420 stack.

AEDP requires several modifications to the radio stack to support its operations, as listed below. For the purposes of this implementation, we have performed these modifications on TinyOS 2.1.1's default CC2420 + BoX-MAC-2 stack.

First, we add a *PacketInfo* interface between the MAC layer and LinkEstimator to expose the ETX and RSS values of each incoming packet. The LinkEstimator buffers the values in sliding windows, calculating the average ETX and RSS values for the variables $ETX$ and $T_{max}$ respectively.

Second, we augment the radio core to count wakeup events. This counter is exposed to the WakeupRateMonitor through the *WakeupCounter* interface and used to compute the values of $WR$ and $WR_L$.

Finally, we add a *CCAcontrol* interface to the radio core to expose the radio's hardware CCA threshold setting. On the CC2420, this is implemented by writing the new threshold to the radio's `CCA_THR` register, plus a 45 dBm offset specified by the datasheet [3]. The CCA Control Engine uses this interface to set the newly-computed wakeup threshold $T$.

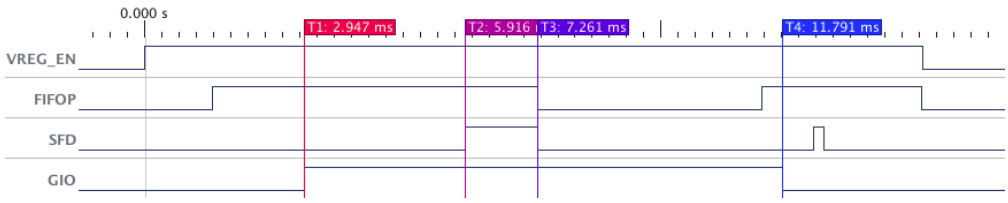



Figure 4.5: A logic analyzer trace demonstrating the CC2420 fully decoding a packet during the energy detection check. The microcontroller uses the VREG_EN pin to control the CC2420's power state. The CC2420 uses the SFD and FIFOP pins to signal the beginning (T2) and end (T3) of packet reception, respectively. The GIO pin indicates the duration of the check (T4–T1).

### 4.6.2 System Parameters

When testing our first implementation of AEDP on the TelosB mote, we were initially surprised to discover that increasing the wakeup threshold had little impact on network reliability. Figure 4.4 plots the relationship between the wakeup threshold and ETX that we observed in our initial testing, using three different channels and a wide range of threshold values. We initially expected that an excessively high threshold would cause significant packet loss, and a high enough threshold would prohibit the node from receiving packets at all (due to never waking up from sleep). However, in practice, we observed that an overly aggressive threshold only increased the number of retransmissions by a maximum of 20%. Indeed, the node still received packets after setting the threshold to the radio maximum of 82 dBm, or even after modifying the CC2420 stack to *always* put the radio back to sleep regardless of the energy detection result.

From these results, we hypothesized that the radio was fully receiving and decoding entire packets during the CCA check itself. TinyOS's implementation of BoX-MAC-2 on the CC2420 detects energy by sampling the CCA pin up to 400 times in a tight loop. Modern packet-based radios like the CC2420 are designed to fully decode packets without the microcontroller's intervention, and could decode packets while the microcontroller is occupied by polling the CCA pin.

We confirmed this hypothesis using a logic analyzer to trace the sequence of events inside the radio hardware and radio stack. Figure 4.5 presents a sample trace that we captured

with the CC2420 configured to use the maximum threshold[6]. At 0 ms, the radio stack begins sampling the wireless channel by powering on the CC2420. The CC2420 is fully powered on at T1 = 2.947 ms, and the radio stack starts energy detection. At T2 = 5.916 ms, the CC2420 signals the beginning of a packet reception; at T3 = 7.261 ms, the CC2420 signals that the packet is fully decoded. The radio stack will not finish energy detection until $T4 = 11.791$ ms. Indeed, the duration of this check (8.844 ms) is much longer than the on-air time of an 802.15.4 packet ($0.59 - 4.24$ ms in lab experiments, depending on payload size).

The apparent cause for this lengthy check is a long ACK delay built into TinyOS's CC2420 driver. After transmitting a packet, the driver waits up to 8 ms for an ACK packet. In our own measurements, this resulted in BoX-MAC-2 leaving the channel idle for 8.3 ms between retransmissions.

In principle, an ACK delay of this length is unnecessary. From the 802.15.4 specification, we can derive a tight bound of 544 $\mu$s on the ACK delay. (Specifically, the recipient must transmit an ACK exactly 192 $\mu$s after decoding the incoming packet's last bit, and transmitting the ACK packet takes 352 $\mu$s at 802.15.4's 250 kbps data rate [32, 57].) However, TinyOS disables the CC2420's hardware auto-acknowledgement feature due to concerns over its reliability [9]. Consequently, packets must pass partway through the recipient's radio stack before they are acknowledged, adding significant delay.

Nevertheless, we believe that the default ACK delay is overly conservative. In a microbenchmark experiment, we transmitted packets between a pair of TelosB motes with hardware auto-acknowledgement disabled. The transmitter requested an ACK for each transmission, and recorded the delay between finishing a transmission and receiving the corresponding ACK. Out of 2000 transmissions, the transmitter observed a mean delay of 2.2 ms and a maximum delay of 2.4 ms.

This result indicates that an 8 ms ACK delay, and the associated 8.8 ms energy detection length, is excessive. The length of this check contradicts the need for a short, inexpensive energy detection, and arguably even renders the entire check ineffective. From the 20% ETX

[6]For illustrative purposes, we modified BoX-MAC-2 to mark the duration of the energy detection loop using a GIO pin, and to disable a code branch that short-circuits the loop when the radio starts receiving a packet. We have verified that the CC2420 will still fully decode packets during energy detection, even without these modifications.

| Driver | ACK delay |
|---|---|
| CC2420(`cc2420` driver) | 8 ms |
| CC2420(`cc2420x` driver,most platforms) | 1 ms |
| CC2420(`cc2420x` driver,`micaz` platforms) | 0.8 ms |
| CC2520 (most platforms) | 1 ms |
| CC2520(`sam3s_ek` platform) | 0.8 ms |
| RF230 | 1 ms |
| **IEEE 802.15.4 specification** | 0.544 ms |
| **TelosB (lab measurements)** | 2.4 ms |

Table 4.1: The ACK delays used by various 802.15.4 radio drivers in TinyOS, the ACK delay derived from [57], and the actual ACK delay measured on a TelosB.

penalty we observed in our testing, it would have been nearly as effective to simply leave the radio on for 8.8 ms, and ignore the energy detection result. Doing so would have had only a small impact on network reliability, in exchange for *never* incurring a false wakeup.

Instead, for the purposes of implementing and evaluating AEDP, we opt to retain the check but reduce the CC2420 driver's ACK delay to 2.8 ms (2.3 ms + 0.5 ms guard space). We accordingly modify BoX-MAC-2 to poll the CCA pin up to 115 times, reducing the energy detection duration from 8.8 ms to 2.9 ms. In general, the duration of CCA polling must be longer than the ACK delay to avoid false negatives in energy detection which can heavily worsen the performance.

As we show in Section 4.7, this modification alone has the effect of significantly reducing the motes' duty cycle, simply by reducing the cost of energy detection to a fraction of its default length.

Although this modification is specific to the particular radio stack used, it emphasizes the need for a general methodology — such as the analysis performed above — to tune these key radio and MAC layer parameters. Indeed, as shown in Table 4.1, TinyOS employs three different ACK delays on the sender side, depending on the combination of radio driver, radio stack, and underlying mote platform. None of these three different delays is consistent with the theoretical ACK turnaround time from the 802.15.4 standard, or with the actual turnaround time measured on the TelosB. Besides energy efficiency, this inconsistency raises concerns about basic interoperability.

## 4.7 Evaluation

To validate the efficiency of our approach in reducing false wakeup rates, we performed a series of controlled experiments and real-world experiments. (1) We first evaluate the capability of AEDP to effectively converge to the desired wakeup threshold. (2) We then performed an experiment where additional transmitters were added to the network at runtime to test AEDP's resilience to network changes. (3) We evaluate AEDP's impact on duty cycles at the link level, and compare AEDP's performance against LPL configurations in a testbed we deployed in a 3-floor apartment building. (4) We compare AEDP against A-MAC, a state-of-the-art receiver-initiated MAC protocol under different data rates. (5) Finally, we evaluate the impacts of AEDP on multi-hop data collection by running AEDP with CTP in a 55-node testbed in an academic building.

In all experiments, we deploy our benchmark applications on top of TelosB motes running the TinyOS 2.1.1 operating system. BoX-MAC-2 is configured with a wakeup interval of 2 seconds: i.e., the motes sleep for 2 seconds between sampling the channel for activity. We use a data rate of 1 packet/5 minutes [7] for all evaluations except the one in Section 4.7.5, where we evaluate the performance of AEDP under different data rates.

We emphasize that our experiments changed only the CCA threshold used for wakeup and did not change the threshold used for collision avoidance; hence, improvements in duty cycle are attributed to a reduction in false wakeups rather than retransmissions.

### 4.7.1 Self-tuning Wakeup Threshold

We first test the capability of AEDP to automatically adjust its wakeup threshold. For this experiment, we deployed a pair of motes with AEDP on channel 16. We also deployed an 802.11n access point and a laptop producing 1 Mbps of UDP traffic on 802.11 channel 6, which overlaps with 802.15.4 channel 16. We performed two experimental runs: to vary the impact of the interfering 802.11 network on the mote pair, the distance between the mote

[7] The data rate is chosen according to the typical sampling rate of home automation systems (for example, 1 temperature reading every 5 minutes is sufficient for an HVAC system to control ambient temperature).

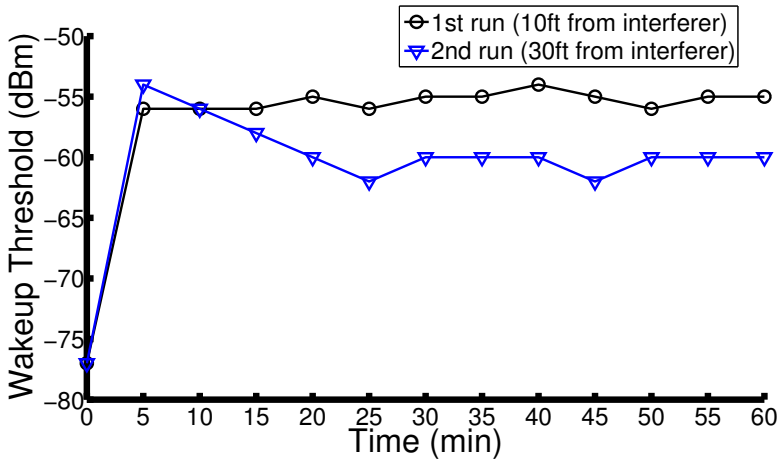



Figure 4.6: AEDP adapting the wakeup threshold over time.

pair and the 802.11n devices was 10 ft during the first run, and increased to 30 ft for the second run.

Figure 4.6 illustrates AEDP reactively changing the wakeup threshold based on runtime conditions. During the first experimental run, the receiver mote quickly increases the wakeup threshold to −56 dBm to avoid false wakeups introduced by the nearby 802.11 interferer. At this point, the mote is still unable to meet the application-specified duty cycle, and hence the threshold remains at about −55 dBm for the remainder of the experiment. In the second experimental run, the receiver mote likewise quickly increases the wakeup threshold to −54 dBm. At this point, because the mote is located further away from the interferer, it is able to meet the application-specified duty cycle; hence, it gradually decreases the wakeup threshold in increments of 2 dBm. AEDP eventually settles on a threshold between −60 and −62 dBm that closely matches the requested duty cycle, where it remains for the remainder of the experiment.

### 4.7.2 Adaptation to Network Changes

To test AEDP's resilience to network changes, we performed an experiment where additional transmitters were added to the network at runtime. We initially deployed a single transmitter mote and a single receiver mote. A second transmitter was added to the network 21 minutes into the experiment, and a third was added at 41 minutes. All three transmitters were

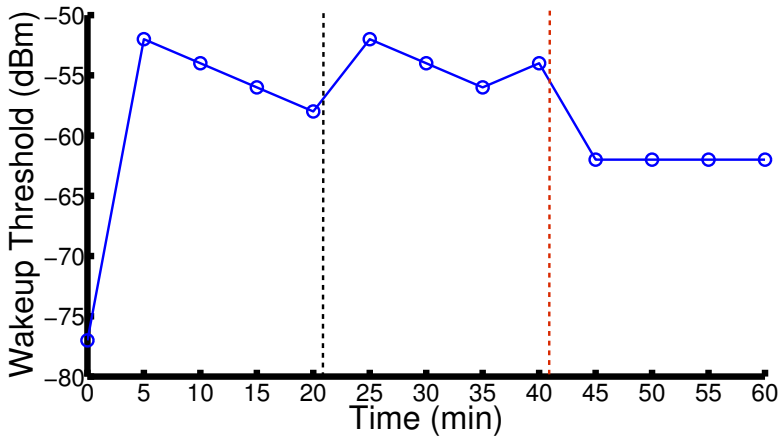



Figure 4.7: AEDP adapting the wakeup threshold over time when new nodes join the network. A second transmitter joined into the network at 21 minutes (vertical black line) and a third at 41 minutes (vertical red line).

configured to send packets to the single receiver node, where we instrumented AEDP to record its wakeup threshold over time.

Figure 4.7 illustrates how AEDP adapts the receiver's wakeup threshold over the course of the experiment. In order to reduce the false wakeup rate, AEDP quickly increases the wakeup threshold to $-52$ dBm; this closely matches the $-50$ dBm RSS of the first transmitter. After AEDP reaches its objective false wakeup rate, it begins steadily decreasing the threshold until the second transmitter joins at 21 minutes. The second transmitter's signal strength is slightly higher ($-46$ dBm) than the existing transmitter. Hence, AEDP responds to the new node by increasing the threshold to $-52$ dBm, slightly lower than the minimum of both transmitters, and again gradually decreases the threshold over time. At 41 minutes, the third transmitter joins with a significantly lower signal strength at the receiver ($-60$ dBm) than the previous two transmitters. Benefiting from the periodical wakeup threshold reset process mentioned in Section 4.5, AEDP adapts by rapidly dropping the wakeup threshold to $-62$ dBm, again slightly below the minimum signal strength of all the transmitters. These results demonstrate AEDP dynamically adjusting the wakeup threshold to successfully accommodate network topology changes.

### 4.7.3 Impact on Duty Cycles

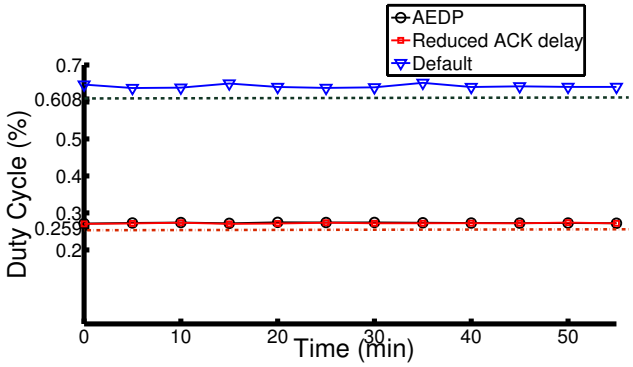
To explore AEDP's impact on duty cycles, we deployed a pair of motes with a modified radio stack to record the *radio on time* — i.e., the cumulative time the radio was active — on each mote. The precise duty cycle is hence derived from the radio on time and the experiment's length.

We first deployed the pair on channel 26 in an office environment, which we confirmed to be clean with a Wi-Spy spectrum analyzer [1]. We performed experimental runs, for 60 minutes each run, once with the default BoX-MAC-2 configuration and once with AEDP. To isolate the effects of the reduced ACK delay (discussed in Section 4.6.2) from AEDP's wakeup threshold tuning, we performed a third experimental run which reduced the ACK delay but was otherwise identical to the default BoX-MAC-2 stack.

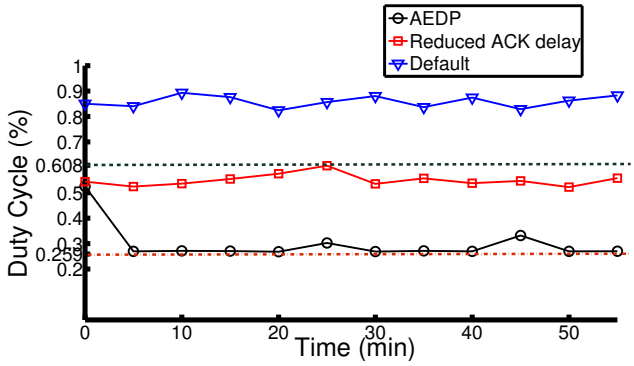
Figure 4.8(a) presents the duty cycle under all three experimental runs, broken down into 5 minute windows. In each 5-minute window, the default BoX-MAC-2 configuration activates the radio with an average duty cycle of 0.64%. AEDP consistently reduces this duty cycle over the entire experimental run, by an average of 57.48%. In this clean environment, the false wakeup rate is very low; hence, AEDP achieves a duty cycle within 99.7% of the reduced-ACK configuration.

For comparison, we also plot the theoretical optimal duty cycle for both ACK delay configurations. Specifically, at a data rate of 1 packet/5 min and a wakeup interval of 2 s, the optimal duty cycle is $149 * T_{idle} + (T_p + T_i)/2 + T_p + T_d$, where $T_{idle}$ is the time the radio is active when no energy is detected (11.5 ms under the default configuration, or 4.5 ms with the reduced ACK delay); $T_p$ is the time needed to receive a packet (4.24 ms); $T_i$ is the gap between packets (8.3 ms under the default configuration, or 2.8 ms with the reduced ACK delay); and $T_d$ is the time the radio remains active after receiving a packet (100 ms). Because interference was limited, all experimental runs remained within 7% of their respective optimal duty cycles.

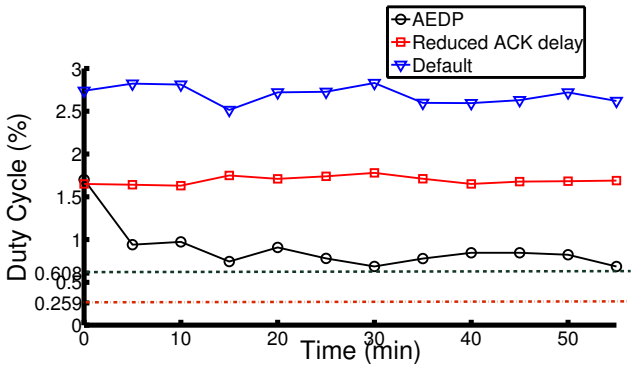
To evaluate AEDP's performance under a more typical deployment, we repeated this experiment in a residential setting. This experiment was carried out under normal wireless condition with residents' regular wireless activity. The mote pair is configured to use channel 16, which overlaps with the residents' 802.11g network. Figure 4.8(b) plots the results




(a) In a clean environment.




(b) In a residence with residents' normal activities.




(c) In a lab stress test with generated 802.11n interference.

Figure 4.8: Duty cycle under minimum interference, normal residential activities, and sustained interference. Horizontal lines indicate the theoretical optimal duty cycles of 0.259% (AEDP and reduced-ACK configurations) and 0.608% (default radio configuration).

under this experimental setup. We observe that the adjusted ACK delay is responsible for a significant reduction in radio usage, with the average duty cycle in each 5-minute window dropping from 0.86% to 0.55%. However, in the face of typical wireless noise, AEDP's wakeup threshold adjustment has a significant impact on duty cycle. AEDP reduces the duty cycle to an average of 0.30%, resulting in a savings of 45.5% over the tweaked radio stack and 65.1% over the default radio configuration.

Because AEDP is largely able to avoid false wakeups, it comes within 15.8% of the theoretically optimal duty cycle. In contrast, the default and reduced-ACK stacks achieves a duty cycle 41.4% and 112.4% higher than their respective optimal duty cycles.

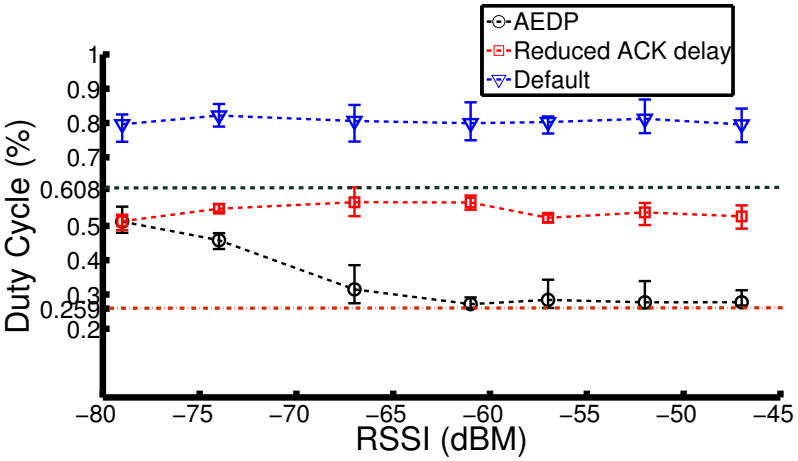
As a stress test, we repeated the experiment once more in a lab setting under controlled interference, in the form of a laptop and an access point, located 10 ft from the mote pair, generating 1 Mbps UDP traffic over an overlapping 802.11n channel 6, which overlaps with 802.15.4 channel 16.

Figure 4.8(c) plots the duty cycle under this controlled experiment. Due to the persistent source of interference, the default stack has an average duty cycle of 2.69% while the reduced-ACK stack has an average duty cycle of 1.69%. In contrast, AEDP achieves a duty cycle of 0.89%, a 47.3% reduction over the reduced-ACK stack and 66.9% over the default stack.

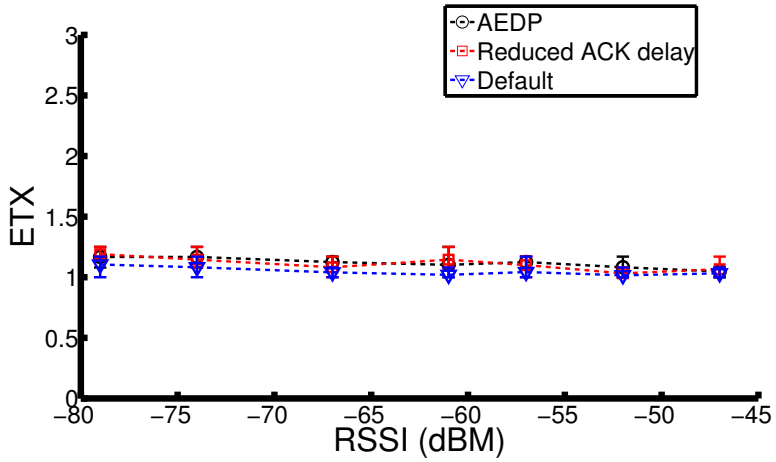
Owing to the challenging nature of the wireless environment, all three stacks perform several times worse than their theoretical optimal duty cycles. However, AEDP comes within the closest of its optimal duty cycle: 244% higher than optimal, compared to 342% for the default stack and 552% for the reduced-ACK stack.

### 4.7.4 Effects of Signal Strength

We explored AEDP's performance on a diverse set of links by selecting 30 links at random from the 380 links detected in a testbed we deployed in a 3-floor residential apartment building. This experiment was carried out under normal wireless condition with four residents' regular wireless activity. As with the previous experiment, we performed three runs, for 60 minutes each experimental run: one with the default LPL configuration, one with a reduced ACK delay, and one with AEDP.




(a) Duty cycle.




(b) Average ETX.

Figure 4.9: AEDP's performance on links with diverse signal strengths.

For the purposes of presentation, we group the 30 links into 7 buckets based on their signal strength, using buckets 5 dBm wide. As shown in Figure 4.9(a), these links show highly diverse RSS at their respective receivers. For the strongest links (RSS $\in (-65, -45]$), AEDP achieves a duty cycle of 0.28%, close to the theoretical minimum of 0.259%. This represents a 40.3% reduction over the reduced-ACK configuration and 65.1% over the default LPL configuration.

AEDP shows a more moderate — but still significant — improvement in duty cycle on intermediate links (RSS $\in (-75, -65]$). For these links, AEDP achieves a 31.2% reduction in duty cycle over the reduced-ACK configuration and 52.6% over the default LPL configuration.

For the links with the lowest signal strength (RSS $\leq -75$), the RSS is already close to the radio stack's default wakeup threshold of $-77$ dBm. AEDP cannot adjust the wakeup threshold below the signal strength, since it sets $T_{max}$ to be the minimum RSS of incoming links to avoid sacrificing network reliability. Hence, AEDP's 35.7% reduction in duty cycle is attributable only to the reduced ACK length.

As shown in Figure 4.9(b), the reduced-ACK configuration and AEDP introduced a small number of false-negative energy detection checks which were not experienced under the default stack, since the number of CCA pin polling was reduced from 400 times to 115 times, as discussed in Section 4.6.2. The reduced-ACK configuration consequently had a 5.5% increase in average ETX (from 1.05 to 1.11) and AEDP had a 6.7% increase in average ETX (from 1.05 to 1.12). The slight increases in average ETX are in exchange for a proportionally much-larger reduction in duty cycle.

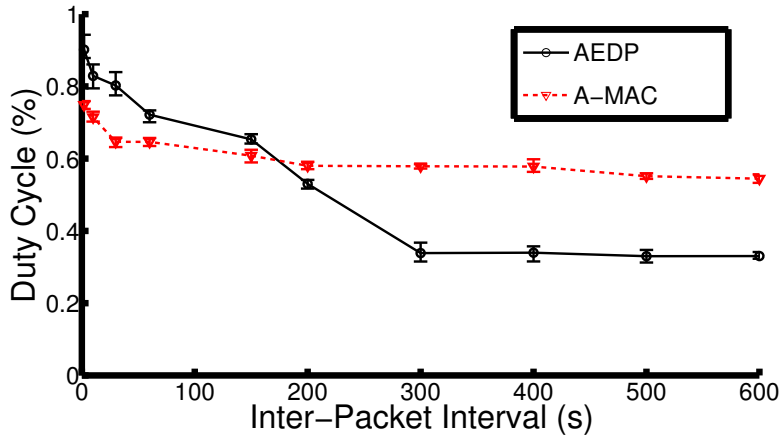
We note that links with the lowest signal strength tend to be highly bursty; while productive for routing, they must be used opportunistically. While AEDP will neither help nor hurt when such links exist, by their nature this will only happen for short bursts during the application's lifetime. During the periods where moderate-to-strong links are used for routing, AEDP will dynamically increase the wakeup threshold, resulting in significant energy savings.

### 4.7.5 Comparison with A-MAC

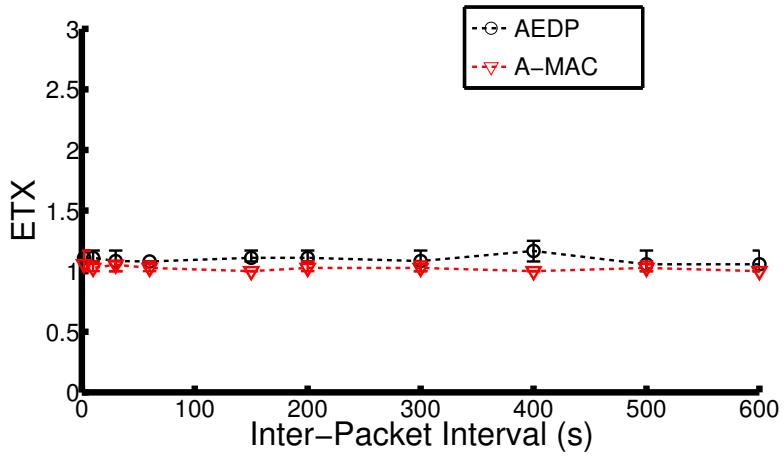
Receiver-initiated MAC protocols [32, 118] avoid the false wakeup problem by transmitting probe packets when nodes are ready to *receive* data, eliminating the need for recipients to actively sample the channel. Although AEDP and receiver-initiated MAC protocols approach the false wakeup problem from different directions, they share the same goal of extending network lifetime by reducing duty cycle in the face of noisy wireless channels. To understand the effectiveness of these two approaches, we performed a set of experiments comparing AEDP's performance with that of A-MAC, a state-of-the-art receiver initiated MAC protocol [32]. For this set of experiments, we choose the same set of links from the residential testbed used in Section 4.7.4, and configured the transmitters to transmit at data rates ranging from 1 packet/2 s to 1 packet/600 s. We performed each experimental run twice, once with AEDP and once with the A-MAC implementation provided by the authors of A-MAC [32]. A-MAC's radio stack was instrumented to record the radio on time, but was otherwise set to its default configuration. For fairness, we used the *default* parameters for both BoX-MAC-2 in TinyOS 2.1.1 [4] and A-MAC provided by the authors [32]. The only change we made for BoX-MAC-2 is reducing the ACK delay because of the implementation flaw discussed in Section 4.6.2.

As shown in Figure 4.10(a), at low data rates (Inter Packet Interval (IPI) within $[300, 600]$ s) AEDP leads to lower duty cycles than A-MAC. For instance, AEDP achieves an average duty cycle of 0.338%, representing a 41.5% reduction over A-MAC (0.578%) when IPI is 300 s. AEDP and A-MAC achieve similar duty cycles at intermediate data rates (IPI within $[100, 200]$ s). In contrast, at high data rates (IPI $\leq$ 100 s), AEDP leads to a higher duty cycle than A-MAC. For instance, with an IPI of 30 s AEDP achieves an average duty cycle of 0.803%, which is 24.1% higher than A-MAC (0.647%). As shown in Figure 4.10(b), AEDP introduced a small number of false-negative energy detection checks leading to an up to 16.7% increase in average ETX (from 1.000 to 1.166 when IPI is 400 s) in exchange for a proportionally much-larger reduction in duty cycle at low data rates.

The protocols' respective advantages at different date rates may be understood by analyzing their respective strategies. Under LPL, senders repeatedly transmit long preambles indicating that they are ready to send data; recipients periodically sample the channel, and turn on the radio if energy is detected. Under receiver-initiated MACs like A-MAC, recipients




(a) Duty cycle




(b) Average ETX.

Figure 4.10: Comparing AEDP and A-MAC with different inter-packet intervals (IPIs)

periodically broadcast beacons announcing that they are ready to receive data; senders keep their radios on waiting for the recipient's beacon, and then immediately ACK it. In principle, receiver-initiated MACs replace LPL's short channel sampling with an entire transmission plus a short delay waiting for a response. As discussed in Section 4.6.2, the default BoX-MAC-2 configuration suffers from an unnecessarily high channel sampling cost of 10.0 ms; in comparison, A-MAC pays a probing cost of 6.2 ms under our oscilloscope measurement. Consequently, previous literature has found that the overhead of receiver-initiated MAC protocols can be even lower than LPL [32]. However, after tuning the energy detection length, AEDP pays a significantly lower sampling cost of 2.9 ms. We note that receiver-initiated MACs *inherently* must pay the overhead of an entire packet transmission; hence A-MAC's overhead cannot be tuned in this fashion.

Thus, A-MAC has a higher overhead than AEDP at low data rates. However, since A-MAC saves the cost of sending a long preamble, it is able to outperform AEDP at sufficiently high data rates. This result suggests that AEDP is more suitable for low date rate applications, while A-MAC has advantages in high data rate applications. They therefore represent complementary approaches in the design space of low-power MAC protocols in noisy environments [8].

### 4.7.6 Collection Tree Protocol Performance

Finally, we study how well CTP protocol [42] performs over AEDP. Since AEDP is implemented as a layer situated on top of LPL BoX-MAC-2 MAC layers, running CTP over AEDP is largely a matter of changing configuration wirings. To explore the performance on a large scale, multi-hop networks, we run the experiments on an indoor testbed consisting of 55-TelosB motes in Jolley and Bryan Hall at Washington University in St. Louis [125]. Figure 4.11 shows the network topology with transmission power of 0 dBm. Each node produces data at a rate of 1 packet every 5 minutes and all data packets are forwarded to a sink node. We performed two 24-hour experimental runs one with the AEDP and the other

[8]The pTunes project [135] shows that the performance of MAC protocols are sensitive to their parameters. Optimizing parameters of a MAC protocol is not the focus of this Chapter. The pTunes system does not support TinyOS and hence cannot be used to select the MAC parameters for our experiments. Nevertheless the experimental study presented in this subsection reveals the general trend of the complementary behavior of AEDP and a receiver-initiated MAC when facing different data rates.

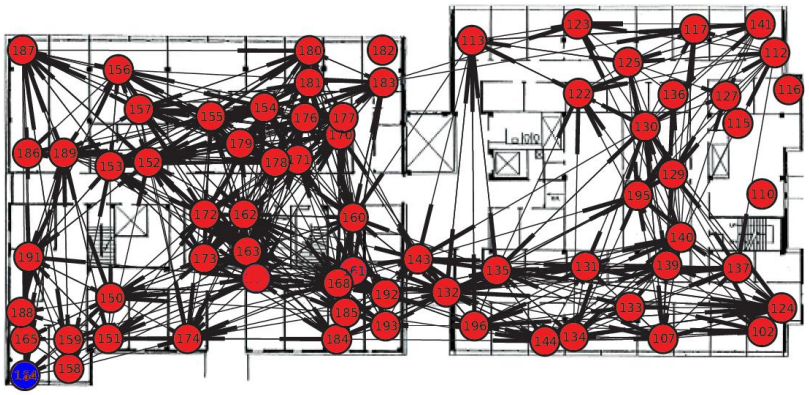
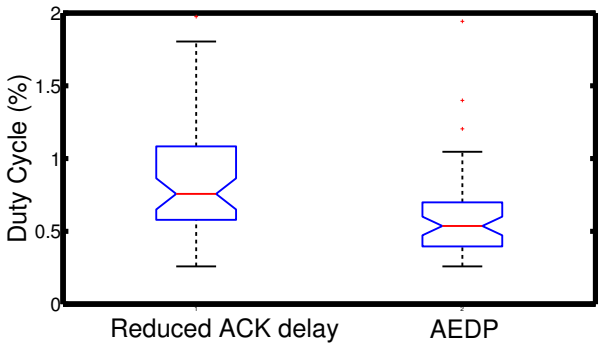


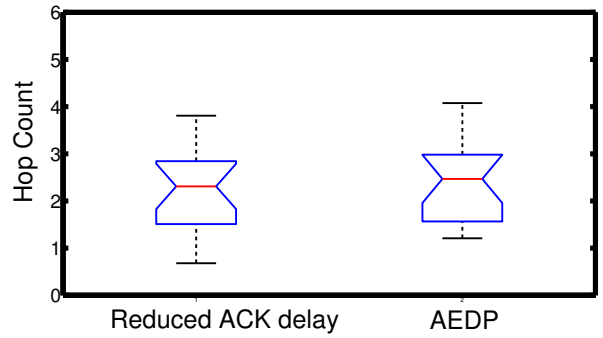
Figure 4.11: The Testbed topology with a transmission power of 0 dBm. Blue node is a sink node.

with LPL BoX-MAC-2 configuration with the reduced ACK delay. We use the default CTP setting in both two runs. To test the network's performance in a noisy environment, we set the nodes operating on channel 18 overlapping with the campus Wi-Fi channel.

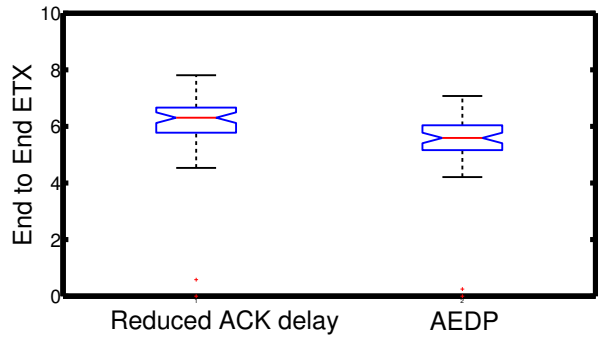
Figure 4.12(a)- 4.12(c) show the box-plots of the duty cycles of all nodes in the testbed and the average hop counts and end-to-end ETX of the routes of all nodes. Since the routes of nodes may change dynamically under CTP, for each node we calculates the average values of hop count and end-to-end ETX during each 24-hour experimental run. As shown in Figure 4.12(a) and 4.12(c), AEDP reduces the median duty cycle by 35.44% (from 0.79% to 0.51%), while also reducing the median end-to-end ETX by 11.26% (from 6.30 to 5.59). This result shows that AEDA is able to mitigates the impacts of noise on LPL on node duty cycles while simultaneously reducing the multi-hop transmission cost under CTP. As shown in Figure 4.12(b), AEDP does result in a slight increase in the median hop count of the routes (from 2.30 hops to 2.46 hops) as a result of a higher CCA threshold used to filter out noise. The combination of a lower end-to-end ETX and higher hop counts indicate that AEDP was able to filter out weak links affected by noise while still enabling CTP to take advantage of enough good links for low-cost multi-hop communication.




(a) Duty cycle.




(b) Hop count.




(c) End-to-end ETX.

Figure 4.12: Box-plot comparison between AEDP and LPL BoX-MAC-2 with reduced ACK delay. Central mark in box indicates median; bottom and top of box represent the 25th percentile ($q_1$) and 75th percentile ($q_2$); crosses indicate outliers ($x > q_2 + 1.5 \cdot (q_2 - q_1)$ or $x < q_1 - 1.5 \cdot (q_2 - q_1)$); whiskers indicate range excluding outliers.

## 4.8 Conclusion

Maintaining energy efficiency in noisy environments has become an increasingly critical problem as wireless sensor networks are gaining widely deployment in residential and office environments. While LPL has been a popular and effective approach to energy-efficient MAC protocols, false wakeups caused by wireless noise can significantly increase the duty cycle and compromise the benefit of LPL. To address this problem, we first perform an empirical study of the false wakeup problem of LPL in real-world residential environments and find that the CCA wakeup threshold is an effective knob for controlling false wakeups. We then propose AEDP, an adaptive protocol that dynamically adjusts a node's wakeup threshold to improve network reliability and duty cycle based on application-specified bounds. AEDP has been implemented on TinyOS 2.1.1 and the TelosB platform. Experimental results from both real-world residential deployments and testbed experiments show that AEDP can effectively maintain low duty cycles in noisy environments and adapt to network changes and links with varying signal strength. We also found AEDP and A-MAC more energy-efficient for applications with low data rate and high data rate, respectively, and therefore provide complementary approaches suitable for different classes of applications.

There are two limitations to AEDP. First, tuning CCA threshold is ineffective for links with low signal strength that can be close to or below the signal strength of noise. In this case AEDP will set the wakeup threshold to the minimum RSS of incoming links and as a result cannot effectively reduce false wakeups caused by noise. This makes AEDP less effective for highly sparse networks connected by mostly long links. Second, our implementation is specific to the particular CC2420 radio stack used. It is important to develop a general methodology — such as the analysis performed in Section 4.6.2 — to select the key radio and MAC layer parameters. For a new radio stack, developers should firstly measure the ACK delay and then tune the duration of CCA polling accordingly. In general, the duration of CCA polling must be longer than the ACK delay to avoid false negatives in energy detection which can heavily worsen the performance. On the other hand, a long energy detection contradicts the need for a short, inexpensive energy detection, and arguably even renders the entire check ineffective. Therefore, the duration of CCA polling should be slightly longer than the ACK delay.

# Chapter 5

# Self-Adapting MAC Layer for Wireless Sensor Networks

## 5.1 Introduction

Research efforts in the last decade produced numerous MAC protocols for wireless sensor networks (WSNs). Many of the MACs were designed to achieve low latency, high throughput, low power consumption, or robustness to interference. However, none of the existing protocols deliver optimal performance in all desirable dimensions under varying environmental conditions. For instance, sender-initiated low-power listening (LPL) protocols [81] are power-efficient in absence of ambient noise, but suffer high power consumption due to false wake ups in noisy environments [105]. Receiver-initiated MACs are resilient to interference, but incur higher overhead at low data rates in a clean environment. TDMA protocols can deliver high throughput for high data rate applications by avoiding channel contentions, whereas CSMA/CA protocols can achieve low latency for low data rate applications.

A fixed MAC protocol therefore cannot meet the demands of varying workloads, diverse Quality of Service (QoS) requirements, or changing environmental conditions. This problem is exacerbated with the increased interest in connecting smart phones and wireless sensors placed on the user's body or in the surrounding environment. The fusion of a smart phone and a network of motes opens up opportunities for novel and exciting applications (e.g. fall detection, vital sign monitoring, and fitness assessment), while also introducing the fundamental challenge of maintaining optimal wireless communication between mobile phones and sensors under varying conditions and demands.

1. **Wireless Environment**: The wireless environment changes when the user moves around. At times WSN will need to be able to deal with a highly noisy environment; at other times it may enjoy a clean environment. For example, Bluetooth devices, our own and our neighbors' Wi-Fi access points, and our microwaves – all generate interference that interacts with WSN. This interference is unruly in homes and more orderly in office buildings [106]. A resilient MAC protocol may be required in noisy environments, while a different MAC may be more efficient in clean environments;

2. **Network Traffic**: The network traffic is subject to spontaneous changes. For example, in a wireless health application, the wireless sensors may produce a low amounts of data during some hours of the day, but sporadically, in response to a critical medical condition, require rapid transmission of a large volume of data. Moreover, different sensors have different traffic patterns and a system may turn ON or OFF any one of the sensors at any given time. For instance during stable periods, heart rate sensing may occur every minute and the data being sent is typically a single integer. But if the medical condition changes, the application may activate the pulse sensor continuously, or decide to activate ECG sensors. The data rate can increase from less than 1 to 750 bytes per second [34];

3. **QoS Requirement**: The application QoS requirements may also change. While it may be reasonable to lose a packet sporadically during clinically-stable periods, it may not be acceptable during imminent clinical deterioration.

Given the dynamic nature of communication between mobile phones and sensors, a traditional one-MAC-fit-all approach cannot meet the challenges under significant dynamics in operating conditions, network traffic and application requirements. To fill this need we design the *Self-Adapting MAC Layer (SAML)* that makes available multiple MACs in an efficient manner and selects the protocol most suitable for the current conditions and requirements. SAML comprises two key components.

1. *Reconfigurable MAC Architecture (RMA)* that supports dynamic switching among different MACs. RMA holds multiple MACs without bloating its memory footprint due to its modular design based on shared components;

2. *MAC Selection Engine* with a machine learnable model that optimally selects MAC protocols in terms of reliability, energy consumption, latency, and resiliency against ambient noise.

We have implemented SAML in TinyOS 2.x on the TelosB platform and a gateway device that we developed to integrate an Android smart phone with a 802.15.4 radio. We have validated the efficacy of SAML and the efficiency of its operation by measuring the memory footprint and the overhead of key operations. We have also performed a four-day real-world case study in which SAML provides efficient and reliable MAC switching, while saving 31.6% of energy compared with a fixed MAC layer and meeting the QoS requirement of the application.

The rest of the chapter is organized as follows. Section 5.2 reviews related work. Section 5.3 presents the overview of our system architecture. Section 5.4 describes the design of RMA and Section 5.5 shows how we realized RMA in TinyOS. Section 5.6 describes our MAC Selection Engine and Section 5.7 presents experimental results. Section 5.8 concludes the chapter.

## 5.2 Related Work

Many approaches have been proposed to reconfigure a WSN by disseminating code to the nodes. Hui et al. [54] proposes a reliable dissemination protocol that distributes an entire TinyOS image compiled with a new MAC protocol, across the network and replaces the current running image on the nodes across the network by reprogramming them. By distributing an image, nodes benefit from having only the pieces of software that they require at a given time, but the network pays a large communication overhead during the distribution, and the nodes incur a large shut down and load times. Many efforts have been made to reduce the communication overhead. Marron et al. [77] proposes an adaptive cross-layer framework that selects and disseminates only new fragments of code instead of an entire image. Mottola et al. [82] uses a reconfiguration programming model to identify a subset of nodes that should be reconfigured, avoiding flooding to the entire network for each image distribution. Tavakoli et al. [119] designs an interval-cover graph to minimize communication

redundancies between multiple applications on shared sensors, while Gauger et al. [40] designs an approach to exchange and relink nesC components by defining an uniform external interface for all changeable nesC components. In contrast to these dissemination approaches, our research investigates the efficiency and effectiveness of hosting multiple MACs and enabling much more dynamically switching between them at runtime. Our measurements of code size increase of TinyOS resulting from making available numerous MACs shows that it is only marginally larger than an OS image containing a single protocol. Our measurements also suggest that the resulting code size is a non-linear, decreasing function of the number of MACs included in RMA.

Recently, Zhang et al. [131] designs a toolchain to enable the MAC reconfiguration by relinking MAC components at runtime. This design takes a different approach than our RMA: a computer is used to compute a MAC description, and a parser processes the description and generates a new MAC based on preloaded MAC components of the sensors participating in the network. The MAC is then sent to the sensors through radio communication. MAC switching therefore requires more bandwidth than RMA. Furthermore, the user is responsible for initiating the work on the computer side to issue the MAC switching command, while RMA autonomously determines which protocol is optimal for a given scenario at runtime and performs the switch automatically.

The research community has a growing interest in designing hybrid MACs to combine some of the advantages of different MAC protocols. Z-MAC [93] allows nodes to compete for the channel within unassigned TDMA slots. Funneling-MAC [12] allows nodes close to the sink to run TDMA schedules while all other nodes follow a scheduled contention or polling based duty cycle. However, these hybrid MACs provide a limited set of features that are decided at design time, and more features lead to higher degrees complexity in the protocol design. There are also growing interest in adding adaptation to a single protocol at runtime. IDEA [24] and MaxMAC [55] proposed to extend battery lifetime and accommodate bursty traffic demand by adjusting the wakeup interval of LPL BoX-MAC. pTunes [135] allows for runtime adaptation of MAC parameters of a single protocol. In contrast to the hybrid MACs, which provide limited number of features, and parameter tuning approaches, which optimizes the parameters of a single MAC – our work supports switching between various entire MAC protocols , while enjoying the full benefit of a specific protocol to given conditions. Our

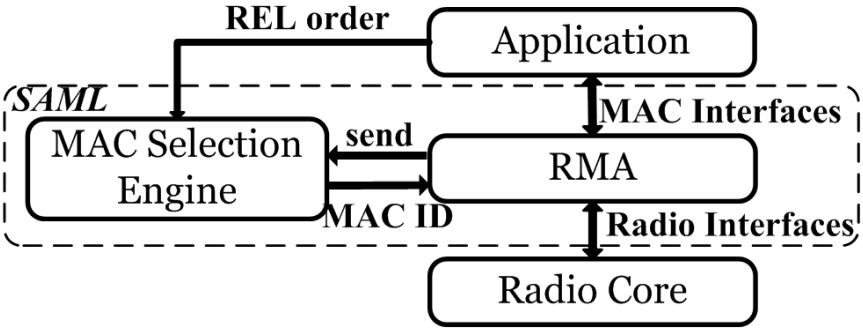



Figure 5.1: Overview of System Architecture.

approach therefore offers applications a richer optimization space and complements existing optimization techniques designed for individual protocols.

Adaptive MACs were also researched in relation to IEEE 802.11 networks. Huang et al. [53] develops an adaptive MAC protocol which can select between multiple MACs. Farago et al. [35] proposed to dynamically combine a set of existing MACs into a single layer. The major difference between these 802.11 dynamic MAC frameworks is that our work is designed to minimize runtime overhead and memory footprint, factors which are critical in low power and resource constrained sensor networks. Our measurements and empirical results suggest that our approach of component reuse and our design of MAC switching protocols effectively reduce static code size and runtime overhead. To the best our knowledge, RMA is the first reconfigurable MAC architecture that allows dynamic MAC switching for low power wireless sensors.

We introduced the high-level concepts of SAML and a proof-of-concept integration of two basic MACs in [61]. Comparing with [61], this chapter presents the complete design of our Self-Adapting MAC Layer comprising Reconfigurable MAC Architecture, introduces the new MAC Selection Engine, and provides a systemic experimental evaluation.

## 5.3 Overview of System Architecture

In this section, we present the overview of the system architecture. Comparing with traditional architecture, we replace the MAC layer with the **RMA** and add the **MAC Selection**

**Engine** as shown in Figure 5.1. RMA stores multiple MACs and supports switching between them at runtime. The MAC Selection Engine is the component responsible for recommending the best MAC according to the specified QoS requirements, monitoring the dynamic ambient conditions, and automatically responding (without the need for the application to manage this process) to changes in the environments. Once a new MAC was determined by the engine, it will send the MAC ID to the RMA.

One of the primary design goals of SAML is to be transparent to its users. For this purpose, SAML exposes a set of unified interfaces to the applications using it and to the lower radio layer. Applications can treat SAML as a traditional single MAC entity and do not need to manage any aspect of the MAC switches occurring in SAML. Five interfaces are available to applications: (1) Initialize and start/stop the MAC layer; (2) Control the radio CCA and backoff policies; (3) Send; (4) Receive; and (5) Specify the QoS requirements of the application. Interfaces (1)-(4) were shown to be enough to perform regular MAC operations [64] and (5) is provided for applications to specify the QoS requirements as an 3-tuple that orders Reliability (R), Energy consumption (E), and Latency (L) in terms of their relative importance to the application. This is named the **REL order** of the application. The REL order can be updated by the application during runtime to accommodate dynamic changes in its mode of operation. For example a duty-cycled clinical monitoring application which ranks energy consumption over reliability and reliability over latency will simply specify E > R > L to achieve longer battery life. However, during a clinical deterioration period above REL should probably be changed to R > L > E when the application becomes reliability-critical.

For the radio interfaces, SAML adopts the same low-level interfaces suggested by [64]. These interfaces are platform independent, rather than specific to a particular radio or microprocessor, enabling portability between different hardware platforms.

We will describe the general design of the RMA in Section 5.4 and then present how we realize it in TinyOS in Section 5.5. We will show the design of MAC Selection Engine in Section 5.6. We believe the architecture and design principles of RMA and MAC Selection Engine are applicable to other OS platforms. To enable the communication between an Android smart phone to communicate with a 802.15.4 radio, we have developed a Gateway device. The design of the Gateway can be found at [102].

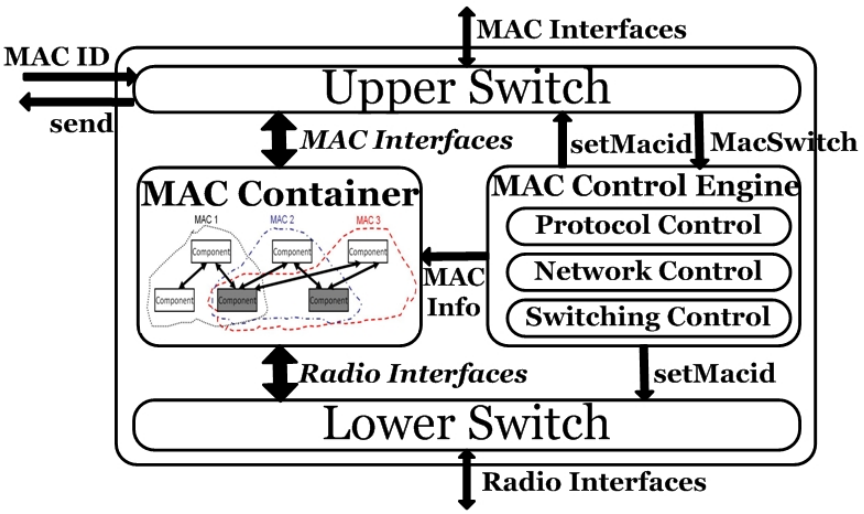



Figure 5.2: RMA Architecture.

## 5.4 RMA Architecture

In this section, we present the RMA architecture. Our design addresses the following goals: (1) The architecture needs to present a unified set of interfaces to both its upper layer application users and lower layer radio developers. (2) RMA should reliably switch MACs while maintaining consistency between different nodes in the network. (3) The incorporation of multiple MACs should not significantly increase the memory footprint of the MAC layer. (4) Switching MACs should incur only a small runtime overhead in term of both CPU cycles and communication bandwidth.

### 5.4.1 Overview of the Architecture

As Figure 5.2 shows, RMA has four major modules. The **MAC Container** stores the MACs which are available at runtime. We leverage a component based architecture to reduce the memory footprint by allowing multiple MACs to share components (Section 5.4.2). **Upper Switch** and **Lower Switch** provide a unified set of interfaces to application and lower radio core; abstracting away the details of the MAC layer (Section 5.4.3). **MAC Control Engine** controls the identification of the active MAC, manages the neighborhood table,

and supervises protocol switching when it receives reconfiguration request from the MAC Selection Engine (Section 5.4.4).

### 5.4.2 MAC Container

The MAC Container stores the MACs which are available at runtime. The design challenge we were facing is to enable multiple MACs without incurring a prohibitive increase in the code size. The naive approach would be to have the container encapsulate multiple monolithic MACs, resulting in a code size that is the sum of the size of all individual MACs; but this would exhaust the onboard memory very quickly. An alternative approach is to distribute the new MAC wirelessly at runtime during switching; but this would incur long latency, consume the network bandwidth, and spend precious energy on radio communication. The problem is exacerbated for mobile WSNs, such as a Body Sensor Network, since the MACs may need to change in a time scale of minutes due to the frequent changes in the environments or application needs.

MACs share many common functions and can be distilled to a set of reusable components [64]. To support a variety of MACs with minimal penalty to code size, our MAC Container is designed to hold these reusable components, from which MACs are built. The components inside the MAC Container can be (re)wired in different ways to construct various MACs. Our experimental results in Section 5.7.1 show that this component based approach only moderately increases the code size, when comparing RMA supporting multiple MACs even code using only a single MAC.

### 5.4.3 Switches

The Upper Switch and Lower Switch are two important design constructs of RMA. Their main purpose is to enable efficient (and possibly frequent) dynamic routing between components. All MACs within the MAC Container provide and use a uniform set of interfaces to the layer above and below the MAC layer. The switch is responsible for routing commands and events through the interfaces provided by the currently active MAC, using only a single variable to identify the active MAC. This variable is used as a *select* signal, determining

which MAC is going to respond. This technique allows very quick protocol changes relative to many other alternatives such as dynamic loading of code functions, thread switching, and so on.

### 5.4.4 MAC Control Engine

The MAC Control Engine implements RMA core logic. It is designed to facilitate the identification of the active MAC, manage the network topology, and supervise the MAC changing process. It includes three major units:

#### Protocol Control

Protocol Control keeps track of the active MAC and makes sure all the components in the MAC Container are synchronized to the same MAC. The components are shared between different MACs and can be (re)wired in different ways to construct various MACs. The names of the components are maintained in a list in Protocol Control to address the components during MAC switching. When a MAC change occurs, Protocol Control treats the change process as a transaction. Either all of the updates occur and the protocol is changed, or the transaction will be rolled back.

#### Network Control

The Network Control unit manages the network topology. A node is specified by the application as a coordinator, which is responsible for nodes joining and leaving the network. In mobile WSNs, due to changing interference levels for example, this can occur frequently. The smart phone is usually a natural choice for the coordinator. To indicate the current MAC used by the network the coordinator disseminates an announcement packet, which includes the active MAC's identification (MAC ID) in each period $T_1$. If the current MAC uses beacons, RMA inserts a single byte into existing beacons instead of generating new ones to reduce communication overhead. For instance, RMA can use the time synchronization beacon of TDMA-based MACs.

Before joining, a new node does not know which MAC is in use by the network and thus cannot join or talk with the rest of the network. For this purpose, we design a **baseline-MAC** which all nodes except the coordinator implement by default as part of RMA. The coordinator does not need this baseline-MAC and can run any MAC it wants when it boots, since all other nodes will synchronize their MAC to the coordinator. The baseline-MAC is designed to allow the initial formation of a network and to allow nodes to join or rejoin an already running network. When a sensor node boots, it first runs the baseline-MAC, which turns on the radio and overhears all packets. After it catches a protocol announcement packet from the coordinator, it changes to this new protocol and sends a join-network request to the coordinator. When the coordinator receives the request, it adds the new node to its neighborhood table and allocates resources for this node.

As discussed above, new nodes, or nodes that lost connectivity to the network, may not know what MAC the network is currently running. Furthermore, at any given time the nodes need to know if they are connected to the network or not. We address these concerns by defining two time intervals: The coordinator needs to announce that it is present every $T_1$ seconds and nodes need to announce that they are connected every $T_2$ seconds. After joining the network, a regular node (non-coordinator) transmits a dummy packet to report alive each $T_2$ seconds, unless it happens to send data packets during this period. If the coordinator does not hear from a node (newly joined or one that was already in the network) within $5 \times T_2$ seconds it treats it as a node that left the network. It is removed from the neighboring table and all other resources assigned to it are deallocated. A regular node in the network changes back to the baseline-MAC and trys to rejoin if it does not hear from the coordinator within $5 \times T_1$ seconds. This simple protocol maintains synchronization across the network, while accounting for the highly dynamic nature of mobile networks.

It is worth noting that we choose the values, 5, empirically, as they provided good balance between giving up too soon on a network node and allowing for dynamic nodes to join and leave as they wish. However the $T_1$ and $T_2$ factors do not need to be equal and other values can be used.

**Switching Control**

When a coordinator receives a MAC ID from its MAC Selection Engine, it verifies that the new MAC is different from the current one and checks whether the requested MAC is stored in the MAC Container. If both conditions hold, the Switching Control unit notifies all sensor nodes in its neighborhood table and then performs the protocol change. It firstly refuses new packets issued by the application, waits for all existing packets buffered in the MAC layer to be transmitted, and then shuts down the current MAC orderly. The variable holding the active MAC ID will be updated, and then the new MAC will be started. If the new MAC has successfully started transmitting, then Switching Control unit returns a success to the requesting application. Otherwise, it rolls back to the old MAC and retries the change. This process is allowed to repeat 30 times before the protocol change request is discarded. We choose the constant 30 empirically; fewer attempts will consume less energy, but will result in higher chance of giving up.

## 5.5 Realization of RMA in TinyOS

In this section, we describe how we realize RMA in TinyOS 2.1.1. While RMA has been implemented in TinyOS, we believe its design principles are applicable to other OS.

### 5.5.1 MAC Container

The MAC container stores various components from which MACs are built. Our impetus in creating the abstraction of the MAC Container is to minimize code size, which is essential given the limited resources on motes. Notably, for TinyOS the nesC compiler only creates one instance for each non-generic component, no matter how many times it is used by different code segments [4]. RMA reuses the components from the MAC Layer Architecture (MLA) [64], which distill common features of different MACs to a set of reusable components. However, MLA was designed to facilitate the implementation of MACs at development time

and does not support MAC layer reconfiguration at runtime. Our MAC Container encapsulates MLA components and adds new mechanisms to enable (re)wiring components such that they support runtime reconfigurations.

In TinyOS allowing component sharing raises the typical fan-out issue of nesC, which is not a desirous effect for RMA. Here we briefly describe this issue, and a detailed explanation can be found in [69]. Fan-out: A single interface of a component (caller) is wired to two interfaces belonging to different components (callees). When the caller invokes a command on that interface, both callees will be invoked in an undefined order and may return different results.

When components are shared by different MACs they may be configured to perform different operations at different times. Only one MAC should run at a time and other MACs should stay inactive. Invoked by an command that was not meant for it, the code of inactive MACs may perform some unwanted operations, such as polling the channel, turning off the radio or starting an alarm, which may conflict with the active MAC. Our solution is adding a very lightweight **RMA Wrapper** that wraps each shared interface with a parameterized interface. In TinyOS, we use nesC parameterized wiring feature using the active MAC ID as the parameter. The code below shows the nesC code for a parameterized interface in such a RMA Wrapper.

```
module SharedComponent {
  //uses interface A;
  uses interface A[uint8_t id];
}
implementation {
  uint8_t currentMac;
  //call A.anycommand();
  call A.anycommand[currentMac]();
}
```

In each RMA Wrapper, a parameter is added on each interface declaration. A integer variable, "currentMac", indicates the active MAC ID. We use a local variable instead of a global variable to avoid potential race conditions. When a component calls a command

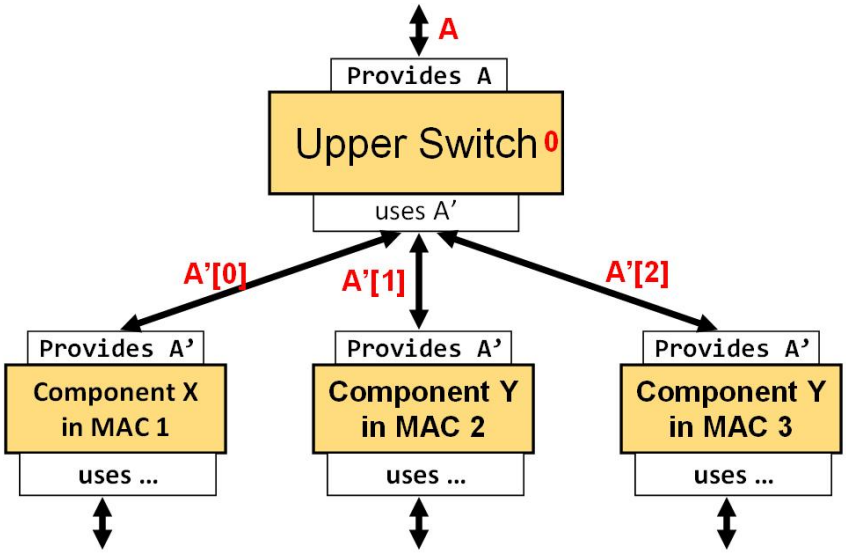



Figure 5.3: Parameterized wiring based Switch design.

through its parameterized interface, only the code of the current active MAC will be called. The same idea is applied when signaling an event. Each RMA Wrapper also needs to expose a `setMacid` interface for Switching Control unit to change the "currentMac" value. The MAC switching logic will be discussed in Section 5.5.3.

### 5.5.2 Switches

Similar to the design of RMA Wrapper, we use nesC parameterized wiring to implement the Upper and Lower Switches of RMA in TinyOS. Figure 5.3 shows the design of Upper Switch which provides one single set of interfaces to the MAC Selection Engine, while using multiple set of interfaces provided by different MACs. Hence connecting many MACs to a single consumer. The active MAC ID is used to select between, or enable, only one of the MACs at runtime. The code below shows the nesC code for a parameterized interface that achieves this.

```
module UpperSwitch {
  provides interface A;
  uses interface A'[uint8_t id];
}
```

```
implementation {
 uint8_t currentMac;
 command void A'.command(){
    call A'.command[currentMac]();
  }
}
```

In the Upper Switch, a parameter is added on each interface that the Upper Switch provides. At runtime the variable, "currentMac", indicates the current active MAC. As explained before this parameterized interface declaration makes the upper layer application calls invoke only the code of the current active MAC. Upper Switch expose a `setMacid` interface for Switching Control unit to change the "currentMac" value. Upper Switch initializes the MAC switching through the `MacSwitch` interface when a new decision is generated by the MAC Selection Engine and its value is difference from "currentMac". The MAC switching logic will be discussed in Section 5.5.3.

The same idea is applied in the design of Lower Switch, which allows many internal components of RMA to be connected to the single interface provided by the radio core. In the Lower Switch, a parameter is added on each interface that the Lower Switch uses to avoid the fan-out problem when signaling an event.

### 5.5.3 MAC Control Engine

#### Protocol Control

An integer variable named "currentMac" is maintained in all RMA Wrappers and Switches, indicating the active MAC ID. The Protocol Control unit uses an array to store the list of RMA Wrappers and it uses this list to treat a MAC switch as a transaction. Either all wrappers and switches will change their "currentMac" value to indicate the new protocol, or none of them will. In TinyOS, this is achieved using an atomic block.

### Network Control

Network Control, when running on a coordinator node, maintains a neighborhood table to keep track of the nodes in the network [9]. We reserve the first byte in payload packets for broadcasting the MAC ID. Nodes that did not join the network run the baseline-MAC and use this field to recognize the current MAC. RMA requires each MAC to provide a method for new nodes to join the network. For instance, we observed that some MACs cannot deal with network topology changes since they do not support methods for new nodes to join or leave the network. Pure TDMA implemented by [64] is such a protocol. It initially accounts for the nodes in the network and divides a fixed time window to slots for the number of nodes currently in the network. With all the time slots allocated to nodes, no new node can join. We accounted for that in our own version of pure TDMA by reserving one of the slots to an arbitrary new node. When more than one node is trying to join the network they will compete for this slot using a CSMA-style technique.

### Switching Control

When the coordinator decides to switch MACs it sends out the switching command to every node in its neighborhood, and then reconfigures its MAC layer to the new MAC. Similarly, every regular node switches to the new MAC after receiving the switching command. Since the delivery of a packet with switching command is critical for protocol synchronization across the network, the Switching Control unit on the coordinator node uses a single hop unicast with Automatic Repeat reQuest (ARQ) to notify the new MAC to each node in the network[1]. The coordinator treats a node as a one that left the network and will release its resources after 30 failed attempts.

Switching Control exposes to the Upper Switch the `MacSwitch` interface. When the `changemac` command is invoked by the Upper Switch, Switching Control unit calls the `stop()` command of the `SplitControl` interface to shut down the current MAC whose MAC ID is "current-Mac" value. It then changes "currentMac" values through the `setMacid` interface in each

[9] In principle, RMA can be extended to support multi-hop networks. Dissemination protocols (e.g. [72]) can be used to broadcast the active MAC ID or a switching command in a multi-hop network. Mechanisms to ensure agreement on the MAC among sensors within the network will need to be employed [20].

RMA Wrapper based on the list maintained in Protocol Control as well as the values in the Switches. After this is completed it invokes the `start()` command to turn on the new MAC, whose MAC ID is the new "currentMac" value.

The `stop()` command of the `SplitControl` interface is critical for RMA for performing MAC change. `stop()` command does not attract enough attention by MAC developers since it is not commonly used in a single MAC environment. When we reviewed the original implementations of MACs we found out that the `start()` command, which is used to start the MAC, is carefully implemented; but often the `stop()` command is vacuous or not carefully implemented. Sometimes developers forget to stop the timer or alarm which was started in the protocol's `start()` command or during the protocol execution. This may cause unreliable operation in a multi MAC environment. Therefore, we emphasize that all logic units started during protocol initialization and execution must be stopped within the implementation block of the `stop()` command.

### 5.5.4 Implementations

We obtained the source code of MLA, which works with TinyOS 2.1.1 from [79]. The source code includes the implementation of BoX-MAC [81] and pure TDMA. We also obtained the implementation of RI-MAC [118] from its authors [10]. As an exercise, we implement an adaptive TDMA based on pure TDMA (this protocol allows reconfiguration of TDMA frames at runtime) and a ZigBee MAC based on the standard [134].

The naming we used in the TinyOS RMA are as follows: We implemented `SwitchUpC` component as Upper Switch and `SwitchLowC` component as Lower Switch. The `MacC` component is the MAC Container and all the functionality in the MAC Control Engine are fulfilled by `ProtocolsControllerC`. The baseline-MAC is implemented in the `BaseLineC` component.

We have built three prototypes using our TinyOS implementation of RMA: a CSMA/TDMA prototype, a SI/RI-MAC prototype, and a 5-MAC prototype. The CSMA/TDMA prototype includes a CSMA/CA MAC (BoX-MAC) and a TDMA MAC (pure TDMA). The SI/RI-MAC prototype includes a sender-initiated MAC (BoX-MAC) and a receiver-initiated

[10]The authors' implementation is based on TinyOS 2.0.2, which we port to TinyOS 2.1.1

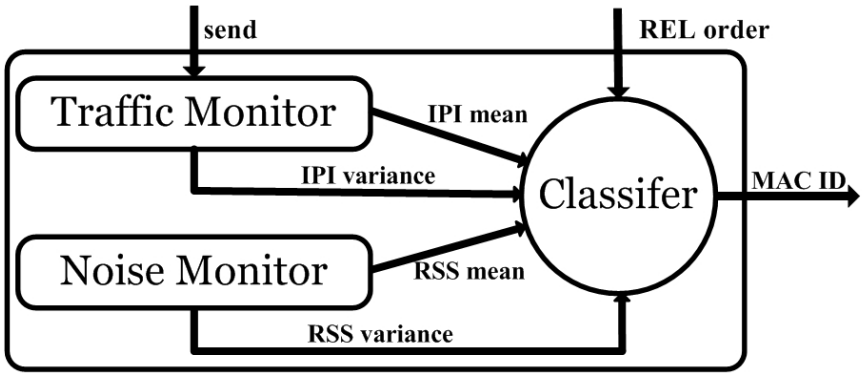



Figure 5.4: MAC Selection Engine.

protocol (RI-MAC). Lastly the 5-MAC prototype includes all five MACs (BoX-MAC, pure TDMA, RI-MAC, adaptive TDMA, and ZigBee MAC).

As a summary for future researchers we summarize the procedure of creating a RMA prototype:

1. Implement the MACs you desire using the reusable components provided by MLA; Make sure each MAC provides a method for adding new nodes to the network;
2. Identify the components which are shared by these MACs and wrap each of them with a RMA Wrapper (discussed in Section 5.5.1).
3. Assign each MAC a unique integer as its MAC ID;
4. Add the names of these shared components into the RMA Wrapper list (discussed in Section 5.5.3);
5. Wire the interfaces exposed by the individual MACs to the parameterized interfaces exposed by `SwitchUpC` component and `SwitchLowC` component (discussed in Section 5.5.2) using MAC ID as the parameter.

## 5.6 MAC Selection Engine

In this section, we present the design of the MAC Selection Engine.

### 5.6.1 Overview of the Engine

The MAC Selection Engine decides, based on a machine learnable model, which is the best MAC protocol for given application QoS requirements (a REL order), current traffic pattern, and ambient interference levels. As shown in Figure 5.4, the engine includes three major modules: The **Traffic Monitor** keeps track of the application traffic pattern by snooping the send commands called by the application and calculating the mean and variance of Inter-packet Interval (IPI) in a sliding window of 100 seconds. The **Noise Monitor** measures the external interference level in the environments by calculating the mean and variance of the Received Signal Strength (RSS) in a sliding window of 10 seconds. We chose RSS as an indicator estimating the interference level since recent studies showed that 802.15.4 sensors can effectively detect external interferers by polling Received Signal Strength Indicator [49]. The **Classifier** determines the best MAC according to the current application specified REL order and the values emitted from the Traffic and Noise Monitors. The Classifier then issues its decision (MAC ID) as a request to the RMA to switch MACs. The detailed design and implementation of the classifier is described in Section 5.6.2.

We emphasize that only the network coordinator node needs to run the engine and the coordinator's RMA is responsible for propagating the MAC protocol switching across the network (See Section 5.4.4). The other nodes need to have only the RMA module.

### 5.6.2 Classifier

We choose to use a decision tree as the classifier for a number of reasons: there are only a handful of MACs that can practically be available in any container; and each protocol is generally good at some characteristics (e.g. most efficient at low data rates) and is not so good at others. So often the classifier will only have limited, discrete choices to make. When we consider the limited amounts of memory available on sensors, decision trees are compact to represent in a data structure. And from the computational point of view, a decision tree will consume a marginal amounts of resources and will be fast at runtime.

To gather training data for the model, we run experiments that vary the features we were interested in, while recording the operating characteristics and the MAC protocol in use

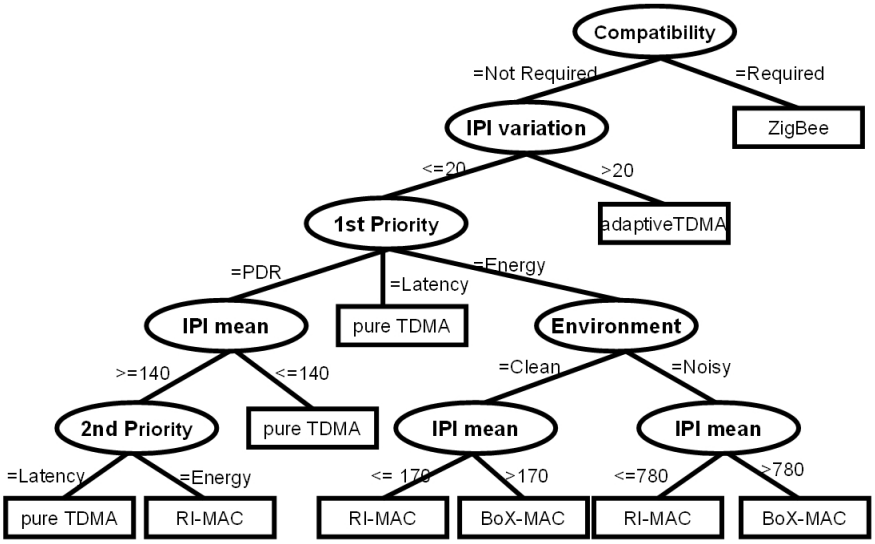



Figure 5.5: Decision tree.

(the class). We determined the features by noting that some protocols are good at high data rates, while others conserve energy and are good at low rates; and some protocols stay reliable at high interference levels, while others do not. With that in mind the characteristics we use for classification are:

1. Application QoS requirements
2. Traffic pattern
3. Ambient interference level

To represent these characteristics, we selected the following features: (1) Application specified REL order; (2) Mean and variance of the Inter-packet Interval (IPI) within a 100 seconds sliding window; and (3) Mean and variance of the RSSI within a 10 seconds sliding window. The protocol being used is recorded and denotes the class.

The **Energy** that each MAC protocol consumes under certain operating conditions is computed by the time duration when the radio is on/off. We define **Latency** as the time interval between a packet being sent by one sensor and the time it is received at another. We use Packet Delivery Rate (PDR) to represent **Reliability**.

We have ran 224 sets of experiments, varying IPI and ambient interference by changing the distance from a pair 802.11 devices with controlled traffic. We measure each combination of operating characteristics multiple times, and collected the operating results of various MACs in terms of the features described above. In total we collected 4624 training examples. Each example is calculated within a sliding window of 10 seconds. We used Weka [120] to learn from the training data and built a decision tree using the C4.5 [90] algorithm. The resulting model (tree) is shown in Figure 5.5.

Referring to Figure 5.5 before decisions are made by the decision tree, if compatibility with other ZigBee devices is required, then at the root node the system instructs to use the ZigBee MAC which was designed for compatibility across different platforms. Otherwise, if the IPI variance is higher than 20 $ms^2$, the classifier selects adaptive TDMA protocol. This is an apt choice that the model learned, since adaptive TDMA allows reconfiguration of TDMA frames to accommodate aperiodic traffic. The next level of the tree takes into account the application's highest REL order (the application's 1st priority) and provides a 3-way split. If the split was on Reliability, the next level of the tree (IPI mean) takes into account the mean IPI. If the Traffic Monitor observes small mean IPI (lower than 140 ms), the system is instructed to use pure TDMA. Otherwise the application 2nd priority is taken into account. If the split was on Energy consumption, the next level of the tree (Environment) takes into account the noise level.

To learn the mode and test its performance measures we used the 10-fold cross validation of Weka. The model obtains a true positive rate of 95.6% and a false positive rate of 7.1%. These performance measures demonstrate that our features can effectively select the best MAC for the specified application's REL order, traffic pattern, and current ambient interference level.

## 5.7 Evaluation

We validate the efficacy of SAML and the efficiency of its operation in numerous ways. We start by measuring the memory footprint of the three prototypes discussed in Section 5.5.4. In these prototypes, RMA hosted between 2 to 5 MAC protocols in its container. This is a typical range and should provide a good representation of code size. We then measure the

| | ROM (bytes) | RAM (bytes) |
|---|---|---|
| BoX-MAC | 25308 | 1114 |
| pure TDMA | 25362 | 1202 |
| RI-MAC | 25132 | 1268 |
| adaptive TDMA | 25418 | 1126 |
| ZigBee MAC | 27168 | 1272 |
| RMA CSMA/TDMA | 28016 | 1254 |
| RMA SI/RI-MAC | 27752 | 1896 |
| RMA 5-MAC | 29990 | 1968 |

Table 5.1: ROM and RAM usage for each RMA prototype or single MAC.

overhead of key RMA operations such as new node joining the network and MAC switching. We intentionally disable the MAC Selection Engine and manually inject the MAC IDs to support the experiments presented in Section 5.7.2. Finally, we enable the MAC Selection Engine and perform a real-world case study in which we demonstrate the effectiveness and benefits of dynamic MAC switching in terms of reliability and energy consumption.

### 5.7.1 Memory Footprint

RMA's design had to balance two conflicting goals. On one hand, we want RMA to host as many MAC protocols as any application may require to optimize its performance along on some future, and possible unknown, dimension. On the other hand, artlessly, more protocols will increase the OS code size. We addressed this conflict by breaking MAC protocols to reusable components, resulting in a code size that is a concave down function of the number of protocols.

In Table 5.1 we compare the ROM and RAM usage as reported by the TinyOS tool-chain for five single MACs as well as three RMA prototypes discussed earlier.

Comparing the RMA CSMA/TDMA prototype containing two MACs (BoX-MAC and pure TDMA), with the one containing only BoX-MAC, we observe that the RMA consumes only 2708 additional bytes of ROM and 140 additional bytes of RAM [11]. This is only 10.7%

[11]For reference the MSP430F1611 MCU used by the TelosB and Tmote Sky motes provides 48 Kilobytes of ROM and 10 Kilobytes of RAM.

ROM and 12.6% RAM increase from a single MAC. Compared to pure TDMA only, RMA prototype consumes 2654 additional bytes of ROM (10.5%) and 52 additional bytes of RAM (4.3%).

RMA SI/RI-MAC prototype with two MACs (BoX-MAC and RI-MAC) adds only 2444 bytes of ROM (9.7%) and 782 bytes of RAM (70.2%) compared to BoX-MAC. RMA adds 2620 bytes of ROM (10.4%) and 628 bytes of RAM (49.5%) compared to RI-MAC. We are not concerned with the high RAM percentage because in terms of percentages from the mote resources the percent increases are similar to the ones reported in the next paragraph (less than 8% in the worse case for all cases we have studied so far).

Comparing the 5-MAC RMA with the ZigBee MAC, which consumes the most memory out of all single MACs, shows that the RMA version consumes 2822 additional bytes of ROM (10.4%) and 696 additional bytes of RAM (54.7%). This increase represents a 5.7 percent increase in the ROM and 6.8% increase in RAM of the mote's memory resources.

In all cases, we clearly see that RMA is highly effective in avoiding memory bloat. This conclusion holds even when supporting a large and diverse set of MACs, which makes RMA practical to deploy on memory constrained sensors.

### 5.7.2 Micro-benchmark Experiments

We evaluate the latency and power consumption when new nodes join the network and switch between MACs. We measure the consumption using a power meter from Monsoon Solutions [80] whose probes are connected to the sensor voltage pins. For this experiment, we use a Samsung Galaxy Note with a gateway board [102] that works as the coordinator. The coordinator is initialized to run BoX-MAC with a wakeup interval of 150 ms and broadcasts the MAC ID every 5 seconds. A mote running the baseline protocol was added to the network at $T_0 = 0$. After 10 seconds, we issue a command through the phone requesting a switch from BoX-MAC to pure TDMA. The pure TDMA frame is configured to include 20 time slots with 10 ms for each slot. The 10th time slot is configured to allow unknown nodes to perform CSMA-based random access. A second mote running the baseline protocol was added to the network after 5 seconds.

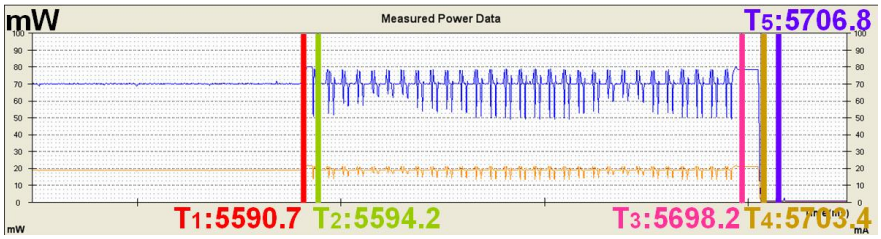



Figure 5.6: Time duration of a node joining a network running BoX-MAC.

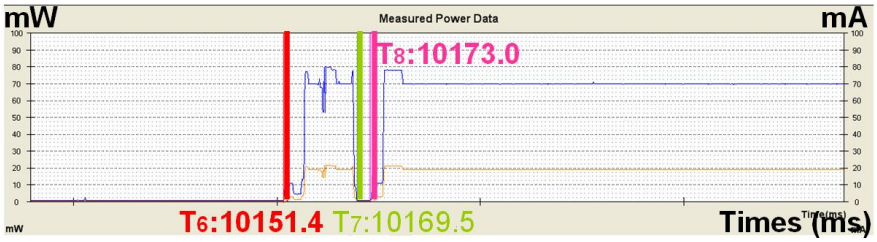



Figure 5.7: Time duration of protocol switching from BoX-MAC to pure TDMA.

Figure 5.6 shows the time duration of the first mote joining this network (running BoX-MAC). After booting, the new mote begins to run the baseline-MAC, snooping the channel continuously. At $T_1 = 5590.7$ ms, the node receives a broadcasted packet with MAC ID of the current protocol sent by the coordinator. At $T_2 = 5594.2$ ms, the node starts sending requests to join the network until it receives an acknowledgement to do so at $T_3 = 5698.2$ ms. This request process takes $T_3 - T_2 = 104$ ms and consumes $7.29mJ$ of energy (70.13 $mW$ of power on average) [12]. At $T_4 = 5703.4$ ms, the new mote turns off the radio, switches the MAC from baseline-MAC to BoX-MAC and then performs low power listening. The switching process $T_5 - T_4$ takes 3.4 ms and consumes $2.86\mu J$ of energy (0.84 $mW$ of power on average).

Figure 5.7 shows the case where the mote receives a command requesting it to switch from BoX-MAC to pure TDMA. At $T_6 = 10151.4$ ms, the mote wakes up and receives the switching command. Then the mote sends back an acknowledgement at $T_7 = 10169.5$ ms, turns off the radio, and starts switching the protocol. The radio is turned back on at $T_8 = 10173.0$, now waiting for a TDMA time synchronization beacon. The switching process $T_8 - T_7$ takes 3.5 ms and consumes $2.87\mu J$ of energy (0.82 $mW$ of power on average).

[12] This duration depends on BoX-MAC wakeup interval.

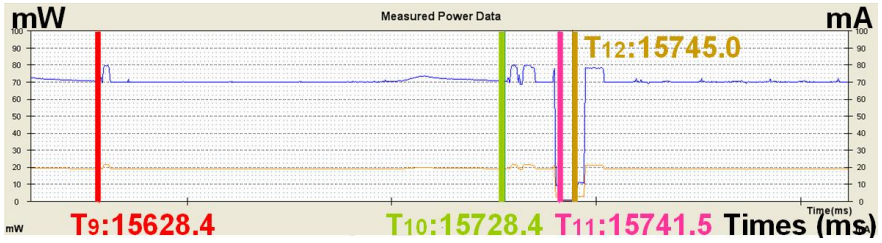



Figure 5.8: Time duration of a node joining the network running pure TDMA.

Figure 5.8 shows the time duration of a second mote, also running baseline-MAC, joining the network running pure TDMA. At $T_9 = 15628.4$ ms, the mote receives a broadcasted packet with MAC ID. With the frame information stored in the beacon, the node realizes that the 10th time slot can be used for new node to randomly access the channel. At $T_{10} = 15728.4$ ms the node performs a CCA check and then transmits a request to join the network. After receiving an acknowledgement, the node turns off the radio at $T_{11} = 15741.5$ ms and switches from the baseline-MAC to pure TDMA. It then waits for a TDMA beacon at $T_{12} = 15745.0$ ms.

For each RMA prototype we randomly generate 100 MAC switching commands through the phone with random intervals ranging from 5 seconds to 10 minutes and obtain a result of 100% of MAC switching success rate. From the power meter traces we observed that the switching process takes about 3.5 ms and consumes about $2.94\mu J$ of energy on average. This short transiting time is achieved by the RMA's fast MAC switching design and this low switching power can be explained by the radio chip being off during the switching. These results demonstrate the efficiency of RMA in terms of controlling nodes as they join the network or switch between different MACs.

### 5.7.3 Case Study

To illustrate the potential benefits of dynamic MAC switching, we perform an empirical case study emulating a wireless health scenario. In our hypothetical scenario, the application periodically (twice per second) samples the person's pulse and oxygen saturation in the blood by using a wireless pulse oximeter. The application starts to collect a 1-hour continuous ECG streaming when an abnormality in these vital signs is detected. In this emulation

we do not perform the sensing because the actual values are irrelevant to the evaluation of SAML. Instead, a corresponding (equivalent to the real application) rate of packet generation is maintained by generating packets following the traffic pattern of 1 packet/500 ms for pulse and oxygen saturation sampling suggested by [28] and adopt a packet rate as high as 1 packet/50 ms with a payload of 15 bytes to accommodate the 500 Hz 12 bits ECG sampling recommended by [34]. In our wireless health scenario, we set (E > R > L) as the REL order during the clinically (periodic pulse and oxygenation sampling) normal period and set (R > L > E) during the clinically abnormal (ECG streaming) period.

For our case study, a Ph.D. student volunteered to wear a TelosB mote on his wrist, a second TelosB mote on his chest [13], and the TelosB gateway in his pocket. The volunteer carefully repeated the same daily schedule over four days to provide similar environments as we collected data for the different protocols.

To compare the performance between single MACs and SAML, the volunteer wore the motes for 12 hours (from 10:00 to 22:00) during four consecutive weekdays and went about his regular daily activities. The activities were repeated at about the same time each day. The motes run our SAML with 5 MACs on the first day, and BoX-MAC, pure TDMA, RI-MAC, one on each of the respective day afterwards. During these days we intentionally emulate three clinically abnormal events to trigger the ECG streaming at 2 hours, 4 hours, and 6 hours after the experiment started each day. Since our volunteer was active and mobile throughout this experiment, we could not measure the power consumption directly. Thus, we instrumented the radio stack, measured the radio ON time $T_{on}$, and use the actual duty cycle to estimate the energy consumption rather than measure it directly by a power meter. We use the equation $U_{on} * I_{on} * T_{on} + U_{off} * I_{off} * T_{off}$ to estimate the power consumption in each cycle (parameters $U_{on}$, $I_{on}$, $U_{off}$, and $I_{off}$ are from CC2420 data sheet).

Figure 5.9 and Figure 5.10 show the raw data of reliability in term of Packet Delivery Rate (PDR) and power consumption during the 10 hours case study. From the figures, we can see that SAML uses BoX-MAC when performing the pulse sampling during the first 2 hours. When an abnormal-vital event is triggered at 12pm, SAML switches to pure TDMA to accommodate the high data rate generated by ECG streaming. SAML switches back to

[13]These two places are where the typical commercial heart rate and ECG sensors would be placed (e.g. Polar Heart Rate Monitor [86], Shimmer ECG mote [109]).

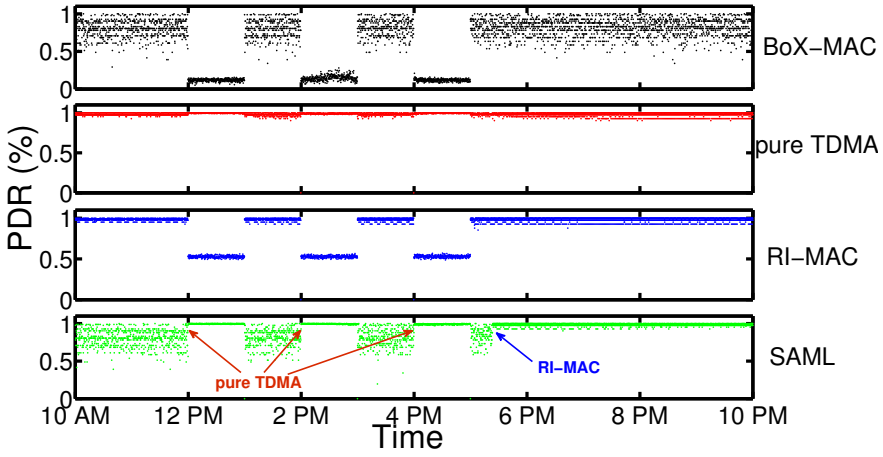



Figure 5.9: PDR of BoX-MAC, pure TDMA, RI-MAC, and SAML during 10 hours.

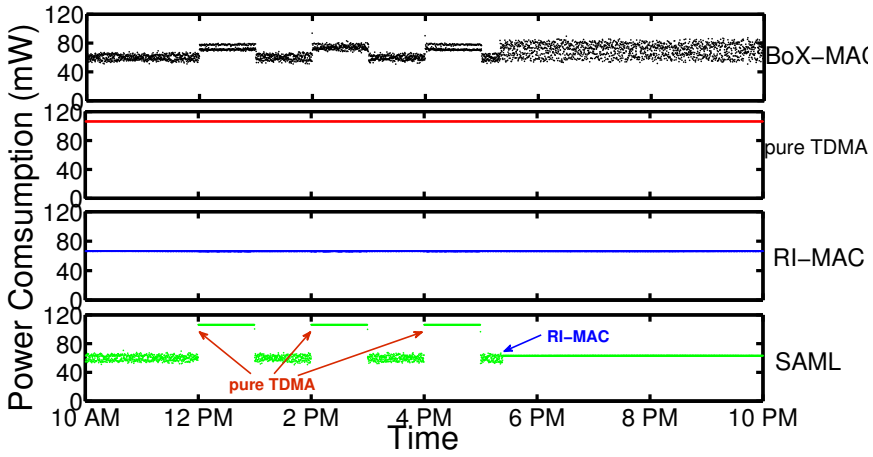



Figure 5.10: Power consumption of BoX-MAC, pure TDMA, RI-MAC, and SAML during 10 hours.

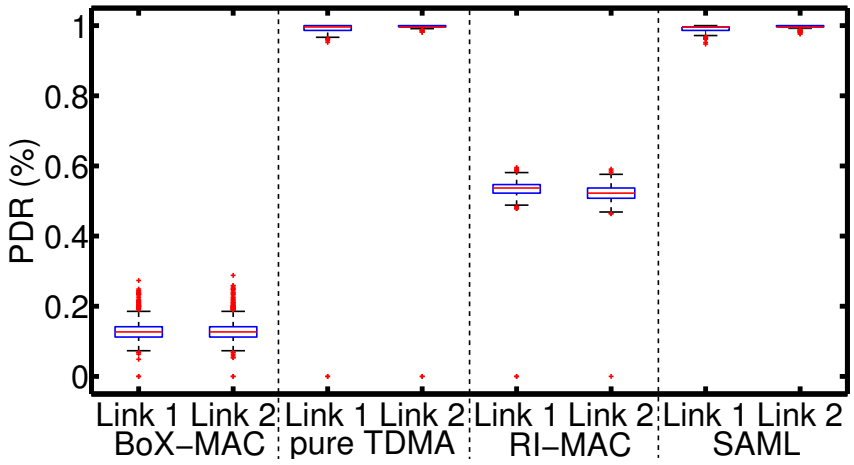



Figure 5.11: Box-plot comparison on PDR of two links during three ECG streaming between BoX-MAC, pure TDMA, RI-MAC, and SAML. Central mark in box indicates median; bottom and top of box represent the 25th percentile ($q_1$) and 75th percentile ($q_2$); crosses indicate outliers ($x > q_2 + 1.5 \cdot (q_2 - q_1)$ or $x < q_1 - 1.5 \cdot (q_2 - q_1)$); whiskers indicate range excluding outliers.

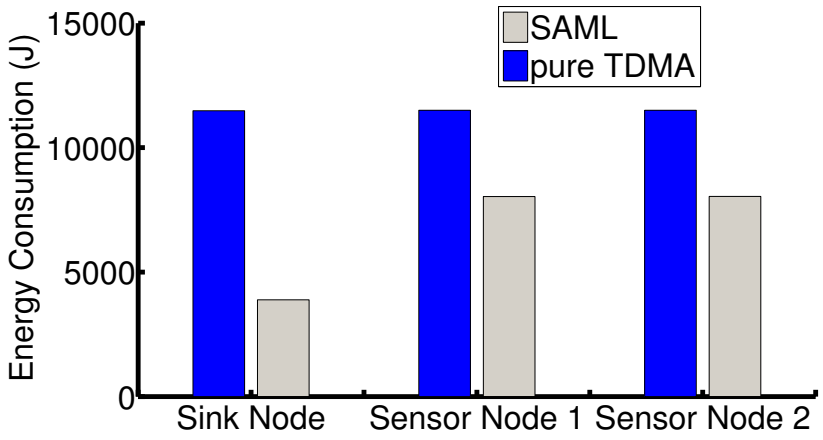



Figure 5.12: Comparison on total energy consumption over three nodes during vital signs sampling between pure TDMA and SAML.

BoX-MAC at 1pm when ECG streaming is stopped. The same pattern repeated at both 2pm and 4pm when two other abnormal-vital event are triggered. Around 5:30pm, SAML switches to RI-MAC when it detects a noisy environment at volunteer's home.

Figure 5.11 shows box-plot of PDR between two links during the three ECG streaming. The plot compares BoX-MAC, pure TDMA, RI-MAC, and SAML in terms of maintaining the link reliability. Only pure TDMA and SAML achieved a PDR higher than 99.6% and met the application reliability requirement (the application specified that as the top priority during ECG streaming). BoX-MAC and RI-MAC failed to provide reliable links and yielded a median PDR of 12.7% and 52.9% respectively. Pure TDMA however, suffers from very high energy consumption due to its fixed time frame and periodic beaconing. From Figure 5.10 we see that during clinically normal period (when data rate is low) pure TDMA (106.4 mW) consumes 77.3% and 60.5% more power than BoX-MAC (60.0 mW) and RI-MAC (66.3mW), respectively. A system with a single MAC, in this case pure TDMA, would not be able to switch during this period to conserve energy. From Figure 5.12, we can see SAML saves 66.2% of energy on the sink node (SAML: 388.6 J and pure TDMA: 1148.2 J) and 30.1% of energy on the sensor node (SAML: 803.6 J and pure TDMA: 1150.0 J) by switching to BoX-MAC on campus (where the interference level is not overly high) and outdoor and switching to RI-MAC at home (high interference due to many WiFi networks). Overall, SAML saves 31.6% of energy (1451.7 J) comparing with pure TDMA but still achieves the QoS reliability requirement of the application throughout the 10-hour case study.

## 5.8 Conclusion

The convergence of mobile phones and wireless sensors exposes the MAC layer to varying applications and dynamic environments where a fixed MAC protocol cannot always deliver satisfactory performance. In contrast to the traditional one-MAC-fit-all approach, we developed SAML, a Self-Adapting MAC Layer that autonomously changes the MAC protocol at runtime. SAML comprises a learning-based MAC Selection Engine and the Reconfigurable MAC Architecture (RMA). We have realized SAML on Android-based mobile phones and TinyOS-based sensors. Experimental results and real-world case studies show that SAML

can select optimal MAC protocols to meet current application demands and switches MAC protocols online in an efficient and reliable fashion.

# Chapter 6

# Experimental Study of Industrial Wireless Sensor and Actuator Networks

## 6.1 Introduction

Process control and automation are crucial for process industries such as oil refineries, chemical plants, and factories. Today's industry mainly relies on wired networks (e.g., HART [46]) to monitor and control their production processes. Cables are used for connecting sensors and forwarding sensor readings to a control room where a controller sends commands to actuators. However, these wired systems have significant drawbacks. It is very costly to deploy and maintain such wired systems, since numerous cables have to be installed and maintained, which often requires laying cables underground. This severely complicates effort to reconfigure systems to accommodate new production process requirements.

WSAN technology is appealing to process control and automation applications because it does not require any wired infrastructure. WSANs can be used to easily and inexpensively retrofit existing industrial facilities without the need to run dedicated cabling for communication and power. IEEE 802.15.4 based WSANs are designed to operate at a low data rate and can be inexpensively manufactured, making them a good fit for industrial automation applications where energy consumption and costs are often important.

However, industrial WSANs pose unique challenges due to their critical demands on reliable and real-time communication. Violation of these reliability and real-time requirements may result in plant shutdowns, safety hazard, or economic/environmental impacts. To meet

the stringent requirements on reliability and predictable real-time performance, industrial WSAN standards such as WirelessHART [122] made a set of unique network design choices.

- The network should support both source routing and reliable graph routing: the source routing provides a single route for each data flow, whereas the graph routing provides multiple redundant routes based on a routing graph.
- The network should also adopt a multi-channel Time Division Multiple Access (TDMA), employing both dedicated and shared time slots, at the MAC layer on top of the IEEE 802.15.4 physical layer. Only one transmission is scheduled in a dedicated slot, whereas multiple transmissions can share a same shared slot. The packet transmission occurs immediately in a dedicated slot, while a CSMA/CA scheme is used for transmissions in a shared slot.

Recently, there has been increasing interest in developing new network algorithms and analysis to support industrial applications. However, there is a lack of academic experimental testbeds for validating and evaluating network research on industrial WSANs. Without sufficient experimental evaluation, industry consequently has shown a marked reluctance to embrace new solutions.

In this chapter, we present an experimental testbed that we have developed for studying and evaluating WSAN protocols. While a multitude of testbeds exist for studying sensor networks, our testbed is unique in that it supports a suite of key networking mechanisms specific to industrial WSANs and a set of end-user tools for managing wireless experiments. We then present a series of empirical studies on WSAN protocol designs including multi-channel TDMA and shared slots at the MAC layer and reliable graph routing. This study leads to a list of insights on the development of resilient industrial WSANs:

- Source routing can cause poor network reliability even with retransmissions due to significant burstiness of transmission failures. In contrast, graph routing is more resilient to interference and its backup routes can be heavily used in noisy environments at the cost of increased latency and energy consumption.

- Channel hopping between transmissions can effectively reduce the burstiness of transmission failures; hopping to far away channels can reduce burstiness further by avoiding using strongly correlated adjacent channels in consecutive transmissions.
- Judicious allocation of multiple transmissions in a shared slot can effectively improve network capacity without significantly impacting reliability.

The rest of the chapter is organized as follows. Section 6.3 introduces the characteristics of industrial WSANs. Section 6.4 presents the hardware and software architecture of our experimental testbed. Section 6.5 describes our empirical study. Section 6.2 reviews related work and Section 6.6 concludes the chapter.

## 6.2 Related Works

In recent years, there has been increasing interest in studying industrial WSANs regarding network algorithms and analysis. Zhang et al. [130] focused on the design of a link scheduling and channel assignment algorithm for a simplified linear network model, while Soldati et al. [113] studied the same problem for tree network models. Rao et al. [92] studied the tradeoff between energy consumption and network performance. Franchino et al. [37] proposed a real-time energy-aware MAC layer protocol. Han et al. [45] presented a graph routing algorithm. Saifullah et al. presented a series of theoretical results on real-time transmission scheduling [98,100], rate selection for wireless control [96], and delay analysis [99,123]. Real-time transmission scheduling algorithms have also been studied in the context of wireless sensor networks (WSNs) [11,29,59,84,97]. All these previous works are based on theoretical analysis and simulation studies, since there is a lack of academic experimental testbeds for validating and evaluating network research on industrial WSANs. Without sufficient experimental evaluation, industry consequently has shown a marked reluctance to embrace new solutions. In this chapter, we present an experimental testbed that we have developed for studying and evaluating WSAN protocols. While a multitude of testbeds exist for studying sensor networks, our testbed is unique in that it supports a suite of key networking mechanisms specific to industrial WSANs and a set of end-user tools for managing wireless experiments.

Industrial WSANs adopt a multi-channel TDMA at the MAC layer, which requires the time synchronization across the entire network. In recent years, there has been increasing interest in developing time synchronization methods for low-power wireless networks. Maroti et al. [76] proposed a time synchronization protocol (FTSP) that floods the time stamps periodically to synchronize the clocks of wireless devices. Rowe et al. designed a time-synchronized link protocol [95] and further proposed a synchronization approach using electromagnetic energy radiating from AC power lines [94]. Buevich et al. [23] developed an external hardware-based clock tuning circuit that can be used to improve synchronization and significantly reduce clock drift over long periods of time without waking up the host microprocessor. In our work, we used the FTSP as an example to serve as the time synchronization protocol since the focus of this chapter is studying industrial WSANs protocols instead of evaluating various time synchronization approaches.

Recently, O'donovan et al. [13] developed a GINSENG system which uses wireless sensor networks to support mission-critical applications in industrial environments and shared their valuable experiences during real-world deployments. In contrast to this work, our work focuses on studying the impact of various protocols, such as channel hopping, graph routing, and channel sharing, on the network performance.

## 6.3 Characteristics of Industrial WSANs

To meet the stringent requirements on reliability and predictable real-time performance, industrial WSAN standards such as WirelessHART [122] made a set of unique network design choices that distinguish industrial WSANs from traditional WSNs designed for best effort services. In particular, we focus on several key network mechanisms supported by WirelessHART [122], a major industrial wireless standard widely used in process industries today.

A typical industrial WSAN consists of a gateway, multiple access points, and a set of field devices (sensors and actuators). The access points and field devices are equipped with half-duplex omnidirectional radio transceivers (compatible with the IEEE 802.15.4 physical layer [57]) and form a wireless mesh network. The access points are connected with the

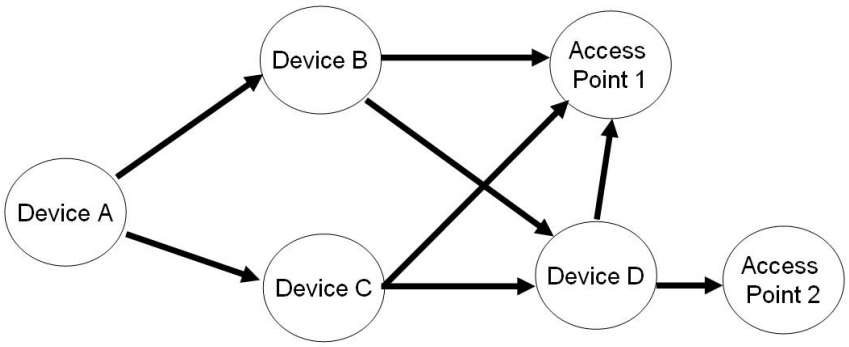



Figure 6.1: An example of graph routing.

gateway through wired links and serve as bridges between the gateway and wireless field devices.

Industrial WSANs adopt a centralized network management architecture that enhances the predictability and visibility of network operations at the cost of scalability. The network manager, a software module running on the gateway, is responsible for managing the wireless network. The network manager collects the network topology information from the devices, determines the routes between the network manager and all devices and the transmission schedule of the network, and then disseminates the routes and schedule to all devices.

Industrial WSANs support both source routing and reliable graph routing. Source routing provides a single route for each data flow, whereas graph routing firstly generates a reliable graph in which each device should have at least two neighbors to which they may send packets and then provide multiple redundant routes based on the graph. Figure 6.1 shows an example. To send a packet to access points, Device A may transmit it to Device B and Device C. From those devices, the packet may take several alternate routes to reach the access points. Compared to source routing, graph routing is designed to enhance network reliability through routing diversity and redundancy.

Industrial WSANs adopt a multi-channel TDMA at the MAC layer. Compared to CSMA/CA, TDMA can provide predictable packet latency which makes it attractive for real-time communication. All devices' clocks are synchronized and time is divided into 10 ms slots classified into dedicated and shared slots. In a dedicated slot, only one sender is allowed to transmit and the transmission occurs immediately after radio is ready in the beginning of the slot. In

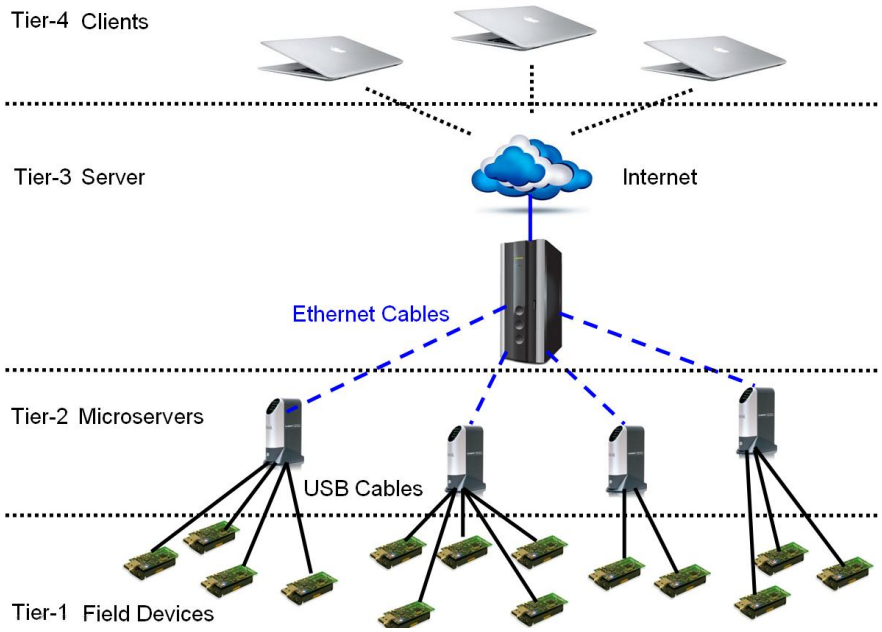



Figure 6.2: Four-tier hardware architecture that consists of field devices, microservers, a server, and clients.

a shared slot, multiple sensors can attempt to transmit and these senders contend for the channel using CSMA/CA. To enhance network capacity and combat interference, industrial WSANs can use up to 16 channels operating in 2.4 GHz ISM band, which are specified in IEEE 802.15.4 standard, and each device switches channel in every slot. Specifically, after transmitting a packet on the channel $x$ in the time slot $k$, a device can hop to the channel corresponding to logical channel $(x+1)\%m$, where m is the number of available channels, for the next transmission in the time slot $k+1$. Channel blacklisting is an optional feature that allows the network operator to restrict the channel hopping of field devices network-wide to selected channels in the wireless band. In each dedicated time slot, the total number of concurrent transmissions cannot exceed the number of available channels.

Figure 6.3: Deployment of the field devices on the fifth floor of Bryan Hall and Jolley Hall at Washington University in St. Louis.

## 6.4 Testbed Architecture

We have implemented the routing and MAC layer protocols in a physical WSAN testbed located in Bryan Hall and Jolley Hall at Washington University in St. Louis. Figure 6.2 illustrates the four-tier hardware architecture that consists of field devices (Tier-1), microservers (Tier-2), a server (Tier-3), and clients (Tier-4).

1. Field devices. The field devices in the testbed are TelosB motes [30], a widely used wireless embedded platform [30] integrating a TI MSP430 microcontroller, a TI CC2420 radio compatible with the IEEE 802.15.4 standard. Figure 6.3 shows the deployment of the field devices on the fifth floor of Bryan Hall and Jolley Hall [125]. A subset of the field devices can be designated as access points in an experiment. The field devices and access points form a multihop wireless mesh network running WSAN protocols.

2. Microservers. A key capability of our testbed is a wired backplane network that can be used for managing wireless experiments and measurements without interfering with wireless communication. The backplane network consists of USB cables and hubs connecting the field devices and microservers which are in turn connected to a server through Ethernet. The microservers are Linksys NSLU2 microservers running Linux. Microservers are responsible for forwarding network management traffic between the field devices and the server.

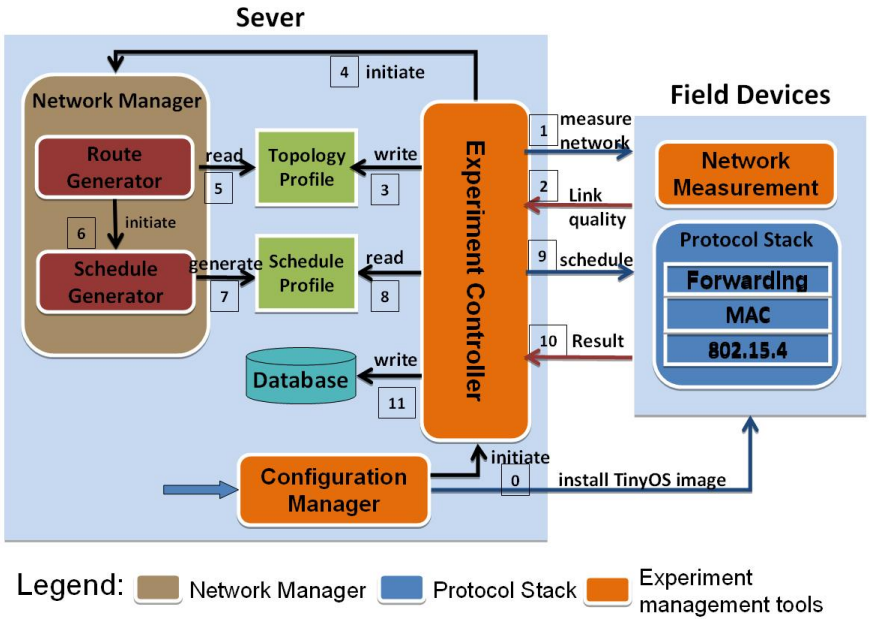



Figure 6.4: Software architecture for the testbed.

3. Server. The server runs network management processes, gathers statistics on network behavior, and provides information to system users. The server also serves as a gateway and runs the network manager of the WSAN.

4. Clients. The clients are regular computers used by users to manage their wireless experiments and collect data from the experiments through the server and the backbone network.

We have developed a software architecture for the testbed, which consists of a network manager running on the server, a protocol stack running on the field devices, and a set of experiment management tools that are distributed across the server and field devices. Figure 6.4 illustrates the software architecture.

1. Network manager. The network manager runs on the server and implements a route generator and a schedule generator. The route generator is responsible for generating source routes or graph routes based on the collected network topology. We use Dijkstra's shortest-path algorithm to generate routes for source routing and follow the algorithm proposed in [45] to generate the reliable graphs. The schedule generator uses rate monotonic scheduling algorithm [74] to generate transmission schedules.

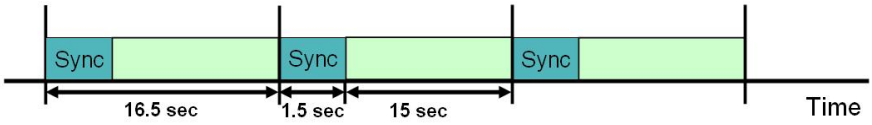



Figure 6.5: Time diagram of RT-MAC.

2. Protocol stack. The protocol stack runs on the field devices and consists of a multi-channel TDMA MAC for dedicated slots, a CSMA/CA MAC for shared slots, and a forwarding layer that forwards packets based on the routes generated by the network manager.

3. Experiment management tools. The tools include a configuration manager, an experiment controller, and a network measurement tool, which are used to initialize and manage the wireless experiments, gather statistics on network behavior, and collect performance results.

### 6.4.1 Protocol Stack

Our protocol stack adopts the CC2420x radio driver as the radio core, which provides an open-source implementation of IEEE 802.15.4 physical layer in TinyOS [4] operating over TI CC2420 radios. CC2420x radio stack takes care of the low-level details of transmitting and receiving packets through the radio hardware.

On top of the radio core, we have developed a multi-channel TDMA MAC protocol-RT-MAC that captures the key features of the MAC protocol used by industrial WSANs. RT-MAC divides the time into 10 ms slots and runs a **Sync** period (1.5 s) in every 1650 slots, as shown in Figure 6.7. Flooding Time Synchronization Protocol (FTSP) [76] is executed during Sync periods to synchronize the clocks of all wireless devices over the entire network. RT-MAC configures the FTSP to flood three time stamps with 500 ms intervals over the network to adjust the local clocks of all devices to a global time source [14]. The period following the Sync period consists of recurring superframes (a series of time slots) and idle intervals. Each slot in superframes begins by allowing a time interval to set the radio to the desired channel.

[14]Our micro-benchmark experiment shows that a FTSP's time stamp packet can finish the traversing of the entire network within 500 ms.

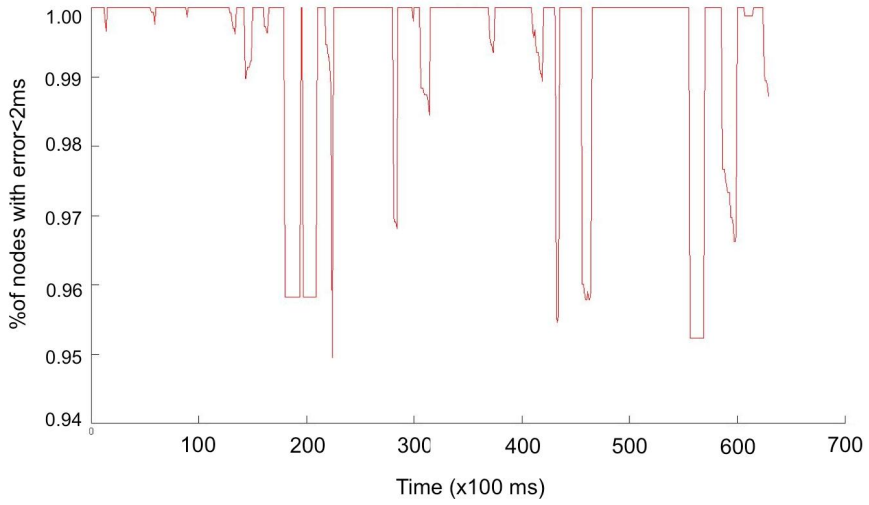



Figure 6.6: Number of field devices with time difference less than 2ms.

We reserve 2 ms in the beginning of each slot to accommodate the channel switching delay and clock synchronization error, since our micro-benchmark experiments show that more than 95% of field devices over the entire network can be synchronized with errors less than 2 ms [15], as shown in Figure 6.8. RT-MAC supports both dedicated and shared slots. In a dedicated slot, only one sender is allowed to transmit and the packet transmission occurs immediately after radio is ready in the beginning of the slot. In a shared slot, more than one sender can attempt to transmit and these senders contend for the channel using CSMA/CA.

### 6.4.2 Experiment Management Tools

We have developed three experiment management tools to manage the wireless experiments, including a network measurement tool, a configuration manager, and an experiment controller. The network measurement tool runs on the field devices, while the configuration manager and experiment controller run on the server.

The network measurement tool on a field device measures the link quality in term of packet reception ratio (PRR). The tool coordinates field devices take turns to broadcast packets

[15]The rest of field devices may disconnect from the network due to larger clock synchronization errors, but they will be reconnected in the next Sync period after they catch the new time stamps generated by FTSP.

over each of the 16 channels and controls the rest of field devices to measure the PRR. Specifically, the transmitter sent a batch of 250 consecutive packets to the broadcast address using a single wireless channel, then proceeded to the next channel in a round robin fashion.

The configuration manager is used for initializing the experiments and installing the TinyOS image to field devices. The experiment controller is responsible for coordinating with the network measurement tool to measure the topology of the network, gathering performance data on network behavior, and saving the data into a database. The performance data collected by the experiment controller includes both end-to-end performance (e.g., packet delivery ratio (PDR) and latency of each data flow) and link performance (e.g., PRR of each link), as well as timestamps of transmission and reception of each packet and radio activities of each field device.

### 6.4.3 Experimental Process

An experiment on the testbed follows the following steps (as illustrated in Figure 6.4). All the network management traffic involved in the process occurs over the wired backplane network.

- Step 0: The configuration manager initiates the experiment controller and installs the TinyOS image to field devices.
- Step 1 and 2: The experiment controller issues a command to the network measurement tool running on all field devices to measure the network topology and then gathers the link quality between field devices.
- Step 3: The experiment controller saves the measured link quality into a topology profile.
- Step 4: The experiment controller initiates the network manager.
- Step 5 and 6: The route generator reads the topology profile, generates the routes, and then initiates the schedule generator.

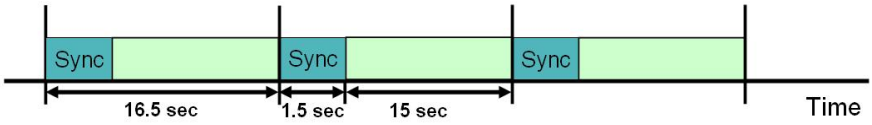



Figure 6.7: Time diagram of RT-MAC.

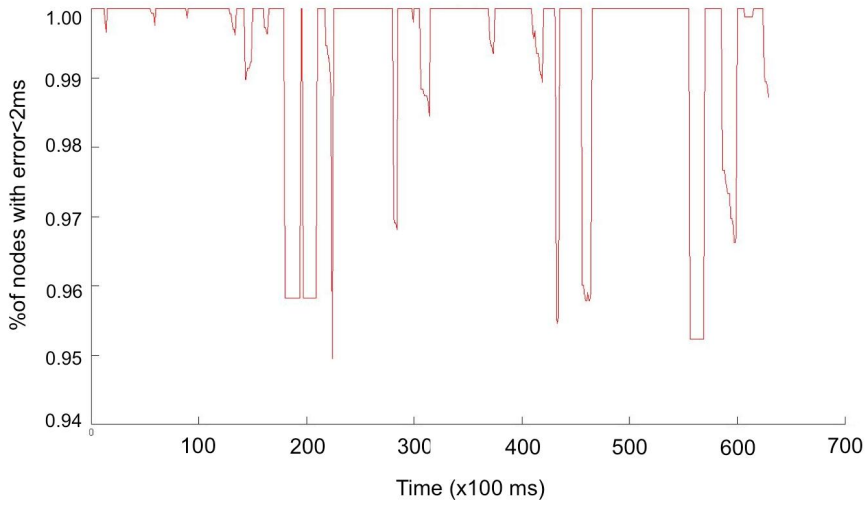



Figure 6.8: Number of field devices with time difference less than 2ms.

- Step 7: The schedule generator generates the transmission schedule based on the routes and then saves it into a schedule profile.
- Step 8 and 9: The experiment controller reads the schedule profile and then forwards the schedule to field devices.
- Step 10 and 11: The experiment controller gathers statistics on network behavior during experiments and saves results into the database.

### 6.4.4 Protocol Stack

Our protocol stack adopts the CC2420x radio driver as the radio core, which provides an open-source implementation of IEEE 802.15.4 physical layer in TinyOS [4] operating over TI CC2420 radios. CC2420x radio stack takes care of the low-level details of transmitting and receiving packets through the radio hardware.

On top of the radio core, we have developed a multi-channel TDMA MAC protocol-RT-MAC that captures the key features of the MAC protocol used by industrial WSANs. RT-MAC divides the time into 10 ms slots and runs a **Sync** period (1.5 s) in every 1650 slots, as shown in Figure 6.7. Flooding Time Synchronization Protocol (FTSP) [76] is executed during Sync periods to synchronize the clocks of all wireless devices over the entire network. RT-MAC configures the FTSP to flood three time stamps with 500 ms intervals over the network to adjust the local clocks of all devices to a global time source [16]. The period following the Sync period consists of recurring superframes (a series of time slots) and idle intervals. Each slot in superframes begins by allowing a time interval to set the radio to the desired channel. We reserve 2 ms in the beginning of each slot to accommodate the channel switching delay and clock synchronization error, since our micro-benchmark experiments show that more than 95% of field devices over the entire network can be synchronized with errors less than 2 ms [17], as shown in Figure 6.8. RT-MAC supports both dedicated and shared slots. In a dedicated slot, only one sender is allowed to transmit and the packet transmission occurs immediately after radio is ready in the beginning of the slot. In a shared slot, more than one sender can attempt to transmit and these senders contend for the channel using CSMA/CA.

### 6.4.5 Experiment Management Tools

We have developed three experiment management tools to manage the wireless experiments, including a network measurement tool, a configuration manager, and an experiment controller. The network measurement tool runs on the field devices, while the configuration manager and experiment controller run on the server.

The network measurement tool on a field device measures the link quality in term of packet reception ratio (PRR). The tool coordinates field devices take turns to broadcast packets over each of the 16 channels and controls the rest of field devices to measure the PRR. Specifically, the transmitter sent a batch of 250 consecutive packets to the broadcast address using a single wireless channel, then proceeded to the next channel in a round robin fashion.

[16] Our micro-benchmark experiment shows that a FTSP's time stamp packet can finish the traversing of the entire network within 500 ms.

[17] The rest of field devices may disconnect from the network due to larger clock synchronization errors, but they will be reconnected in the next Sync period after they catch the new time stamps generated by FTSP.

The configuration manager is used for initializing the experiments and installing the TinyOS image to field devices. The experiment controller is responsible for coordinating with the network measurement tool to measure the topology of the network, gathering performance data on network behavior, and saving the data into a database. The performance data collected by the experiment controller includes both end-to-end performance (e.g., packet delivery ratio (PDR) and latency of each data flow) and link performance (e.g., PRR of each link), as well as timestamps of transmission and reception of each packet and radio activities of each field device.

### 6.4.6 Experimental Process

An experiment on the testbed follows the following steps (as illustrated in Figure 6.4). All the network management traffic involved in the process occurs over the wired backplane network.

- Step 0: The configuration manager initiates the experiment controller and installs the TinyOS image to field devices.
- Step 1 and 2: The experiment controller issues a command to the network measurement tool running on all field devices to measure the network topology and then gathers the link quality between field devices.
- Step 3: The experiment controller saves the measured link quality into a topology profile.
- Step 4: The experiment controller initiates the network manager.
- Step 5 and 6: The route generator reads the topology profile, generates the routes, and then initiates the schedule generator.
- Step 7: The schedule generator generates the transmission schedule based on the routes and then saves it into a schedule profile.
- Step 8 and 9: The experiment controller reads the schedule profile and then forwards the schedule to field devices.

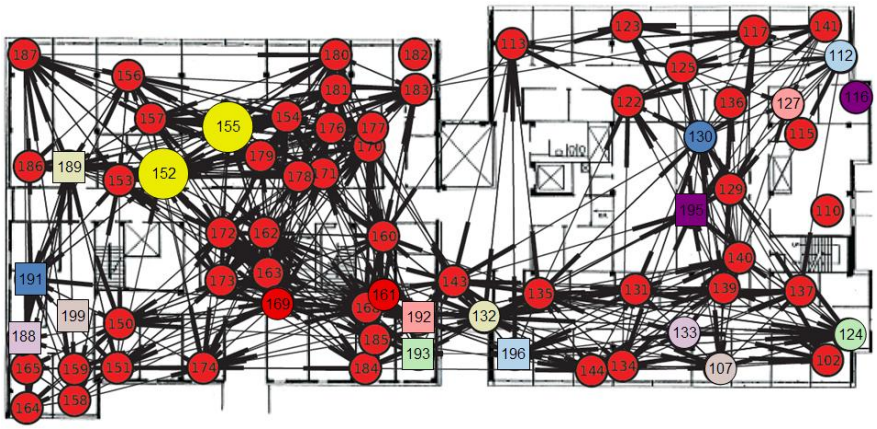



Figure 6.9: Locations of sensors, actuators, and access points.

- Step 10 and 11: The experiment controller gathers statistics on network behavior during experiments and saves results into the database.

## 6.5 Empirical Study

In this study, we first investigate the tradeoff among reliability, latency, and energy consumption under the two alternative routing approaches in clean, noisy, and stress testing environments. We then perform a link study to quantify the burstiness of transmission failures on a same channel. We also study the impact of channel hopping on reducing the burstiness of transmission failures. Finally, we conduct a controlled experiment to explore the potential of channel sharing for improving network capacity.

### 6.5.1 Network Configuration

Figure 6.9 shows one of our network configurations. The bigger yellow spheres denote the access points which communicate with the network manager running on the server through the wired backbone network. The other spheres and squares denote the field devices. The source and destination of a flow is represented as a circle and a square, respectively. The pair of source and destination of a same flow uses the same color. We use 8 flows in our

| Flow ID | Sensor ID | Actuator ID | Period (ms) |
|---|---|---|---|
| 1 | 107 | 199 | 100 |
| 2 | 112 | 196 | 200 |
| 3 | 116 | 195 | 400 |
| 4 | 124 | 193 | 800 |
| 5 | 127 | 192 | 1600 |
| 6 | 130 | 191 | 3200 |
| 7 | 132 | 189 | 6400 |
| 8 | 133 | 188 | 12800 |
| | | | |

Table 6.1: The sensor IDs, actuator IDs, and periods in 8 flows in one of our network configurations.

experiments. The period of each flow (shown in Table 6.1) is picked up from the range of $2^{0\sim7}$ seconds, which are practical periods used in process industries. Following the practice of industrial deployment, the routing algorithms only consider reliable links with PRR higher than 80%. We run our experiments long enough such that each flow can deliver at least 500 packets. We also repeat our experiments with another two network configurations by varying the location of access points, sources, and destinations.

### 6.5.2 Tradeoff between Two Routing Strategies

We conduct a comparative empirical study of the two alternative routing approaches adopted by industrial WSANs, namely source routing and graph routing. Specifically, we investigate the tradeoff among reliability, latency, and energy consumption under the different routing approaches. We run two sets of experiments once with the source routes and once with the graph routing configuration. We repeat the experiments under a clean environment, a noisy environment, and a stress testing environments

1. Clean: we blacklist channels overlapping with our campus Wi-Fi network and run the experiments on other 802.15.4 channels.

2. Noisy: we run the experiments by configuring the network to use channels 16 to 19, which overlap with our campus Wi-Fi network.

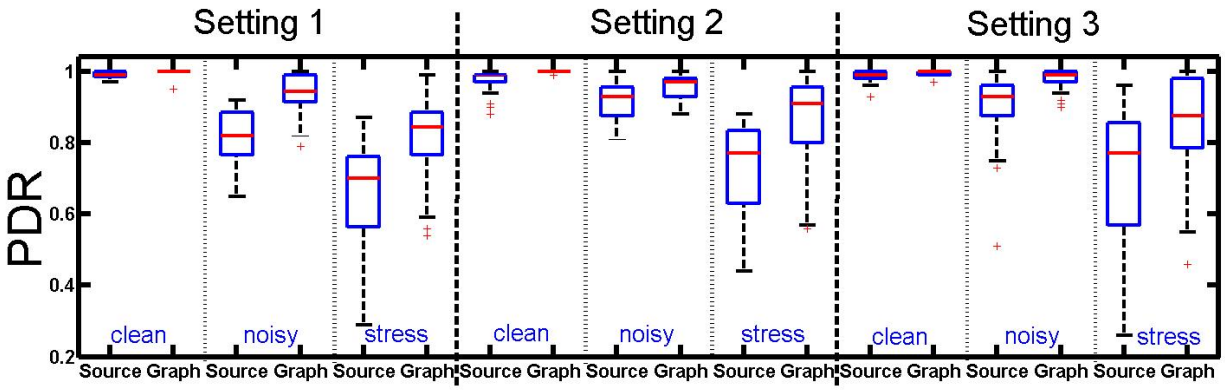



Figure 6.10: Box plot of the PDR of source routing and graph routing in the clean, noisy, and stress testing environments. Central mark in box indicates median; bottom and top of box represent the 25th percentile ($q_1$) and 75th percentile ($q_2$); crosses indicate outliers ($x > q_2 + 1.5 \cdot (q_2 - q_1)$ or $x < q_1 - 1.5 \cdot (q_2 - q_1)$); whiskers indicate range excluding outliers. Vertical lines delineate three different network configurations.

3. Stress testing: we run the experiments with channels 11 to 14 under controlled interference, in the form of a laptop and an access point generating 1 Mbps UDP traffic over the Wi-Fi channel 1, which overlaps with 802.15.4 channels 11 to 14.

We use the PDR to measure reliability. The PDR of a flow is defined as the percentage of packets that are successfully delivered to their destination. Figure 6.10 compares the network reliability under source routing and graph routing in the two environments. As shown in Figure 6.10, under the first network configuration, graph routing achieves an average 1.0% (from 0.99 to 1.0), 15.9% (from 0.82 to 0.95), and 21.4% (from 0.70 to 0.85) increases in median PDR over source routing in the clean, noisy, and stress testing environments, respectively. Graph routing shows similar improvement over source routing under the other two network configurations. More importantly, graph routing shows an improvement in the worst PDR and achieves a smaller variation of PDR over source routing, which can be a significant advantage in industrial applications that demand predictable performance. In the clean environment, graph routing achieves a 4.8% improvement over source routing in term of the worst reliability among the flows. Graph routing shows an even greater improvement (35.5%) in the noisy environment and further improves the min PDR by 63.5% under stress testing. This result shows that graph routing is indeed more resilient to interference due to route diversity. However, as shown in Figure 6.11, the route diversity incurs a cost in term of

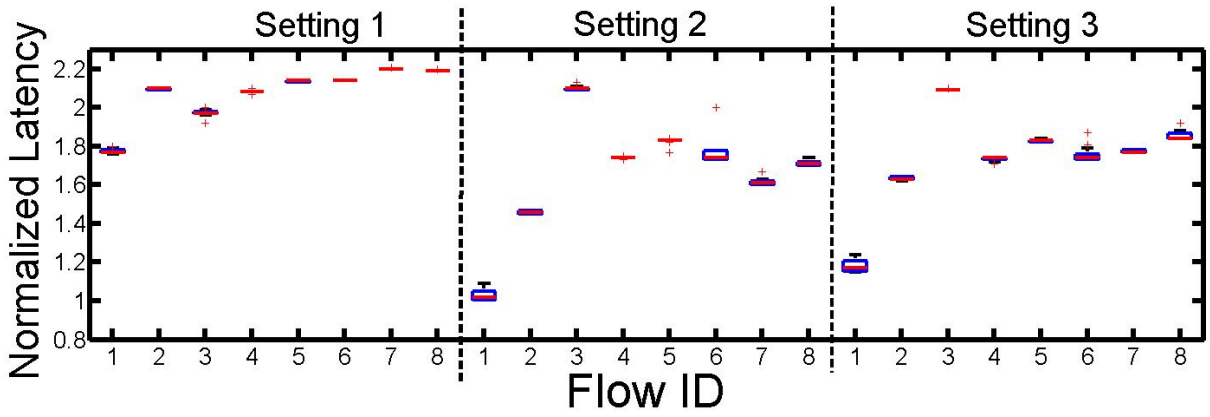



Figure 6.11: Box plot of the normalized latency of source routing and graph routing of each flow under graph routing over that under source routing. Central mark in box indicates median; bottom and top of box represent the 25th percentile ($q_1$) and 75th percentile ($q_2$); crosses indicate outliers ($x > q_2 + 1.5 \cdot (q_2 - q_1)$ or $x < q_1 - 1.5 \cdot (q_2 - q_1)$); whiskers indicate range excluding outliers. Vertical lines delineate three different network configurations.

latency, with graph routing achieving an average of 1.8× (ranging from 1.0 to 2.2) increase in the end-to-end latency.

To estimate the energy consumption, we have measured the time duration and energy consumption of different radio activities by connecting the probes of a power meter from Monsoon Solutions [80] to the voltage pins of a TelosB mote. As Figure 6.12 shows, the mote takes $T_2 - T_1 = 1.0$ ms and consumes $43.6\mu J$ of energy (43.6 $mW$ of power on average) to turn on its radio, while it spends $T_3 - T_2 = 0.6$ ms and consumes $40.6\mu J$ of energy (66.5 $mW$ of power on average) for a Clear Channel Assessment (CCA) check. The packet transmission process $T_4 - T_3$ takes 1.8 ms and consumes $136.8\mu J$ of energy (76.0 $mW$ of power on average). The mote takes $T_5 - T_4 = 0.8$ ms and consumes $56.0\mu J$ of energy (70.0 $mW$ of power on average) when waiting for an ACK, while it spends $T_6 - T_5 = 0.7$ ms and consumes $29.8\mu J$ of energy (42.6 $mW$ of power on average) to turn off its radio. With timestamps of radio activities, we can accurately estimate the energy consumption of all field devices. As Figure 6.13 shows, graph routing consumes an average of 2.3× more energy (ranging from 1.5 to 3.1) over source routing.

***Observation 1****: Graph routing leads to significant improvement over source routing in term of worst-case reliability, at the cost of significant longer latency and higher energy consumption.*

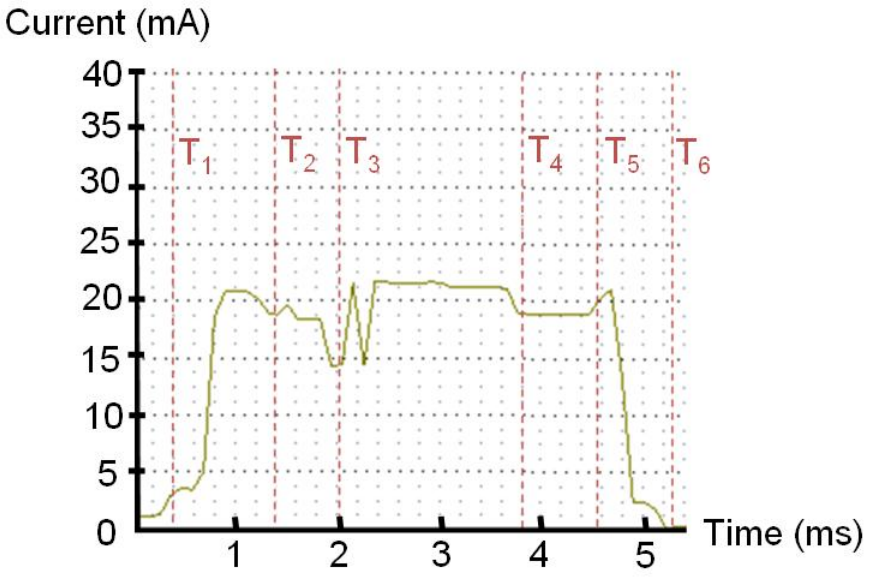



Figure 6.12: The mote turns on the radio ($T_1$–$T_2$), performs a CCA check($T_2$–$T_3$), transmits a packet($T_3$–$T_4$), waits for the ACK ($T_4$–$T_5$), and then turns off the radio ($T_5$–$T_6$).

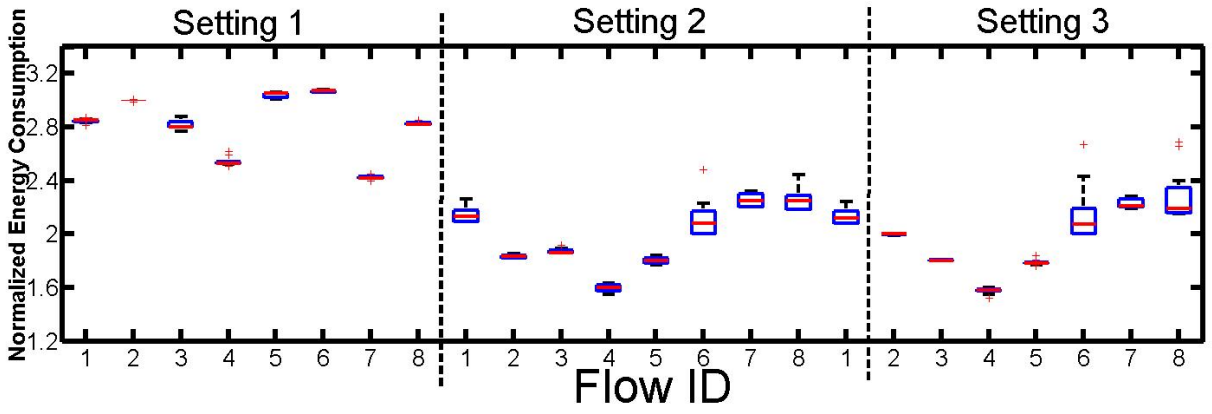



Figure 6.13: Box plot of the normalized energy consumption of source routing and graph routing of each flow under graph routing over that under source routing. Central mark in box indicates median; bottom and top of box represent the 25th percentile ($q_1$) and 75th percentile ($q_2$); crosses indicate outliers ($x > q_2 + 1.5 \cdot (q_2 - q_1)$ or $x < q_1 - 1.5 \cdot (q_2 - q_1)$); whiskers indicate range excluding outliers. Vertical lines delineate three different network configurations.

### 6.5.3 Burstiness of Transmission Failures on a Same Channel

To understand the transmission failures further, we perform a detailed measurement study of the links that suffered from severe packet losses during our previous experiments. In this controlled experiment, the senders of these links transmit a batch of 1000 packets over their interfered channels in the noisy environment. The recipients record the success or failure of each packet and report to the server.

From the experiment, we found that transmissions following a failed one over the same link have a high probability of failures, exhibiting significant burstiness in transmission failures over a same link. It is worth noting that choosing the number of transmissions for each packet must consider the balance between network reliability and efficiency, reserving a large number of retransmissions in the schedule can lead to significant waste in network capacity when the retransmissions are not needed. As a result, industrial WSANs usually can only afford two or three transmissions per packet over a same link. Figure 6.14 illustrates this problem with the Cumulative Distribution Function (CDF) of consecutive packet drops for five least reliable links. Each data point $(x, y)$ on the CDF curve represents $y\%$ of transmission failures has less than $x$ consecutive packet drops. On the most bursty link (link 3), 73.1% of transmission failures are part of at least two consecutive failures and 59.7% of transmission failures are part of at least three consecutive failures. On the remaining four links, 44.8% of transmission failures are part of at least two consecutive failures. Moreover, bursts of 56 consecutive packet drops were observed in our experiment. This result shows that consecutive retransmissions over a same link are insufficient in alleviating transmission failures due to the significant burstiness of transmission failures. A possible cause of the burstiness is that the other wireless devices such as Wi-Fi access points, Bluetooth peripherals, and cordless phones that share the unlicensed 2.4 GHz ISM band cannot sense the ongoing 802.15.4 transmissions which can be easily corrupted due to their low transmission power and long air time. With much shorter packet air time and inter-packet intervals, a backlogged interferer can potentially corrupt the vast majority of consecutive 802.15.4 packets.

***Observation 2****: Consecutive retransmissions over a same link are insufficient in alleviating transmission failures due to the significant burstiness of packet losses.*

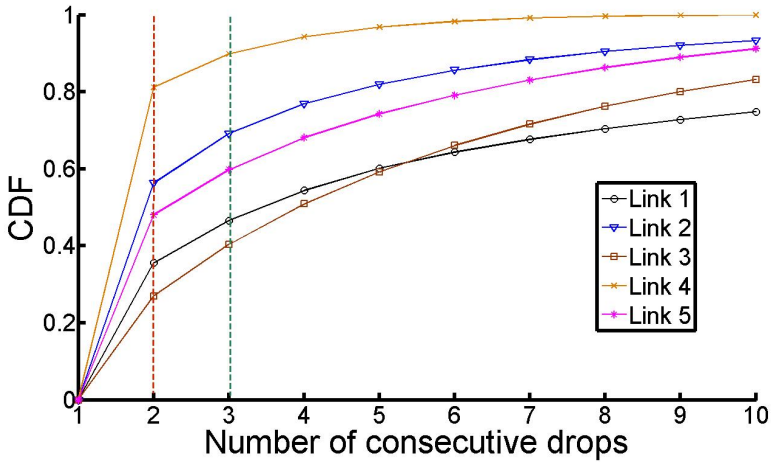



Figure 6.14: CDF of number of consecutive drops (single channel).

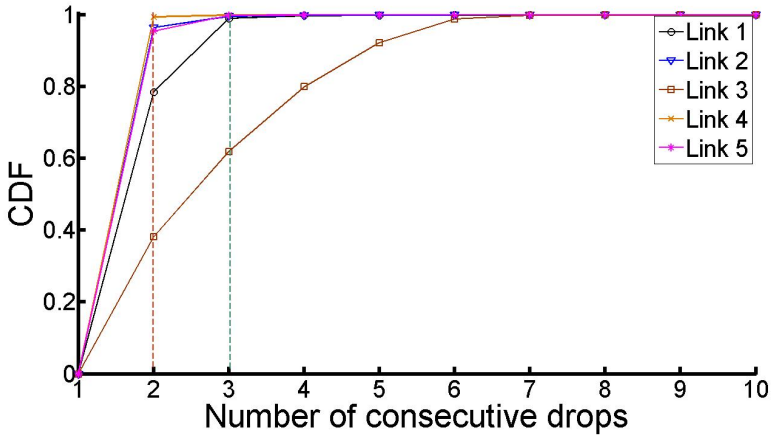



Figure 6.15: CDF of number of consecutive drops (sequential channel hopping).

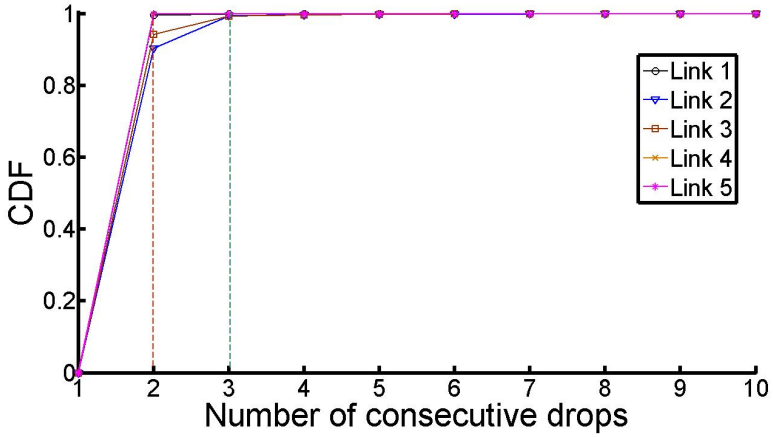



Figure 6.16: CDF of number of consecutive drops (hopping to four-channel-away channels).

### 6.5.4 Impact of Channel Hopping on Burstiness of Transmission Failures

A key mechanism adopted by industrial WSANs is spectrum diversity through channel hopping. Given the burstiness of transmission failures on a same channel observed in the last set of experiments, we now investigate the impact of channel hopping on the burstiness of transmission failures. Specifically, we explore whether the sequential channel hopping approach suggested by the WirelessHART standard [122] is able to eliminate the burstiness of transmission failures. In this set of experiments, instead of staying on a same channel, a sender hops to the next channel if it observes a transmission failure on the current one. As Figure 6.15 shows, the burstiness of transmission failures on link 2, 4, and 5 is reduced significantly. Only fewer than 4.6% of transmission failures are part of at least two consecutive failures. However, the burstiness still exists on the remaining two links. On the most bursty link (link 3), 61.9% of transmission failures are part of at least two consecutive failures and 48.1% of transmission failures are part of at least three consecutive failures. This result suggests channel hopping is an effective mechanism to mitigate the burstiness of transmission failures, but it cannot eliminate the burstiness from all links. A practical solution is to identify the remaining bursty links and selectively allocate more retransmissions to those links.

***Observation 3**: Sequential channel hopping can effectively reduce the burstiness of transmission failures but cannot eliminate it from all links.*

The reason that sequential channel hopping cannot eliminate the burstiness is that there can be strong correlations among adjacent channels. To quantify the correlation, we have studied the effect of channel distance (the absolute difference between channel indices) on the conditional probability of channel failure (the probability that channel $x$ loses a packet when channel $y$ also loses a packet) based on the data traces shown in Section 6.5.2. In Figure 6.17, we plot the probability of simultaneous link failures as a function of channel distance. From Figure 6.17, we observed that this probability can be as high as 34.8% between neighboring channels and 20.5% between every other channel, but drops off as channel distance increases to more than 3. Therefore, we hypothesize that a device can reduce the burstiness in its transmission failures by hopping to a channel farther away from the current one. We test this hypothesis by studying whether hopping to four-channel-away channels is able to eliminate

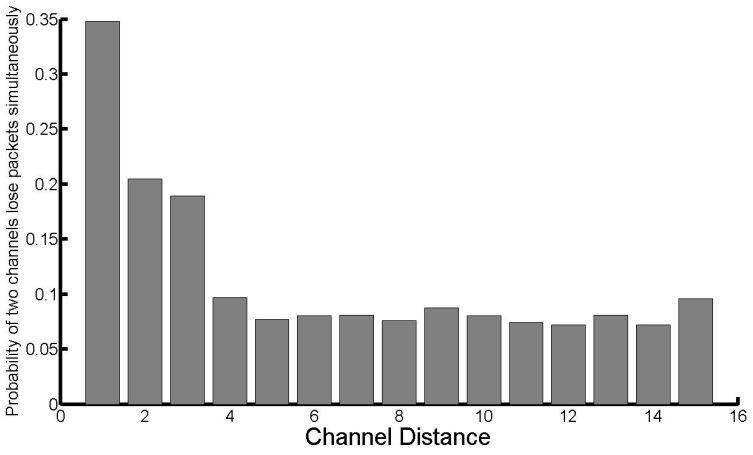



Figure 6.17: Probability of two channels losing packets simultaneously.

the burstiness through a new set of experiments. As Figure 6.16 shows, the burstiness of transmission failures is effectively eleiminated on three links. On the remaining two links, only 5.7% and 9.5% of transmission failures, respectively, are part of two consecutive failures. Three consecutive failures was never observed. The results confirm that hopping to far away channels is more effective than sequential channel hopping in reducing the burstiness of consecutive transmission failures. This suggests an enhancement to the existing sequential hopping policy often adopted by industrial WSANs.

***Observation 4****: Hopping to farther away channels is more effective than sequential channel hopping in reducing the burstiness of transmission failures.*

We note that the appropriate channel hopping distance and proper number of transmissions per packet may vary in different wireless environments. Therefore, network developers may follow our metric to pursue these two variables in their own wireless environments.

### 6.5.5 Time Slot Sharing

There exists an inherent tradeoff between network reliability and capacity of a WSAN when only dedicated slots are used for retransmissions. Scheduling more retransmissions will improve reliability, but may waste slots when the retransmissions are not needed at run time. To mitigate the impact of retransmissions on network capacity, industrial WSANs introduce

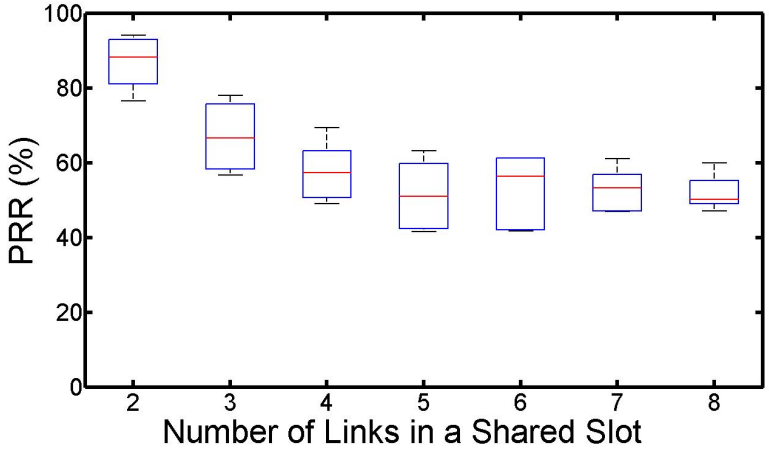



Figure 6.18: PRR of links scheduled in a shared slot.

a mechanism called "shared slots" to allow multiple links to share a slot for retransmissions. The senders of a shared slot must contend for the channel using CSMA/CA.

To study the reliability of transmissions in shared slots, we perform a controlled experiment in which multiple senders contend for the wireless medium. We schedule the senders of two closest links to transmit 1000 packets over the same channel in a sequence of 1000 slots [18], then add more and more nearby links [19] to use the shared slots. The recipients record the success or failure of each packet and report to the server. We repeat this experiment 10 times with three different sets of links located at different places on our testbed.

Figure 6.18 shows the box-plot of the PRR of all links with different number of links in a shared slot. Two senders sharing a same slot achieves a median PRR of 88.2%, while the median PRRs reduce to 66.7%, 52.7%, and 46.2% when more senders are added into the slot.

For comparison, we also calculate the theoretical time to deliver multiple packets. Specifically, the time is $T^{on} + T_1^{cca} + T_1^{tx} + T_1^{ack} + \Sigma_{i=2}^{n}(T_i^{backoff} + T_i^{tx} + T_i^{ack}) + T^{off} + T^{guard}$, where $T^{on}$ is the time that the radio is turned on and the channel is configured (1.0 ms); $T^{cca}$ is

[18] We intentionally added time gaps between transmissions to avoid the impact of burstiness of transmission failures mentioned in Section 6.5.3

[19] We choose nearby links in our study, since we are interested in the slot duration sharing and capture effect between adjacent links instead of the traditional spatial reuse approaches (allocating far away links to reuse a same channel), which have been well studied in the literature.

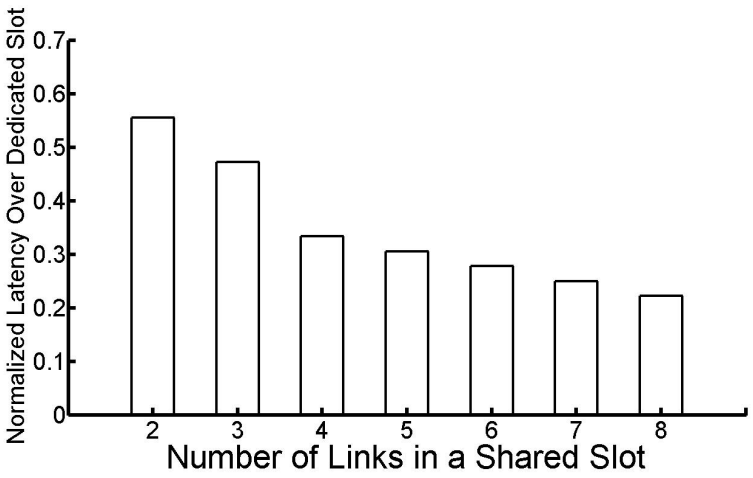



Figure 6.19: Normalized latency using shared slots relative to that using dedicated slots.

the time for a CCA check (0.6 ms); $T^{tx}$ is packet air time (0.35 ms $\sim$ 4.3 ms); $T^{ack}$ is the waiting time for ACK (0.8 ms); $T^{backoff}$ is the random backoff time before devices attempt to access the wireless medium; $T^{off}$ is the time that the radio is turned off (0.7 ms); and $T^{guard}$ is the guard time for time synchronization (2.0 ms). In our experiments, the air time of packets is around 1.8 ms. Therefore, two packets can be delivered in a 10 ms slot in most cases when the random generated $T^{backoff}$ is smaller than 0.5 ms ($T^{backoff} \in [0, 0.64]ms$). To take advantage of slot duration, network developers may follow the above calculation to determine how many links can be safely assigned to a shared slot.

However, as shown in Figure 6.18, the minimum PRR is 41.6% even when we added more senders into the share time slots. According to the timestamps collected during the measurement, data packets were transmitted simultaneously when transmitters cannot detect each others' signals and hence do not defer their transmissions even when there exist ongoing transmissions. However, multiple simultaneous transmissions can be delivered successfully, as a packet can be successfully decoded in spite of a collision if the signal-to-interference-plus-noise-ratio is above a certain threshold [107, 114]. Figure 6.19 shows the normalized latency using shared slots relative to that using dedicated slots to sequentially schedule all the transmissions. The normalized latency of delivered packets reduces from 55.6% to 22.2% when we added more senders into the share time slots, while increasing the number of links in a shared slot consequently has a 5.5% increase in normalized energy consumption (from 1.09 to 1.16) since the transmitters have to leave the radio on longer to contend for the

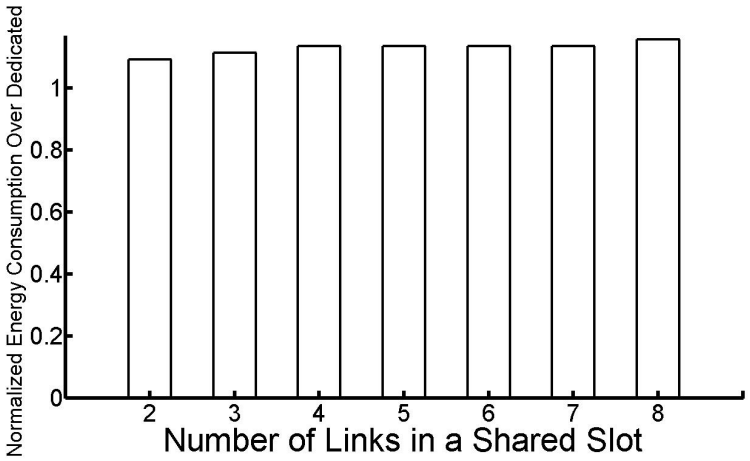



Figure 6.20: Normalized energy consumption using shared slots relative to that using dedicated slots

channel. This observation suggests multiple transmissions may be scheduled in a shared slot to exploit the capture effect [121], whereby a packet with the stronger signal strength can be received in spite of a collision, that allows concurrent transmissions.

***Observation 6****: Multiple transmissions can be scheduled in a shared slot to take advantage of slot duration and the capture effect allowing concurrent transmissions.*

## 6.6 Conclusion

Wireless technology offers a promising platform for process control and automation applications since it does not require any wired infrastructure. WSANs can be used to easily and inexpensively retrofit existing industrial facilities without the need to run dedicated cabling for communication and power. However, industrial WSANs pose unique challenges due to their critical demands on reliable and real-time communication. To enable the validation and evaluation of network research on industrial WSANs, we developed an experimental testbed by realizing the key networking mechanisms specific to industrial WSANs and a set of end-user tools for users to manage wireless experiments. We then performed a series of empirical studies on WSAN protocol designs such as multi-channel TDMA and shared slots at the MAC layer and reliable graph routing, looking both at end-to-end and link

performance. The observations made in our study highlight the significant challenges that face industrial applications for achieving reliable wireless communication. Nevertheless, our observations also suggest that these challenges may be tamed through the judicious use of channel diversity and channel sharing. Specifically, we have distilled our observations into a list of insights which could greatly impact the development of resilient industrial WSANs:

- Source routing can cause poor network reliability even with retransmissions due to significant burstiness of transmission failures. In contrast, graph routing is more resilient to interference and its backup routes can be heavily used in noisy environments at the cost of increased latency and energy consumption.
- Channel hopping between transmissions can effectively reduce the burstiness of transmission failures; hopping to far away channels can reduce burstiness further by avoiding using strongly correlated adjacent channels in consecutive transmissions.
- Judicious allocation of multiple transmissions in a shared slot can effectively improve network capacity without significantly impacting reliability.

# Chapter 7

# Conclusion

In recent years, there has been growing interest in deploying various wireless sensing applications in real-world environments. For example, smart energy systems provide fine-grained metering and control of home appliances in residential settings. Similarly, assisted living applications such as vital sign monitoring and fall detection leverage wireless sensors to provide continuous health monitoring in homes. WSNs offer a promising platform for these applications because they do not require a fixed wired infrastructure. However, real deployments of WSNs pose significant challenges due to their low-power radios and uncontrolled ambient environments. Our empirical study in over 15 real-world apartments showed that low-power WSNs based on the IEEE 802.15.4 standard are highly susceptible to external interference beyond user control, such as Wi-Fi access points, Bluetooth peripherals, cordless phones, and numerous other devices prevalent in residential environments that share the unlicensed 2.4 GHz ISM band with IEEE 802.15.4 radios.

To address these real-world challenges, we developed two practical wireless network protocols including the ARCH and the AEDP. ARCH enhances network reliability through channel diversity: devices opportunistically change their radio's frequency in order to avoid adverse channel conditions such as interference and environmental noise; AEDP reduces false wakeups in a noisy wireless environment by dynamically adjusting the wakeup threshold of low-power radios.

Another major trend in WSNs is the convergence with smart phones. To deal with the dynamic wireless conditions and varying application requirements of mobile users, we developed the SAML to support adaptive communication between smart phones and wireless sensors.

SAML dynamically selects and switches MAC protocols to changes in ambient conditions and application requirements.

Compared with the residential and personal wireless systems, industrial applications such as process automation pose unique challenges due to their critical demands on reliability and real-time performance. We developed an experimental testbed by realizing key network mechanisms of industrial WSANs including multi-channel TDMA with shared slots at the MAC layer and reliable graph routing. We then performed an in-depth empirical study on the reliability, latency, and energy consumption of variant solutions under clean, noisy, and stress testing conditions, providing key insights for meeting the reliable and real-time constraints of industrial applications. Our study shows that graph routing is more resilient to interference and its backup routes may be heavily used in noisy environments, which demonstrate the necessity of path diversity for reliable WSANs. Our study also suggests that combining channel diversity with retransmission may effectively reduce the burstiness of transmission failures and judicious allocation of multiple transmissions in a shared slot can effectively improve network capacity without significantly impacting reliability.

# References


[1] Wi-Spy, http://www.metageek.net/.

[2] `http://www.metageek.net/recordings`.

[3] 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver, Texas Instruments.

[4] `http://www.tinyos.net/`.

[5] Specification of the Bluetooth System, Version 4.0.

[6] Technical Overview of Time Synchronized Mesh Protocol, White Paper, http://www.dustnetworks.com.

[7] `http://www.contiki-os.org/`.

[8] `http://www.zti-telecom.com/EN/LanTrafficV2.html`.

[9] `http://docs.tinyos.net/index.php/CC2420_Hardware_and_Software_Acks`.

[10] IEEE 802.15.4e WPAN task group.

[11] T. F. Abdelzaher, S. Prabh, , and R. Kiran. On real-time capacity limits of multihop wireless sensor networks. In *RTSS*, 2004.

[12] Gahng-Seop Ahn, Se Gi Hong, Emiliano Miluzzo, Andrew T. Campbell, and Francesca Cuomo. Funneling-mac: a localized, sink-oriented mac for boosting fidelity in sensor networks. In *SenSys*, 2006.

[13] Tony O'donovan andJames Brown, Felix Busching, Alberto Cardoso, Jose Cecilio, Jose Do, Pedro Furtado, Paulo Gil, Anja Jugel, Wolf-Bastian Pottner, Utz Roedig, Jorge Sa Silva, Ricardo Silva, Cormac J. Sreenan, Vasos Vassiliou, Thiemo Voigt Lars Wolf, and Zinon Zinonos. The ginseng system for wireless monitoring and control: Design and deployment experiences. *ACM Transactions on Sensor Networks*, 10(1), November, 2013.

[14] Paramvir Bahl, Ranveer Chandra, Thomas Moscibroda, Rohan Murty, and Matt Welsh. White space networking with Wi-Fi like connectivity. In *Sigcomm*, 2009.

[15] Victor Bahl, Ranveer Chandra, and John Dunagan. Ssch: Slotted seeded channel hopping for capacity improvement in ad hoc networks. In *MobiCom*, 2004.

[16] S. Bandopadhya and E. Coyle. An energy efficient hierarchical clustering algorithm for wireless sensor networks. In *INFOCOM*, 2003.

[17] Taylor Bell. Common sources of interference. https://support.metageek.net/hc/en-us/articles/200628894-Common-Sources-of-Interference.

[18] M. Bertocco, G. Gamba, and A. Sona. Experimental optimization of CCA thresholds in wireless sensor networks. In *EMC*, 2007.

[19] Carlo Alberto Boano, Thiemo Voigt, Nicolas Tsiftes, and Luca Mottola. Making sensornet mac protocols robust against interference. In *EWSN*, 2010.

[20] Carlo Alberto Boano, Marco Antonio Zuniga, Kay Roemer, and Thiemo Voigt. Jag: Reliable and predictable wireless agreement under external radio interference. In *RTSS*, 2012.

[21] Micah Z. Brodsky and Robert T. Morris. In defense of wireless carrier sense. In *SIGCOMM*, 2009.

[22] Michael Buettner, Gary V. Yee, Eric Anderson, and Richard Han. X-mac: a short preamble mac protocol for duty-cycled wireless sensor networks. In *SenSys*, 2006.

[23] Maxim Buevich, Niranjini Rajagopal, and Anthony Rowe. Hardware assisted clock synchronization for real-time sensor networks. In *RTSS*, 2013.

[24] Geoffrey Werner Challen, Jason Waterman, and Jason Waterman. Idea: Integrated distributed energy awareness for wireless sensor networks. In *MobiSys*, 2010.

[25] R. Chandra, R. Mahajan, V. Padmanabhan, and M. Zhang. Crawdad data set microsoft/osdi2006. 2007.

[26] Dawei Chen, Sixing Yin, Qian Zhang, Mingyan Liu, and Shufang Li. Mining spectrum usage data: a large-scale spectrum measurement study. In *Mobicom*, 2009.

[27] K. K. Chintalapudi and L. Venkatraman. On the design of mac protocols for lowlatency hard real-time discrete control applications over 802.15.4 hardware. In *IPSN*, 2008.

[28] O. Chipara, C. Lu, T.C. Bailey, and G.-C. Roman. Reliable clinical monitoring using wireless sensor networks: Experience in a step-down hospital unit. In *SenSys*, 2010.

[29] Octav Chipara, Chengjie Wu, Chenyang Lu, and William Griswold. Wbust: A real-time energy-aware mac layer protocol for wireless embedded systems. In *ECRTS*, 2011.

[30] Crossbow Technology. TelosB mote platform. `http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/TelosB_Datasheet.pdf`.

[31] Adam Dunkels. The contikimac radio duty cycling protocol. Technical Report 5128, Swedish Institute of Computer Science, 2011.

[32] 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 *SenSys*, 2010.

[33] Prabal Dutta, Jay Taneja, Jaein Jeong, Xiaofan Jiang, and David Culler. A building block approach to sensornet systems. In *SenSys*, 2008.

[34] Charles E. Korrmann et al. Recommendations for Standardization of Leads and of Specifications for Instruments in Electrocardiography and Vectorcardiography. XXXV:583–602, March 1967.

[35] Andras Farago, Andrew D. Myers, Violet R. Syrotiuk, and Gergely V. Zaruba. Meta-mac protocols: Automatic combination of mac protocols to optimize performance for unknown conditions. In *IEEE Journal on Selected Areas in Comunication*, 2000.

[36] Rodrigo Fonseca, Omprakash Gnawali, Kyle Jamieson, and Philip Levis. Four-bit wireless link estimation. In *HotNets VI*, 2007.

[37] G. Franchino and G. Buttazzo. Wbust: A real-time energy-aware mac layer protocol for wireless embedded systems. In *ETFA*, 2012.

[38] Yong Fu, Mo Sha, Gregory Hackmann, and Chenyang Lu. Practical control of transmission power for wireless sensor networks. In *ICNP*, 2013.

[39] S. Ganu, S. Zhao, L. Raju, B. Anepu, I. Seskar, and D. Raychaudhuri. Architecture and prototyping of an 802.11-based self-organizing hierarchical ad-hoc wireless network (sohan). In *PIMRC*, 2004.

[40] Matthias Gauger, Pedro Jose Marron, and Christoph Niedermeier. Tinymodules: Code module exchange in tinyos. In *INSS*, 2009.

[41] D. Gay, P. Levis, R. V. Behren, M. Welsh, E. Brewer, and D. Culler. The nesC Language: A Holistic Approach to Network Embedded Systems. In *ACM PLDI*, 2003.

[42] Omprakash Gnawali, Rodrigo Fonseca, Kyle Jamieson, David Moss, and Philip Levis. Collection tree protocol. In *SenSys*, 2009.

[43] Ramakrishna Gummadi, David Wetherall, Ben Greenstein, and Srinivasan Seshan. Understanding and mitigating the impact of RF interference on 802.11 networks. In *Sigcomm*, 2007.

[44] Gregory Hackmann, Octav Chipara, and Chenyang Lu. Robust topology control for indoor wireless sensor networks. In *SenSys*, 2008.

[45] Song Han, Xiuming Zhu, A. Mok, D. Chen, and M. Nixon. Reliable and real-time communication in industrial wireless mesh networks. In *RTAS*, 2011.

[46] http://www.hartcomm.org/.

[47] Jan-Hinrich Hauer, Vlado Handziski, and Adam Wolisz. Experimental study of the impact of WLAN interference on IEEE 802.15.4 body area networks. In *EWSN*, 2009.

[48] W.R. Heinzelman, A. Chandrakasan, , and H.Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *HICSS*, 2000.

[49] Frederik Hermans, Olof Rensfelt, Thiemo Voigt, Edith Ngai, Lars-Ake Norden, and Per Gunningberg. Sonic: Classifying interference in 802.15.4 sensor networks. In *IPSN*, 2013.

[50] Timothy W. Hnat, Vijay Srinivasan, Jiakang Lu, Tamim I. Sookoor, Raymond Dawson, John Stankovic, and Kamin Whitehouse. The hitchhiker's guide to successful residential sensing deployments. In *SenSys*, 2011.

[51] Ivan Howitt and Jose A. Gutierrez. IEEE 802.15.4 low rate - wireless personal area network coexistence issues. In *WCNC*, 2003.

[52] Jun Huang, Guoliang Xing, Gang Zhou, and Ruogu Zhou. Beyond co-existence: Exploiting wifi white space for zigbee performance assurance. 2010.

[53] Kuo-Chun Huang, Xiangpeng Jing, and Dipankar Raychaudhuri. Mac protocol adaptation in cognitive radio networks: An experimental study. In *ICCCN*, 2009.

[54] Jonathan W. Hui and David Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *SenSys*, 2004.

[55] Philipp Hurni and Torsten Braun. Maxmac: a maximally traffic-adaptive mac protocol for wireless sensor networks. In *EWSN*, 2010.

[56] Jamil Ibriq and lmad Mahgoub. Cluster-based routing in wireless sensor networks: Issues and challenges. In *SPECTS*, 2004.

[57] IEEE. *Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)*, 2006.

[58] ISA 100. http://www.isa.org/isa100.

[59] Praveen Jayachandran and Matthew Andrews. Minimizing end-to-end delay in wireless networks using a coordinated edf schedule. In *INFOCOM*, 2010.

[60] Xiangpeng Jing, Shanmuga S Anandaraman, Mesut Ali Ergin, Ivan Seskar, and Dipankar Raychaudhuri. Distributed coordination schemes for multi-radio co-existence in dense spectrum environments: An experimental study on the orbit testbed. In *DySPAN*, 2008.

[61] Tae-Suk Kim, Taerim Park, Mo Sha, and Chenyang Lu. Toward mac protocol service over the air. In *GLOBECOM*, 2012.

[62] Youngmin Kim, Hyojeong Shin, and Hojung Cha. Y-mac: An energy-efficient multi-channel mac protocol for dense wireless sensor networks. In *IPSN*, 2008.

[63] Alexey Kiryushin, Aleksandr Sadkov, and Alan Mainwaring. Real-world performance of clear channel assessment in 802.15.4 wireless sensor networks. In *SENSORCOMM*, 2008.

[64] Kevin Klues, Gregory Hackmann, Octav Chipara, and Chenyang Lu. A component-based architecture for power-efficient media access control in wireless sensor networks. In *SenSys*, 2007.

[65] Dimitrios Koutsonikolas, Saumitra Das, Y. Charlie Hu, and Ivan Stojmenovic. Hierarchical geographic multicast routing for wireless sensor networks. In *SensorComm*, 2007.

[66] Sooyeon Shin Taekyoung Kwon, Gil-Yong Jo, Youngman Park, and Haekyu Rhy. An experimental study of hierarchical intrusion detection for wireless industrial sensor networks. *IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS*, 6(4):744–757, 2010.

[67] Hieu Khac Le, Dan Henriksson, and Tarek Abdelzaher. A control theory approach to throughput optimization in multi-channel collection sensor networks. In *IPSN*, 2007.

[68] Hieu Khac Le, Dan Henriksson, and Tarek Abdelzaher. A practical multi-channel media access control protocol for wireless sensor networks. In *IPSN*, 2008.

[69] Philip Levis and David Gay. Tinyos programming. April 13, 2009.

[70] C.-J. M. Liang, B. Priyantha, J. Liu, and A. Terzis. Surviving Wi-Fi interference in low power zigbee networks. In *SenSys*, 2010.

[71] Chieh-Jan Mike Liang, Nissanka Bodhi Priyantha, Jie Liu, and Andreas Terzis. Surviving wi-fi interference in low power zigbee network. 2010.

[72] Kaisen Lin and Philip Levis. Data discovery and dissemination with dip. In *IPSN*, 2008.

[73] S. Lin, J. Zhang, G. Zhou, L. Gu, T. He, and J. A. Stankovic. ATPC: Adaptive Transmission Power Control for Wireless Sensor Networks. In *ACM SenSys*, 2006.

[74] Jane W. S. Liu. Real-time systems. 2000.

[75] W. Lou. An efficient n-to-1 multipath routing protocol in wireless sensor networks. In *MASS*, 2005.

[76] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi. The Flooding Time Synchronization Protocol. In *ACM SenSys*, 2004.

[77] Pedro Jose Marron, Daniel Minder, Andreas Lachenmann, and Kurt Rothermel. Tinycubus: An adaptive cross-layer framework for sensor networks. In *Information Technology*, volume 47, pages 87–97, 2005.

[78] V. Mhatre and C. Rosenberg. Homogeneous vs heterogeneous clustered sensor networks: a comparative study. In *ICC*, 2004.

[79] MLA Source Code. `http://tinyos.cvs.sourceforge.net/viewvc/tinyos/tinyos-2.x-contrib/wustl/`.

[80] Monsoon Solutions. `http://www.msoon.com/LabEquipment/PowerMonitor/`.

[81] David Moss and Philip Levis. BoX-MACs: Exploiting physical and link layer boundaries in low-power networking. Technical Report SING-08-00, Stanford Information Networks Group, 2008.

[82] Luca Mottola, Gian Pietro Picco, and Adil Amjad. Figaro: Fine-grained software reconfiguration for wireless sensor networks. In *EWSN*, 2008.

[83] Vishnu Navda, Aniruddha Bohra, Samrat Ganguly, and Dan Rubenstein. Using channel hopping to increase 802.11 resilience to jamming attacks. In *Infocom Minisymposium*, 2007.

[84] R. Oliver and G. Fohler. Probabilistic estimation of end-to-end path latency in wireless sensor networks. In *MASS*, 2009.

[85] K. Papagiannaki, M. Yarvis, and W. S. Conner. Experimental characterization of home wireless networks and design implications. In *INFOCOM*, 2006.

[86] Polar Heart Rate Monitor. `http://www.polarusa.com/`.

[87] Joseph Polastre, Jason Hill, and David Culler. Versatile low power media access for wireless sensor networks. In *SenSys*, 2004.

[88] Joseph Polastre, Robert Szewczyk, and David Culler. Telos: Enabling ultra-low power wireless research. In *IPSN*, 2005.

[89] S. Pollin, M. Ergen, M. Timmers, A. Dejonghe, L. van der Perre, F. Catthoor, I. Moerman, and A. Bahai. Distributed cognitive coexistence of 802.15.4 with 802.11. In *Cognitive Radio Oriented Wireless Networks and Communications*, 2006.

[90] J. R. Quinlan. C4.5: Programs for machine learning. In *Morgan Kaufmann Publishers*, 1993.

[91] Iyappan Ramachandran and Sumit Roy. On the impact of clear channel assessment on mac performance. In *GLOBECOM*, 2006.

[92] Lei Rao, Xue Liu, Jian-Jia Chen, and Wenyu Liu. Joint optimization of system lifetime and network performance for real-time wireless sensor networks. In *QSHINE*, 2009.

[93] Injong Rhee, Ajit Warrier, Mahesh Aia, and Jeongki Min. Z-mac: a hybrid mac for wireless sensor networks. In *SenSys*, 2005.

[94] Anthony Rowe, Vikram Gupta, and Raj Rajkumar. Low-power clock synchronization using electromagnetic energy radiating from ac power lines. In *SenSys*, 2009.

[95] Anthony Rowe, Rahul Mangharam, and Raj Rajkumar. Rt-link: A time-synchronized link protocol for energy constrained multi-hop wireless networks. In *SECON*, 2006.

[96] Abusayeed Saifullah, Chengjie Wu, Paras Tiwari, You Xu, Yong Fu, Chenyang Lu, and Yixin Chen. Near optimal rate selection for wireless control systems. In *RTAS*, 2012.

[97] Abusayeed Saifullah, Chengjie Wu, Paras Tiwari, You Xu, Yong Fu, Chenyang Lu, and Yixin Chen. Near optimal rate selection for wireless control systems. In *RTAS*, 2012.

[98] Abusayeed Saifullah, You Xu, Chenyang Lu, and Yixin Chen. Real-time scheduling for wirelesshart networks. In *RTSS*, 2010.

[99] Abusayeed Saifullah, You Xu, Chenyang Lu, and Yixin Chen. End-to-end delay analysis for fixed priority scheduling in wirelesshart networks. In *RTAS*, 2011.

[100] Abusayeed Saifullah, You Xu, Chenyang Lu, and Yixin Chen. Priority assignment for real-time ows in wirelesshart networks. In *ECRTS*, 2011.

[101] Mo Sha. Empirical studies for reliable home area wireless sensor networks. *Master Thesis*, Computer Science and Engineering Department, Washington University in St. Louis, 2011.

[102] Mo Sha, Rahav Dor, Gregory Hackmann, Chenyang Lu, Tae-Suk Kim, and Taerim Park. Self-adapting mac layer for wireless sensor networks. In *RTSS*, 2013.

[103] Mo Sha, Gregory Hackmann, and Chenyang Lu. Arch: Practical channel hopping for reliable home-area sensor networks. In *RTAS*, 2011.

[104] Mo Sha, Gregory Hackmann, and Chenyang Lu. Multi-channel reliability and spectrum usage in real homes: Empirical studies for home-area sensor networks. In *IWQoS*, 2011.

[105] Mo Sha, Gregory Hackmann, and Chenyang Lu. Energy-efficient low power listening for wireless sensor networks in noisy environments. In *IPSN*, 2013.

[106] Mo Sha, Gregory Hackmann, and Chenyang Lu. Real-world empirical studies on multi-channel reliability and spectrum usage for home-area sensor networks. *IEEE Transactions on Network and Service Management*, 10(1), March, 2013.

[107] Mo Sha, Guoliang Xing, Gang Zhou, Shucheng Liu, and Xiaorui Wang. C-mac: Model-driven concurrent medium access control for wireless sensor networks. In *INFOCOM*, 2009.

[108] Rahul C. Shah, Lama Nachman, and Chieh-yih Wan. On the performance of bluetooth and ieee 802.15.4 radios in a body area network. In *BodyNets*, 2008.

[109] Shimmer Wireless ECG sensor. `http://www.shimmer-research.com/`.

[110] Soo Young Shin, Hong Seong Park, Sunghyun Choi, and Wook Hyun Kwon. Packet error rate analysis of ZigBee under WLAN and Bluetooth interferences. In *IEEE trans. on wireless communications*, 2007.

[111] Soo Young Shin, Hong Seong Parky, Sunghyun Choi, and Wook Hyun Kwon. Packet error rate analysis of IEEE 802.15.4 under IEEE 802.11b interference. In *WWIC*, 2005.

[112] Axel Sikora and Voicu F. Groza. Coexistence of IEEE 802.15.4 with other systems in the 2.4 GHz ISM band. In *IMTC*, 2005.

[113] Pablo Soldati, Haibo Zhang, and Mikael Johansson. Deadline-constrained transmission scheduling and data evacuation in wirelesshart networks. In *ECC*, 2009.

[114] Dongjin Son, Bhaskar Krishnamachari, and John Heidemann. Experimental study of concurrent transmission in wireless sensor networks. 2006.

[115] Kannan Srinivasan, Prabal Dutta, Arsalan Tavakoli, and Philip Levis. An empirical study of low power wireless. In *ACM Transactions on Sensor Networks*, 2010.

[116] Kannan Srinivasan and Philip Levis. RSSI is under appreciated. In *EmNets*, 2006.

[117] Stephen M. Stigler. Francis Galton's account of the invention of correlation. *Statistical Science*, 4(2), 1989.

[118] 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 *SenSys*, 2008.

[119] Arsalan Tavakoli, Aman Kansal, and Suman Nath. On-line sensing task optimization for shared sensors. In *IPSN*, 2010.

[120] Weka. `http://www.cs.waikato.ac.nz/ml/weka/`.

[121] K. Whitehouse, A. Woo, F. Jiang, J. Polastre, and D. Culler. Exploiting the capture effect for collision detection and recovery. 2005.

[122] WirelessHART, 2007. http://www.hartcomm2.org.

[123] C. Wu, M. Sha, D. Gunatilaka, A. Saifullah, C. Lu, and Y. Chen. Analysis of EDF Scheduling for Wireless Sensor-Actuator Networks. In *IWQoS*, 2014.

[124] Yafeng Wu, John A. Stankovic, Tian He, and Shan Lin. Realistic and efficient multi-channel communications in wireless sensor networks. In *Infocom*, 2008.

[125] WUSTL Wireless Sensor Network Testbed. `http://wsn.cse.wustl.edu/index.php/Testbed`.

[126] Guoliang Xing, Mo Sha, Jun Huang, Gang Zhou, Xiaorui Wang, and Shucheng Liu. Multi-channel interference measurement and modeling in low-power wireless networks. In *RTSS*, 2009.

[127] D. G. Yoon, S. Y. Shin, W. H. Kwon, and H. S. Park. Packet error rate analysis of IEEE 802.11b under IEEE 802.15.4 interference. In *VTC Spring*, 2006.

[128] Ossama Younis and Sonia Fahmy. Heed: A hybrid, energy-efficient, distributed clustering approach for ad-hoc networks. *IEEE Transactions on Mobile Computing*, 3(4):366–369, 2004.

[129] Wei Yuan, Jean-Paul M. G. Linnartz, and Ignas G. M. M. Niemegeers. Adaptive cca for ieee 802.15.4 wireless sensor networks to mitigate interference. In *WCNC*, 2010.

[130] Haibo Zhang, Pablo Soldati, and Mikael Johansson. Optimal link scheduling and channel assignment for convergecast in linear wirelesshart networks. In *IEEE International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks*, 2009.

[131] Xi Zhang, Junaid Ansari, Luis Miguel Amoros Martinez, Noemi Arbos Linio, and Petri Mahonen. Enabling rapid prototyping of reconfigurable mac protocols for wireless sensor networks. In *WCNC*, 2013.

[132] T. Zhong, C. Mengjin, Z. Peng, and W. Hong. Real-time communication in wia-pa industrial wireless networks. 2010.

[133] Gang Zhou, Chengdu Huang, Ting Yan, Tian He, John A. Stankovic, and Tarek F. Abdelzaher. MMSN: Multi-frequency media access control for wireless sensor networks. In *Infocom*, 2006.

[134] ZigBee Alliance. Zigbee and wireless radio frequency coexistence, 2007.

[135] Marco Zimmerling, Federico Ferrari, Luca Mottolay, Thiemo Voigty, and Lothar Thiele. ptunes: Runtime parameter adaptation for low-power mac protocols. In *IPSN*, 2012.

# Vita

Mo Sha

**Degrees**

Ph.D. Computer Science, August 2014
M.S. Computer Science, May 2011
M.Phil. Computer Science, August 2009
B.Eng. Computer Science, July 2007

**Publications**

Yong Fu, Mo Sha, Chengjie Wu, Andrew Kutta, Chenyang Lu, Humberto Gonzalez, Anna Leavey, Weining Wang, Bill Drake, Yixin Chen, and Pratim Biswas (2014). Thermal Modeling for a HVAC Controlled Real-life Auditorium. The 34th International Conference on Distributed Computing Systems (ICDCS 2014).

Chengjie Wu, Mo Sha, Dolvara Gunatilaka, Abusayeed Saifullah, Chenyang Lu, and Yixin Chen (2014). Analysis of EDF Scheduling for Wireless Sensor-Actuator Networks. The 22nd ACM/IEEE International Symposium on Quality of Service (IWQoS 2014).

Mo Sha, Gregory Hackmann, and Chenyang Lu (2013). Real-world Empirical Studies on Multi-Channel Reliability and Spectrum Usage for Home-Area Sensor Networks. IEEE Transactions on Network and Service Management (TNSM), Special Issue on Quality-of-Service, Volume 10, Issue 1, pp. 56-69, March 2013.

Mo Sha, Rahav Dor, Gregory Hackmann, Chenyang Lu, Tae-Suk Kim, and Taerim Park (2013). Self-Adapting MAC Layer for Wireless Sensor Networks. The 34th IEEE Real-Time Systems Symposium (RTSS 2013).

Mo Sha, Gregory Hackmann, and Chenyang Lu (2013). Energy-Efficient Low Power Listening for Wireless Sensor Networks in Noisy Environments. The 12th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN 2013).

Yong Fu, Mo Sha, Gregory Hackmann, and Chenyang Lu (2012). Practical Control of Transmission Power for Wireless Sensor Networks. The 20th IEEE International Conference on Network Protocols (ICNP 2012).

Tae-Suk Kim, Taerim Park, Mo Sha, and Chenyang Lu (2012). Toward MAC Protocol Service over the Air; The 55th IEEE Global Communications Conference (GLOBECOM 2012).

Mo Sha, Gregory Hackmann and Chenyang Lu (2011). ARCH: Practical Channel Hopping for Reliable Home-Area Sensor Networks. The 17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2011).

Mo Sha, Gregory Hackmann, and Chenyang Lu (2011). Multi-Channel Reliability and Spectrum Usage in Real Homes: Empirical Studies for Home-Area Sensor Networks. The 19th ACM/IEEE International Symposium on Quality of Service (IWQoS 2011).

August 2014