ARCH: Practical Channel Hopping for Reliable Home-Area Sensor Networks

Mo Sha, Gregory Hackmann, Chenyang Lu
Department of Computer Science and Engineering
Washington University in St. Louis
St. Louis, MO, USA
{sham,gwh2,lu}@cse.wustl.edu

***Abstract*—Home area networks (HANs) promise to enable sophisticated home automation applications such as smart energy usage and assisted living. However, recent empirical study of HAN reliability in real-world residential environments revealed significant challenges to achieving reliable performance in the face of significant and variable interference from a multitude of coexisting wireless devices. We propose the Adaptive and Robust Channel Hopping (ARCH) protocol: a lightweight receiver-oriented protocol which handles the dynamics of residential environments by reactively channel hopping when channel conditions have degraded. 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 results demonstrate that ARCH can reduce 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 routing scenario, ARCH reduced radio usage by 31.6% on average, by reducing the ETX of each link by up to 83.6%. Due to ARCH's lightweight reactive design, most links achieve this improvement in reliability with 10 or fewer channel hops per day.**

I. 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 [1]. 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 paper, 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 phenomenon.
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 paper is organized as follows. Section II reviews related work. Section III discusses the design of ARCH channel-hopping protocol. Section IV 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 V.

II. Related Work

In recent years, there has been increasing interest in using channel hopping to enhance MAC layer performance. SSCH [2] aims to improve network capacity by using channel hopping to prevent interference among simultaneous transmissions. [3] proposes a rapid channel hopping scheme to protect from jamming attacks in the 802.11a band. Other multi-channel protocols [4]–[9] 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. [10] 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 [11], Bluetooth's AFH [12], and ZigBee 2007's optional frequency agility [13], 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 pseudorandomly 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.

III. Protocol Design

In this section, we present the design of our Adaptive and Robust Channel Hopping (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 [14]. 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.

A. 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 [1].

1) 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 infrequent channel hopping.*

2) 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.*

3) 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.*

B. 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 III-C. When the ETX exceeds a specified $ETX\ Threshold$, the receiver node will select a new channel from the channel pool (see Section III-D) and initiate a channel hop. The receiver then notifies all of its senders of this channel hop using the mechanism discussed later in Section III-E.

To avoid the bursty packet loss observed in [1], 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 $Standby\ Channel\ Threshold$.

C. 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 [15]. 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 IV-A2 demonstrates that this approach can predict channel reliability with sufficient accuracy using as little as 15 minutes' worth of history.

D. 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.

E. Coordinated Channel Hopping

In [1], 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 III-D.

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 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.

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
```

F. 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[1]. 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.

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.

IV. 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 [16], 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.

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

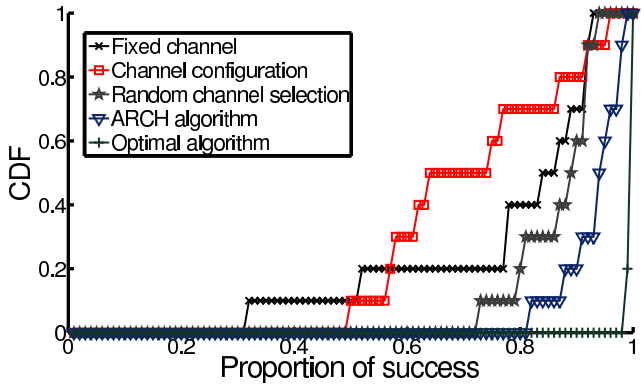
A. Simulator-Based Microbenchmarks

Our microbenchmark experiments were carried out in a C++ simulator environment, and are driven by two sets of link quality data previously collected for [1]. Both data sets were collected in 10 real-world apartments using networks of Tmote Sky and TelosB [17] 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.

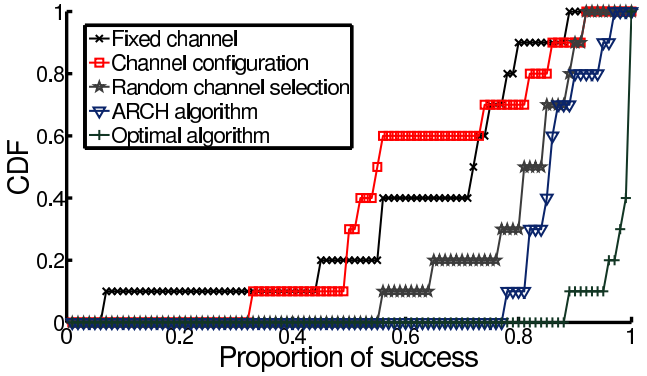
For the first data set, the transmitter used a data rate of 100 packets per channel every 5 minutes and no packet retransmissions. This provided high-granularity PRR data, which we used to evaluate ARCH's channel selection policy. For the second data set, the transmitter used a reduced data rate of 1 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.

1) 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



(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 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.

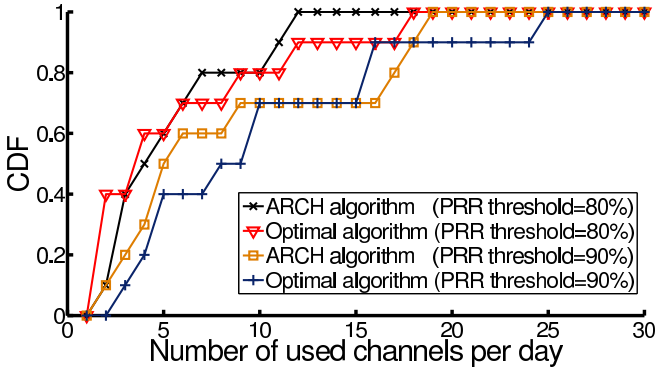


Figure 2. CDF of number of channel hops per day under ARCH and optimal channel selection schemes. Results were measured under simulation using experimentally-collected PRR traces from 10 apartments.

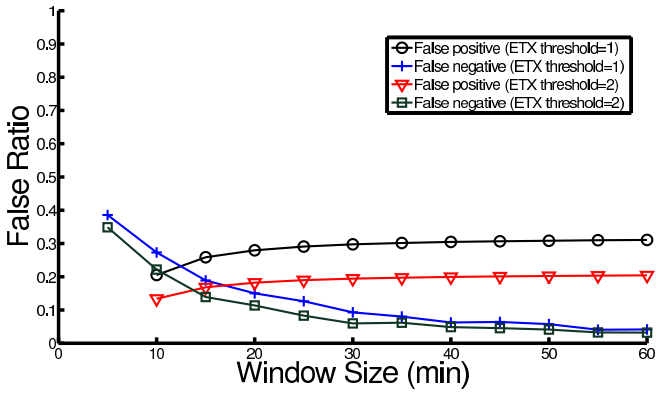


Figure 3. False-positive and false-negative rates of ARCH's channel estimation scheme. Results were measured under simulation using experimentally-collected ETX traces from 10 apartments.

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.

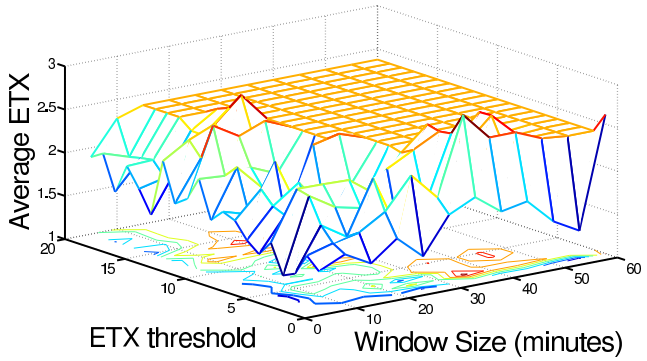
Figure 1(a) plots the CDF of the nodes' success, defined as the proportion of time that the node met the PRR threshold of 80%. On average, ARCH achieves 18% higher success than the fixed channel and channel configuration schemes. In addition, ARCH's channel selection and blacklisting schemes allow it to improve on the random channel selection scheme by 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 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.

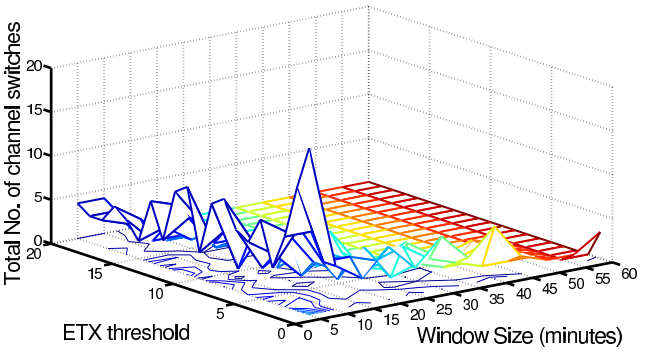
As shown in Figure 2, ARCH achieves this degree of reliability with relatively few channel hops. At most 25 channel switches are needed per link per day to meet the 90% PRR requirement, with a median of fewer than 10.

2) 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 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 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



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



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

Figure 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.

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 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.

B. 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.

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

Table I
ROM AND RAM USAGE COMPARISON OF OUR MULTI-HOP MACROBENCHMARK APPLICATION.

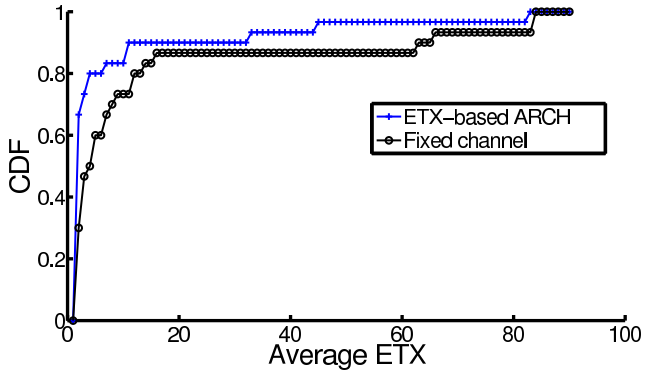
Owing to its lightweight design, ARCH introduces very little code size overhead. Table I shows the program ROM and RAM usage for the benchmark application described in Section IV-B2, as reported by the TinyOS toolchain. Compiling the application with ARCH enabled only consumes 480 extra bytes of ROM 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.

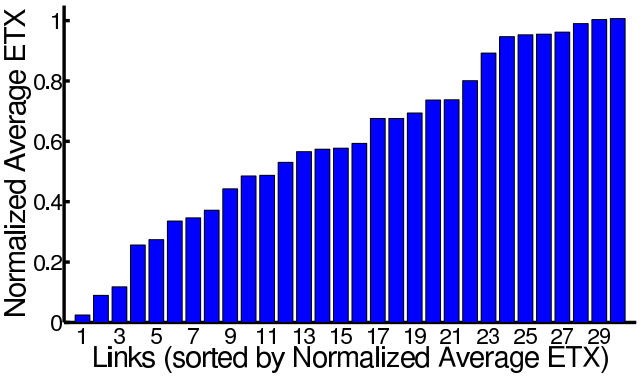
1) Single-Hop Data Collection: Our first experiment measured ARCH's performance under a simple single-hop data collection application. In this scenario, one node was designated as a data sink. The remaining nodes produced packets at a rate of 1 packet/5 minutes and attempted to deliver the packet directly 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 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 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 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



(a) CDF of average ETX of 30 links.



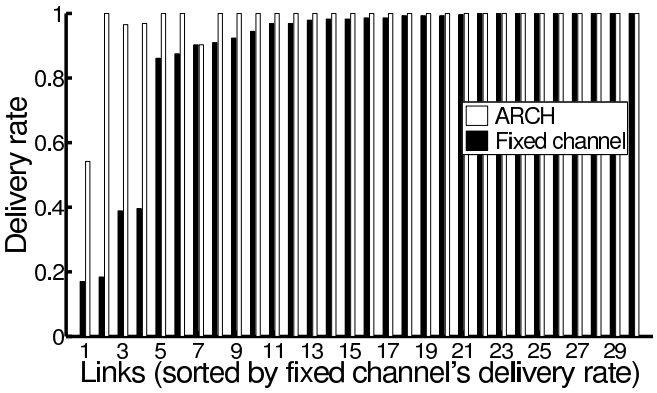
(b) Normalized average ETX (ARCH's divided by fixed-channel's ETX) of all links.

Figure 5. Comparison of ETX during single-hop data collection experiments under ARCH and fixed-channel schemes.

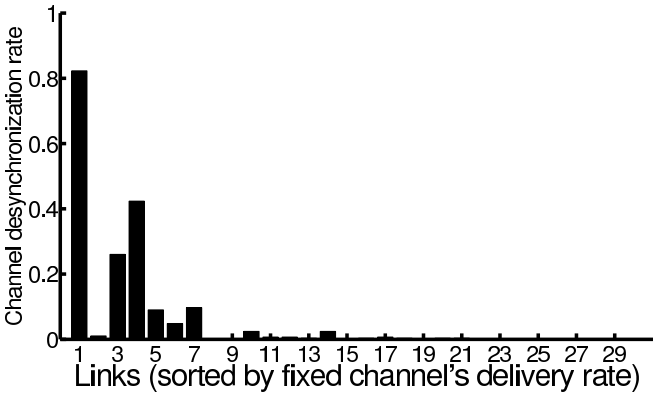
for only 21 links. ARCH also achieves a much higher minimum delivery rate (54.2% vs. 17.0%) than the fixed-channel scheme.

Figure 6(b) illustrates the number of channel desynchronizations detected for each corresponding link in Figure 6(a). (The links are sorted in the same order as in Figure 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 III-F, 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.

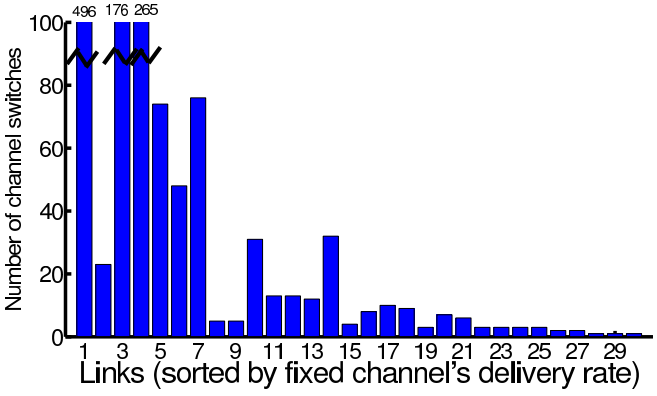
Figure 6(c) presents the overhead of ARCH in terms of



(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 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.

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 6(a) and 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 $2.5\times$–$5.6\times$ compared to the fixed-channel scheme.

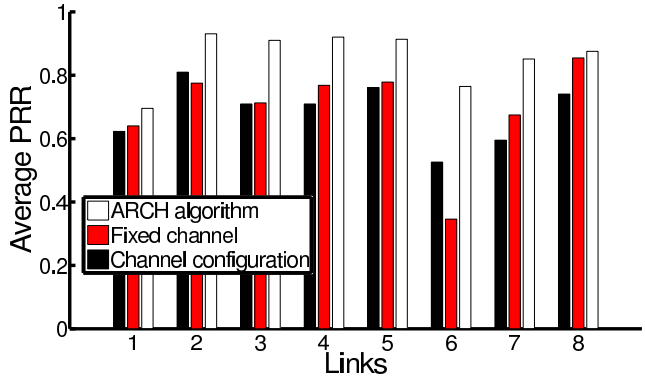
To further explore ARCH's real-world performance, we repeated the experiment using the BoX-MAC-2 [18] low power listening MAC layer. BoX-MAC-2 is a commonly-used protocol which automatically duty-cycles the radio in order to reduce nodes' idle listening cost. In order to directly measure the effect of 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 IV-A. 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 7(a) and 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 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 $1.6\times$ and $4.1\times$ as many packets as under ARCH, respectively, to maintain high reliability. Moreover, from Figure 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.

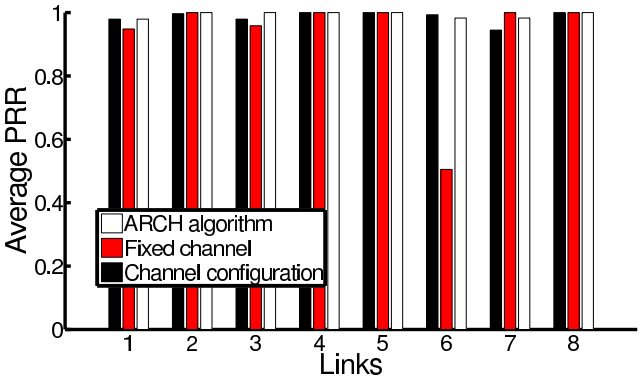
2) Multi-Hop Data Collection: Finally, we measured ARCH's performance under a multi-hop routing scenario. For this experiment, we deployed 10 sensor nodes between two floors of an apartment, as shown in Figure 8; an 11th node was deployed in the building's basement as the data sink. In order to isolate 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.

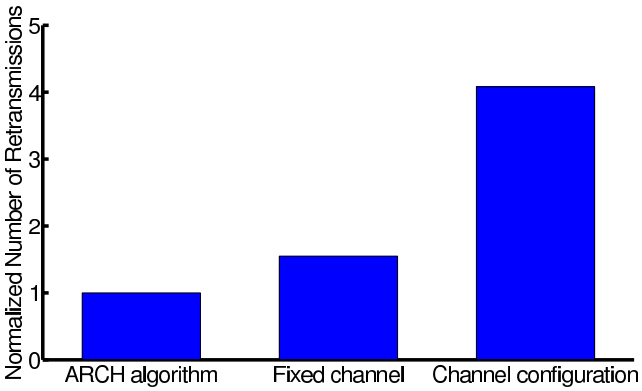
Figure 9 compares the end-to-end delivery rate under ARCH and the fixed-channel scheme. As with the previous



(a) Delivery rate of 8 links without retransmissions.



(b) Delivery rate of 8 links with retransmissions.



(c) Number of transmitted packets with retransmissions.

Figure 7. Performance comparison of ARCH, fixed channel, and channel configuration schemes under single-hop data collection when using BoX-MAC-2.

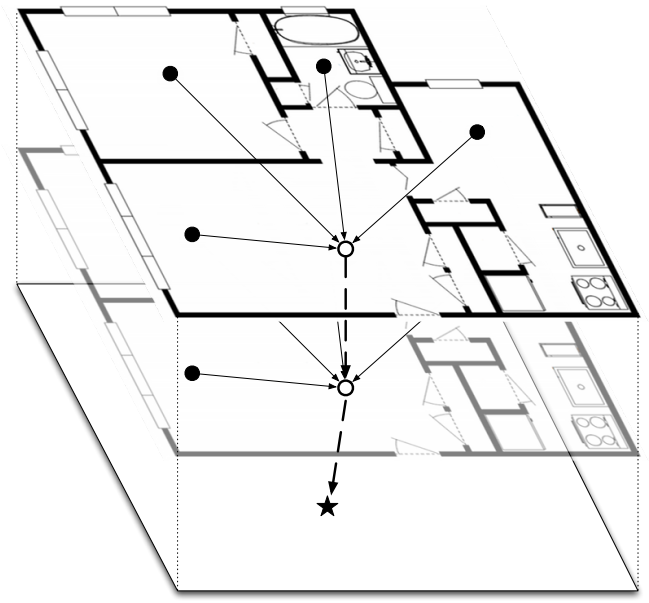


Figure 8. 3D diagram of sensor placement and collection tree for the multi-hop experiment. Nodes (circles) are laid out similarly on two floors, with a sink (star) in the basement. Hollow circles indicate relays.

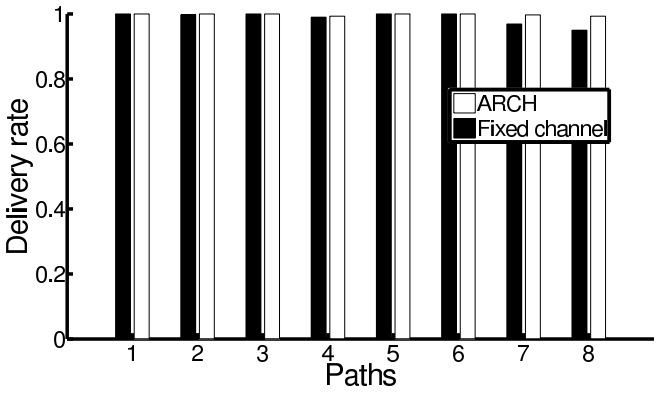
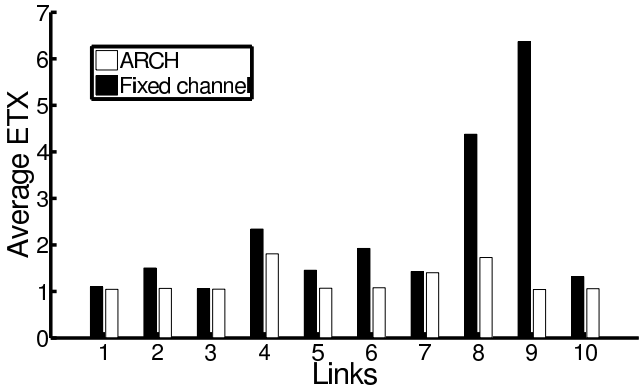


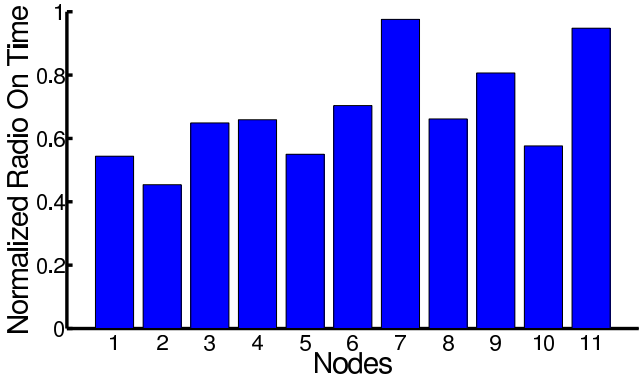
Figure 9. The delivery rate of all 8 multi-hop paths.

benchmark, the use of retransmissions enables uniformly high delivery 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 10, the fixed-power scheme incurred a much higher energy burden to achieve this degree of reliability. Figure 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 10(b) plots the ratio of radio usage between the two experimental runs. ARCH reduced the amount of time the



(a) The average ETX of all 10 links.



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

Figure 10. A comparison of energy efficiency between ARCH and fixed-power under multi-hop data collection.

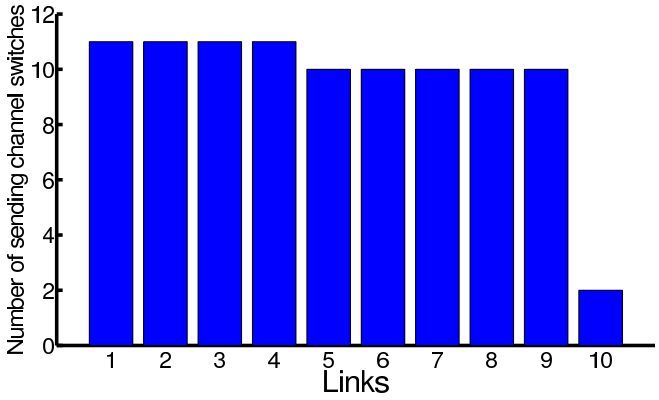


Figure 11. ARCH's overhead in terms of channel switches under multi-hop data collection.

radio was powered on by an average of 31.6%, representing a significant savings in energy consumption.

As shown in Figure 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.

V. 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.

Acknowledgment

This work is supported, in part, by NSF under grants CNS-0448554 (CAREER), CNS-0627126 (NeTS-NOSS), and CNS-1035773 (CPS).

References

[1] M. Sha, G. Hackmann, and C. Lu, "Multi-channel reliability and spectrum usage in real homes: Empirical studies for home-area sensor networks," Washington University in St. Louis, Tech. Rep. WUCSE-2010-32, 2010. [Online]. Available: http://cse.wustl.edu/Research/Pages/technical-reports.aspx

[2] V. Bahl, R. Chandra, and J. Dunagan, "Ssch: Slotted seeded channel hopping for capacity improvement in ad hoc networks," in *MobiCom*, 2004.

[3] V. Navda, A. Bohra, S. Ganguly, and D. Rubenstein, "Using channel hopping to increase 802.11 resilience to jamming attacks," in *Infocom Minisymposium*, 2007.

[4] H. K. Le, D. Henriksson, and T. Abdelzaher, "A control theory approach to throughput optimization in multi-channel collection sensor networks," in *IPSN*, 2007.

[5] ——, "A practical multi-channel media access control protocol for wireless sensor networks," in *IPSN*, 2008.

[6] Y. Kim, H. Shin, and H. Cha, "Y-mac: An energy-efficient multi-channel mac protocol for dense wireless sensor networks," in *IPSN*, 2008.

[7] Y. Wu, J. A. Stankovic, T. He, and S. Lin, "Realistic and efficient multi-channel communications in wireless sensor networks," in *Infocom*, 2008.

[8] G. Zhou, C. Huang, T. Yan, T. He, J. A. Stankovic, and T. F. Abdelzaher, "MMSN: Multi-frequency media access control for wireless sensor networks," in *Infocom*, 2006.

[9] G. Xing, M. Sha, J. Huang, G. Zhou, X. Wang, and S. Liu, "Multi-channel interference measurement and modeling in low-power wireless networks," in *RTSS*, 2009.

[10] J.-H. Hauer, V. Handziski, and A. Wolisz, "Experimental study of the impact of WLAN interference on IEEE 802.15.4 body area networks," in *EWSN*, 2009.

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

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

[13] ZigBee Standards Organization, *ZigBee specification, document 053474r17*, 2008.

[14] G. Hackmann, O. Chipara, and C. Lu, "Robust topology control for indoor wireless sensor networks," in *SenSys*, 2008.

[15] R. Fonseca, O. Gnawali, K. Jamieson, and P. Levis, "Four-bit wireless link estimation," in *HotNets VI*, 2007.

[16] http://www.tinyos.net/.

[17] J. Polastre, R. Szewczyk, and D. Culler, "Telos: Enabling ultra-low power wireless research," in *IPSN*, 2005.

[18] D. Moss and P. Levis, "BoX-MACs: Exploiting physical and link layer boundaries in low-power networking," Stanford Information Networks Group, Tech. Rep. SING-08-00, 2008.