Adaptive Time Synchronization and Frequency Channel Hopping for Wireless Sensor Networks

Branko Kerkez
Kristofer Pister

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2012-56
http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-56.html

May 2, 2012

Adaptive Time Synchronization and Frequency Channel Hopping for Wireless Sensor Networks

by Branko Kerkez

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Kristofer S.J. Pister
Research Advisor

Date

* * * * * *

Steven D. Glaser
Second Reader

Date

Acknowledgements

This thesis sprung up from the interest to immerse myself in the inner-workings of the wireless sensor networks (WSNs) that I was deploying as part of my Ph.D. dissertation in Civil Engineering. Inevitably, too much immersion led to discovery of open ended research topics. Exposure to these topics, as well as exposure to fundamental WSN theory would not have been possible without the support of Kris Pister's research group. Kris Pister and his group members were more than welcoming, and taught me invaluable lessons that have ultimately made me a better systems engineer. I would like to thank Thomas Watteyne for his continued support and patience during our work on OpenWSN and various papers. Almost all the papers resulting from this thesis were a direct product of a collaboration with Thomas. He has taught me the fundamentals of embedded programming, and has demystified to me the various tools necessary to implement complex embedded systems. Through his help I have learned the value of writing clean, organized, and modularized code, something which has fundamentally changed how I now develop software. I would like to thank the whole OpenWSN group, and especially Fabien Chraim, for his collaboration and for teaching me the importance of "really" reading data sheets. I would like to thank Ankur Mehta for his ideas on pushing TDMA networks towards mobile agent applications. I would also like to thank my Ph.D. adviser, Steven Glaser, for supporting my work in EECS, and allowing me to dabble in disciplines outside of the Civil Engineering "core." Last, but certainly not least, I would sincerely like to thank Kirs Pister, for graciously hosting me in his group, guiding my work, sharing his academic philosophies, and teaching me the value of counting micro-amps.

Contents

Part I

Frequency Agile Communications for Wireless Sensor Networks

1 Introduction

Most wireless sensor networks (WSNs) need to operate on battery power. As such, overall power consumption becomes a critical constraint for many deployments. Since radio transmissions consume the vast majority of a motes of power use [1] (not including sensors), it is important to design WSNs that will minimize radio up-time, while reducing the need for retransmissions due to external or environmental interference. By optimizing use of radio resources, and thus ensuring low power use, motes can be deployed to new or existing sensing infrastructure to provide real time wireless data access, without noticeably affecting overall power consumption. Successful design and operation of such a low-power field-deployed WSN hinges upon two rational rules:

- Keep the radio off whenever possible to conserve power, only turning on radio hardware when actively transmitting or receiving packets
- Mitigate the effects of external interference inasmuch as possible to reduce the need for packet retransmissions.

The implementation of these features will significantly improve network-wide reliability, while reducing drain on energy resources.

Duty cycling mote radio resources depends fundamentally on the ability of WSN nodes to share the same sense of time. If all nodes within a network are synchronized to the same clock, mote pairs can then be scheduled to only wake up when data transmissions need to occur. In such a scenario, motes can remain in a low power state while not actively transmitting data. Furthermore, the ability to schedule communications removes intra-network interference by ensuring that no two mote pairs

are scheduled to transmit packets at the same time. This manifests itself in tremendous energy savings (radio on-time less than 1 percent) when compared to most current WSN architectures, where the radio resources are liberally employed. As this thesis will lay out, this scheduling comes at the price of a higher software overhead at the MAC layer, but has significant power consumption benefits. A direct benefit of WSN synchronization relates to reliability, which not only stresses that that packet delivery guarantees be provided, but that packets should arrive at their intended destination with a minimal set of retransmissions.

Communication in WSNs is challenged by multipath radio propagation and narrow-band interference (e.g by other nearby radio sources) [2, 3]. The IEEE802.15.4 standard divides the 2.4 GHz band into 16 transmission channels (or, narrow-band sub-frequencies) that can be used to mitigate communication challenges posed by external interference and multipath propagation. Improper channel selection can cause data to be lost during transmission, leading to the need to retransmit data, and manifesting itself in lower packet delivery and higher energy requirements. A common approach to address this issue is to conduct an expert survey, which entails a physical visit to the field to select the frequency channel on which the loss of transmitted packets will be minimized. When a WSN deployment only uses one of the available channels for communication, multi-path propagation or external radio sources can cancel out the original radio signal, effectively eliminating the ability of certain mote pairs to communicate. The adverse effects of multipath propagation on radio communications are a time-varying function of the deployment environment, and it has been shown that even slight changes in node-to-node distance (on the order of centimeters) can have significant impact on this behavior [4]. An option to address this time-varying behavior involves equipping WSNs with the ability to channel hop. In such deployments, motes within the network select a different channel every time a transmission occurs, rather than persistently transmitting information on a single channel. As will be shown, the ability to ”schedule” channels naturally emerges from the tight time synchronization mentioned above.

The next sections will describe previous work and fundamental concepts behind time synchronization and channel selection within WSNs. The rest of this thesis will investigate advanced synchronization procedures, and adaptive channel selection algorithms, which aim to optimize reliability and power efficiency within low-throughput WSNs. The thesis will conclude by extending these developments to high-throughput, mobile, multi-agent WSN applications.

The specific contributions of this thesis can be summarized as follows:

- Show that TDMA-based synchronization and channel hopping can be achieved without the use of accurate crystal time sources
- Show the explicit benefits of an optimized channel hopping controller, which selects the best possible single channel based on current state information
- Implement and compare two realistic channel hopping schemes on a commonly available hardware platform
- Show how TDMA-based communications can be modified to support both low-throughput, persistent sensing applications, while facilitating high-throughput traffic from mobile network agents

2 Synchronization

The vast majority of communicating entities contain a clock source, which ticks at a frequency governed by a given application. For two such entities, the process of synchronization involves aligning the `phase` and the `period` of their clocks. Once this is achieved, the two devices effectively share the same sense of time, and can thus more effectively coordinate future tasks between each other. Environmental fluctuations, or imperfections in the clock construction, will cause one of the clocks to tick faster than the other. Even if this difference in clock frequency is small, it will eventually cause the two clocks to fall out of phase, widening the `synchronization error`, and thus requiring re-synchronization. The acceptable error is governed by an application, from seconds (e.g. a wrist watch) to nano-seconds (e.g. time-of-flight localization).

In the case of WSNs, network wide synchronization is often used for clock dissemination, which builds a common time base upon which to time-stamp sensor data. Typical multi-hop implementations use a Directed Acyclic Graph centered at a `time master`. As depicted in Fig. 1, each node elects a time parent to which it synchronizes. This also means that, the more hops a node is from the `time master`, the higher its overall synchronization error.

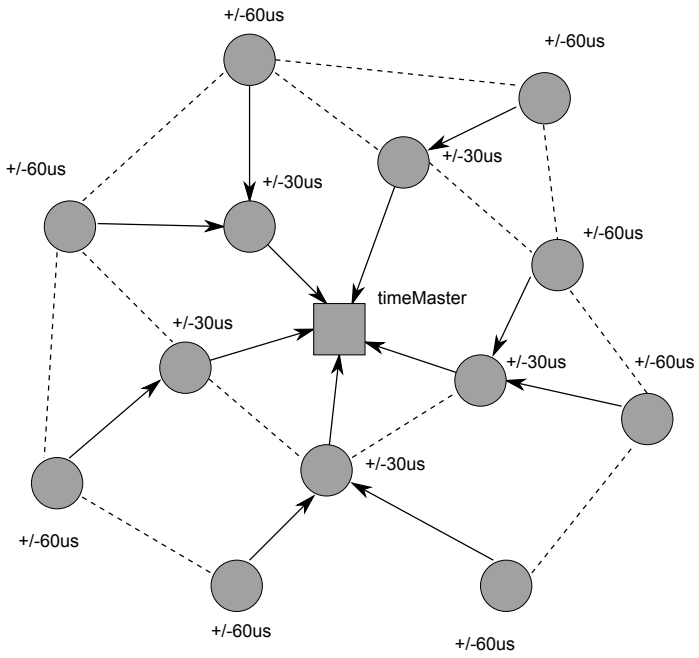

Figure 1: A Directed Acyclic Graph centered at a `time master` can be used to distribute timing information in a multi-hop topology. Every node elects a `time parent` to which it keeps synchronization.

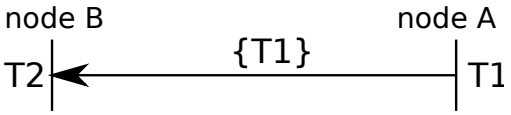

Figure 2: One-way synchronization.

The following sections detail the most common techniques used to synchronize two clocks: one-way, two-way, and slot-based synchronization. In all cases, we assume the two entities wishing to synchronization are able to send information to one another. These methods are different in the overhead required for implementation, but can successfully be applied for purposes of WSN synchronization.

2.1 One-way Synchronization

In most WSN deployments, a synchronization error of tens to hundreds of micro-seconds is acceptable. Such an error can be achieved via `one-way synchronization` (Fig. 2), a method which relies on accurate time-stamping of transmitted packets.

In Fig. 2, node B is desynchronized. Its clock is set to T_2, and it is synchronizing off node A, the time master, whose clock is set to T_1. Node A periodically transmits synchronization packets to its neighbors, thereby advertising its time to other nodes (here, node B). The last bytes of a synchronization packet contain a time-stamp field, which, prior to the transmission of the packet is set to an error code. Node A then sends that packet, including the error code, to its radio chip, and instructs the radio to send the packet. As soon as the the first byte leaves the radio, an interrupt line goes high, alerting node A's micro-controller, which timestamps this instance as T_1. While the packet is still leaving the radio, the micro-controller replaces the error code by T_1. As soon as node B receives the packet, it timestamps T_2 and uses equation 1 to determine its synchronization error:

$$error = T2 - T1 \tag{1}$$

This technique relies on the fact that the packet is long enough to ensure the micro-controller has enough time to overwrite the error code before the packet is completely transmitted. If, for some reason, this is not the case, node B will receive the error code, thus instructing it not to ignore the synchronization request. This one-way synchronization technique is used by the Flooding Time Synchronization Protocol (**FTSP**) [5].

2.2 Two-way Synchronization

Traditional computer networks utilize the Network Time Protocol (**NTP**) [6] to synchronize system clocks. NTP follows a generalized client-server architecture, where client computers contact time servers to ask for the time. Clients typically re-contact the server at regular intervals. In the wired Internet, round-trip delays of hundred of milliseconds are commonplace, so by the time the server's response reaches the client, time has passed and the time-stamp becomes inaccurate.

To cancel the effect of non-negligible propagation delays, NTP uses a technique known as `two-way synchronization`. The same method can be implemented in WSNs, and is illustrated in Fig. 3, in which node B is synchronizing its clock to node A's. Two-way synchronization

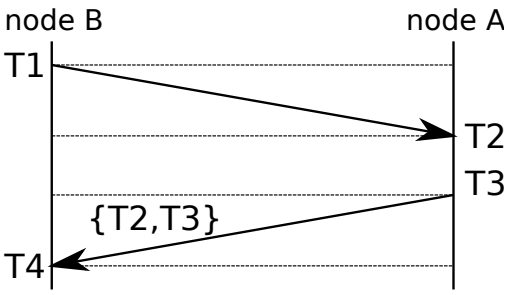

Figure 3: Two-way synchronization is used to evaluate the offset between two clocks when the propagation delay is not negligible, see quation 2.

relies on the ability to accurately time-stamp transmission and reception of packets. Node B begins by transmitting a packet to node A, noting at what time T_1 it did. Node A time-stamps the reception of this packet at T_2, and replies with an acknowledgment packet containing T_2, as well as T_3, the time the reply was transmitted. At reception, node B timestamps the arrival of the reply as T_4, and uses 2 to re-align its clock by computing the propagation delay and the synchronization error between its clock and that of node A.

$$\begin{cases} delay &= \frac{(T4-T1)-(T3-T2)}{2} \\ error &= \frac{(T2-T1)+(T3-T4)}{2} \end{cases} \tag{2}$$

While NTP version 4 [6] can theoretically achieve micro-second accuracy, it is often limited by potential software delays. This is mitigated by the **IEEE1588** standard [7], which is based on the same two-way synchronization method, but implemented in hardware. This causes any delays introduced by network elements to be largely deterministic. [8] presents an implementation of this standard, with experimental results indicating synchronization errors on the order to tens of nano-seconds.

Very tight synchronization is also being used in low-power WSNs for Radio-Frequency Time-of-Flight (**ToF**) ranging. [9] presents a prototype radio which uses the equivalent of two-way synchronization to measure the time it takes for an wireless signal to travel between two radios. Because its takes that signal only $1ns$ to travel $30cm$, extremely tight synchronization is required.

2.3 Slot-based Synchronization

Since clock drift causes WSNs nodes to de-synchronize, it may appear that that network-wide synchronization may not be energy-efficient, and that the overhead associated with synchronization adds unnecessary complexity to the wireless node firmware. However, a robust synchronization procedure, coupled with a well designed communication schedule, can play a significant role in the reduction of the radio duty cycle, and thus directly facilitates lower energy consumption [10]. Unlike GPS techniques, a synchronization error of tens to hundreds of micro-seconds is acceptable for use within WSNs [11]. Combined with the fact that propagation delays are almost negligible in wireless systems, this implies that two-way synchronization is not required, and a simpler one-way synchronization scheme can be employed.

Time-Division Multiple Access (**TDMA**) is a synchronization technique that relies on an agreed upon transmission schedule between network nodes (Fig. 4). Time is sliced up into `time slots` of equal length; a constant number of slots make up a `slot frame` which repeats indefinitely over time. Once synchronized, network node pairs are scheduled to exchange communications at a specified time slot, and `channel offset` (frequency channel) within the repeating slot frame. Every slot has an $absoluteslotnumber$ (ASN),which counts the number of slots that have expired since the the network has been initiated. A major requirement stipulates that no two node pairs can communicate during the same time slot and on the channel offset, thus ensuring collision-free communication.

One-way synchronization can be significantly simplified if the network operates in such a time-slotted manner. Instead of exchanging `explicit` timestamps, all nodes agree that during a scheduled transmission `any` packet will be transmitted exactly `TX Offset` after the beginning of a slot (Fig. 4). Every packet is thereby `implicitly` timestamped as being sent `TX Offset` after the beginning of the slot. In Fig. 4 node B is synchronizing to its time parent A, by time stamping received packet and comparing this time stamp to the expected `TX Offset`. This allows the computation of the synchronization error computed using equation 1. Node B then re-synchronizes by offsetting the `phase` of its slot frame (Fig. 5) to match that of node A. Provided the schedule is built correctly, TDMA can significantly increase the throughput of a WSN, while reducing energy consumption and packet collision rate [12].

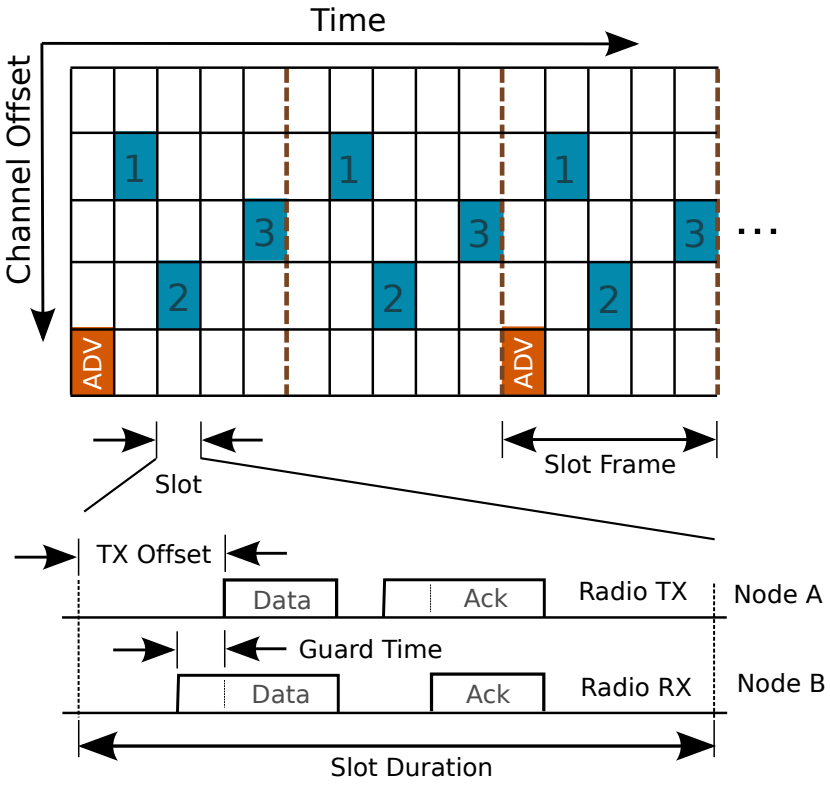

Figure 4: Slot-based communication splits time into repeating frames, which repeat indefinitely. Nodes are scheduled to communicate at specific slots and frequency channels within the frame. All packets are transmitted exactly `TX Offset` after the beginning of a slot. A node synchronizes to its `time parent` by time-stamping arriving packets and comparing them to the expected value TX Offset. In the event that two nodes are slightly de-synchronized, a `guard time` is used to turn on a receiving node's radio before the expected arrival of a packet to ensure that an incoming transmission is not missed.

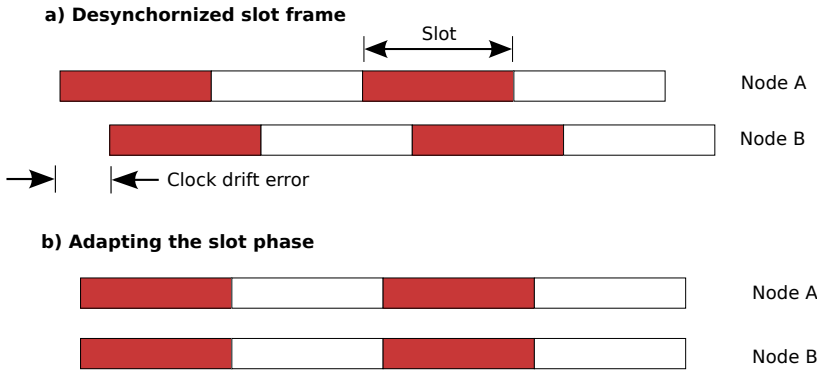

Figure 5: During slot-based synchronization, a node offsets the phase of its `slot frame` to match that of its time parent.

3 Channel Hopping

As stressed previously, communication reliability in Wireless Sensor Networks (WSNs) is challenged by narrow-band interference (WiFi, Bluetooth, etc.) and persistent multi-channel fading. Low reliability often manifests itself in needless packet retransmissions, and thus higher energy consumption. This problem can be mitigated through diversity in the routing protocol, by intelligently routing packets to mitigate areas of low connectivity [13]. This, however, may often not be enough, and further methods to improve overall connectivity are desired. Another option to mitigate communication difficulties is offered through frequency diversity. This option, also known as **channel hopping**, involves periodically changing the communication channel to cope with unexpected losses in link quality. As the next sections will show, frequency diversity is essential to deploying reliable, scalable, and low-power WSNs, and while current hardware platforms are up to the task, the ability to channel hop does come at a greater, but worthwhile, software implementation overhead.

3.1 Connectivity metrics and real-world traces

We call a **link** two motes that can communicate directly (i.e. an arrow in Fig. 6); links are directional. Health reports indicate how many transmissions (packets) where attempted, and how many of these transmissions were successful. A link failure is detected when the transmitter senses an occupied channel before a transmission, or when it does not receive an acknowledgment after transmitting the data. The collection of failure statistics is useful both for real-time WSN health reports and debugging, as well as for future research and post-analysis of network problems.

A measure of the goodness of a link between two motes can be given through the Packet Delivery Ratio ($\mathcal{P} \in [0, 1]$), i.e. the ratio between the number of successfully transmitted packets and the number of transmission attempts. $\mathcal{P} = 0.5$ would indicate that half transmitted packets on a link are received, or, alternatively, that there is a 50 percent chance that a transmission will be successful. The PDR will be a key indicator, which will guide the algorithms presented in the next sections.

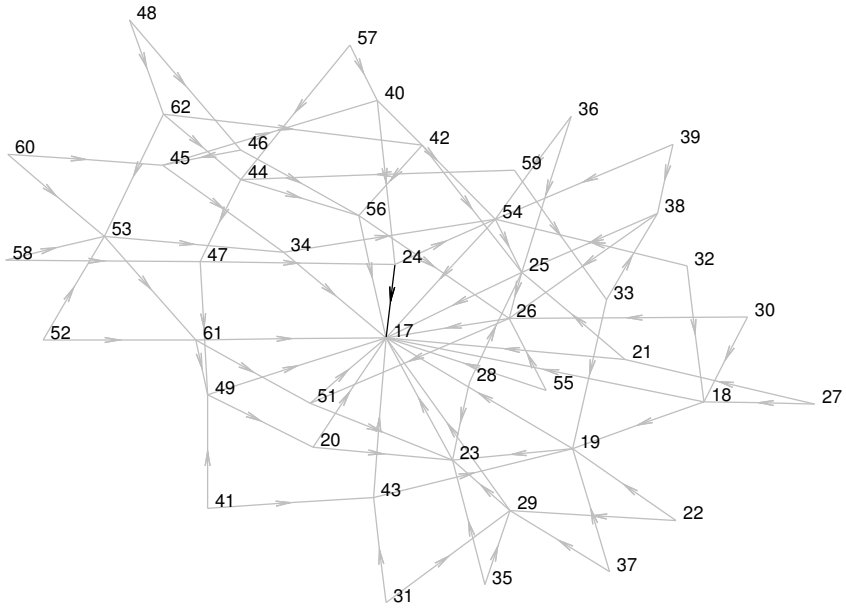

Figure 6: The routing di-graph over which the connectivity data is collected in [14]. Directed arrows represent active routing links (The network [12] assigns at least two routing parents to each node). Node 17 is the sink node.

Simulations based on propagation models may not provide enough information regarding the real-world behavior of WSN links. An alternative is offered by *replaying* communication algorithms on *connectivity traces* gathered from real-world deployments. Since connectivity varies over time, such approaches [13, 15, 16] use dense real-word data-sets to evaluate algorithms in the same conditions. Fig. 7 depicts $\mathcal{P}$ for 16 available channels, for a number of links in a deployment carried out by J. Ortiz and D. Culler in a UC Berkeley. Network wide PDR was calculated to be around 80 percent, but as the figure shows a number links contain channels with low PDR values. PDR can also fluctuate over time for any given channel, suggesting that no single channel may be entirely adequate for transmission.

Another significant connectivity data set was collected by [14]. This is currently one of the few comprehensive datasets which is both sufficiently dense in time, and collected over all IEEE802.15.4 channels. To obtain that dataset, 44 nodes running the Time Synchronized Mesh Protocol (TSMP) [12] were deployed in an indoor printing facility for 28 days. The routing mechanism assigned two parents to each node, creating the routing di-graph depicted in Fig. 6. Fig. 8 depicts the evolution of $\mathcal{P}$ with time for a given link, over all 16 channels. It indicates that activ-

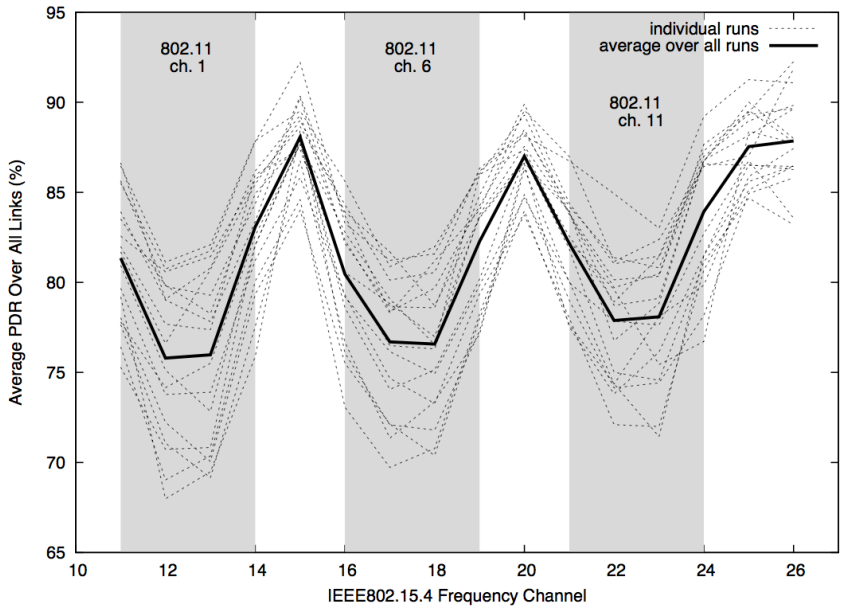

Figure 7: PDR values for all 16 IEEE15.4 channels. IEEE802.11 Wifi interference can be noticed in the gray areas.

ity inside the printing facility induced significant changes in Packet Delivery Ratio; this may also explain why $\mathcal{P}$ is stable over week-ends.

Fig. 8 illustrates the challenges faced by a single channel technique. Overall, every channel undergoes deep fading periods, rendering single channel solutions inadequate[1]. Choosing the methodology for utilizing all of the available channel space optimally is non-trivial. Radio hardware must be able to change the transmission channel adequately fast, and an agreement must exist between nodes to exchange information on a specific channel at a specific time. While not an often utilized feature, current IEEE802.15.4 radios do support agile frequency channel selection. A number of methods have also been proposed to standardize MAC layer enhancements to take advantage of a wider channel set.

[1]Strictly speaking, channel 17 on link $24 \rightarrow 17$ features a constant high $\mathcal{P}$. Nevertheless, a single channel MAC protocol assigns the same channel to all links, and, as shown in Section 17, there is no channel which works well at all times, for all links.

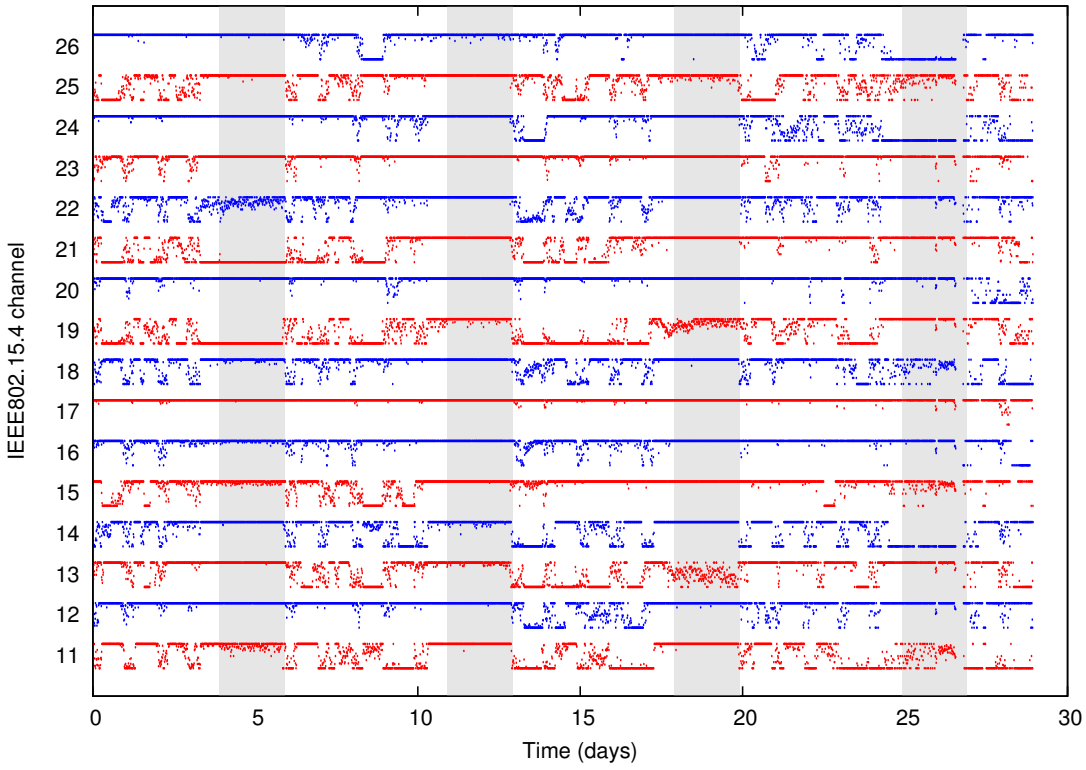

Figure 8: Packet Delivery Ratio evolving over time on the $24 \rightarrow 17$ link (colored black in Fig. 6), for all 16 channels. The grayed out time intervals represent week-ends.

3.2 Hardware support

Frequency-agile communication require nodes to change the frequency channel they transmit on or listen to regularly. Luckily, radio chips have become very efficient at doing this. As an example, all IEEE802.15.4-compliant [17] radio chips – the *de facto* standard for WSN hardware – switch channels in less than $192\mu s$. Such chips live in most popular platforms, such as TelosB, MICAz, IRIS, SUNspot, IMote or EPIC. Moreover, non-IEEE802.15.4 chips such as Texas Instruments' CC1100, CC1101 and CC2500 [18] feature low turn-around times of $88.4\mu s$. Combined with the fact that typical clocks drift by $10ppm$ or less, fast channel hopping capabilities have made frequency-agile communication very feasible.

3.3 Standardization efforts and frequency-agile MAC Protocols

Critical applications require reliable solutions, and as the previous section showed, channel hopping is one answer to this need. Various professional bodies have been working on on addressing this need through the development of a number of standards.

3.3.1 Major standards

IEEE802.15.1 [19] is the technology used by the Bluetooth consortium. The physical layer features 79 1-MHz channels in the 2.4GHz ISM band. Devices wishing to communicate group around a leader and synchronize to that leader's clock. Time is sliced up into $625\mu s$-long slots, and a hashing function translates the leader's address into a channel hopping pattern. All nodes follow that pattern blindly, changing channels roughly 1600 times per second.

The HART Communication Foundation standardizes embedded networking solutions for industrial applications. Their wireless extension, called **WirelessHART** [20], uses a central controller to schedule communication. WirelessHART uses IEEE802.15.4 radios to blindly hop on 15 frequency channels in the 2.4GHz band. Reliability is increased by having each node maintain connectivity to at least two parent nodes in the routing graph, enabling the network to resist link failures.

Another industrial wireless standardization body is the ISA100 Wireless Compliance Institute. Their latest standard **ISA100a** [21] is similar in essence to TSMP or WirelessHART, and features a uniquechannel hopping mechanisms. Successive channels in the hopping pattern are separated by at least 15 MHz (three IEEE802.15.4 channels). When retransmissions occur, they will not encounter or cause interference in the same IEEE802.11 – Wi-Fi – channel. The hopping pattern is set manually during network ramp-up, and is not dynamic.

3.3.2 MAC Layer enhancements

Lightweight MAC [22] (**LMAC**) assigns slots to nodes in a distributed way. **Multichannel LMAC** [23] proposes, when all slots are assigned, to pick a slot on another – randomly chosen – frequency. The number of potential slots is roughly multiplied by the number of frequency channels, which allows more nodes to communicate than LMAC. Omnet++ simulations show that the use of multiple channels decreases the number of active nodes while reducing collisions.

Y-MAC [24] is primarily designed to decrease latency. Nodes are synchronized and reception slots are assigned to each node on a common base channel. In case multiple packets need to be sent between neighbor nodes, successive packets are sent on a different frequency. As a result, bursts of messages ripple across channels, which reduces latency. Y-MAC was implemented in the RETOS operating system on the TmoteSky motes and compared to LPL. With $8s$ resynchronization period and 5 frequency channels, the idle duty cycle when sending one packet every $10s$ is around 7%.

The workgroup **IEEE802.15.4E** focuses on enhancing the MAC protocol proposed in IEEE802.15.4, while keeping the same physical layer. In its current proposal [25], nodes can switch between different hopping sequences. Slots can be added/removed during the lifetime of the network. An open-source implementation of this proposal is presented in [26]. This implementation has been part of Berkeley's open source *OpenWSN* project (opwnwsn.eecs.berkeley.edu). While beyond the scope of this thesis, the *OpenWSN* stack features a full IPv6 stack with a hardware independent IEEE802.15.4 MAC layer. Work on *OpenWSN* was a central guiding point for this thesis, and motivated many of the resulting papers.

4 Time synchronized channel hopping: lower power and reliability through TDMA-based communications

As shown above, long-term persistent wireless sensing can greatly benefit from both time synchronization and channel hopping. The two concepts complement each other, and can be combined under the larger umbrella of *Time Synchronized Channel Hopping* (TSCH). Some notable studies have explored the use of mulithop WSNs for purposes of long-term persistent sensing. A number of such systems were Time Division Multiple Access (TDMA) based networks which support both time and frequency division to facilitate robust communications while permitting for low battery consumption [10, 27]. Successful applications have ranged from industrial control [28], military applications [29], and habitat monitoring [30]. The majority of these studies was built upon the IEEE802.15.4 stack [31], which specifies hardware and software requirements to be met to facilitate interoperability between low-power wireless devices.

A number of key factors, emerge when considering the needs imposed on wireless sensing infrastructure:

- Reliability: Guarantees must be given to ensure reliable data packet data delivery, and the network should have methods to mitigate the effects external radio interference.
- Low power consumption: Nodes have to meet long-term deployment goals while running only on batteries.
- Scalability: The network should support an arbitrary number of nodes.
- Security: Communication should be encrypted to protect data integrity and mitigate malicious attacks.

These requirements are all met by the comprehensive Time Synchronized Mesh Protocol (TSMP) [10]. Versions of this protocol has also been standardized under the IEEE15.4E workgroup [32], as well as the Wireless Hart Foundation [20], and ISA100 [21]. At its core, TSMP ensures reliable, secure, and scalable wireless communications by combining very tight time synchronization, with frequency channel hopping and routing diversity.

Time Synchronized Mesh Protocol (**TSMP**) [12] was designed to improve reliability. TSMP employs tight time synchronization as well as channel hopping: different links use different frequency channels and the same link hops during its lifetime across different channels. This reduces the impact of narrow-band interference and persistent multipath fading. [14] presents experimental results in which 44 nodes run TSMP for 28 days in a printing facility. The authors show how channel hopping, combined with a retransmission policy, yields an end-to-end delivery ratio of 99.999%. TSMP uses a central coordinator which retrieves the list of nodes, their neighbors and their traffic requirements. This allows it to construct a schedule which is then communicated back to the network.

At its core, TSMP is a time division multiple access (TDMA) architecture and relies on a number of MAC-layer enhancements (and thus larger implementation overhead) to facilitate synchronization and frequency channel diversity. TSMP relies on slot based synchronization (shown previously) and an agreed upon transmission schedule between network nodes (Fig. 4). Time is sliced up into `time slots` (10ms in most implementations) of equal length; a constant number of slots make up a `slot frame` which repeats indefinitely over time. Once synchronized, network node pairs are scheduled to exchange communications at a specified time slot, and `channel offset` (frequency channel, 16 of which are stipulated by IEEE15.4E) within the repeating slot frame. A major requirement stipulates that no two node pairs can communicate during the same time slot and on the channel offset, thus ensuring collision-free communication. Since nodes only communicate when scheduled, they keep their radios powered off most of the time. This low duty cycle plays a significant role in keeping battery consumption to a minimum, theoretically permitting a node to last years on a standard AA battery. Such a TDMA-based network can significantly increase the throughput of a WSN, while reducing energy consumption and packet collision rate [10].

A TSMP network is centrally scheduled by a network manager. When a node first joins a network, it leaves its radio on and listens for advertisements (ADV packets, see figure 4) from its neighbors. Upon hearing an ADV packet, the node synchronizes to the network by the adjusting its $slotframe$ to reflect that of its time parent. The network manager than assigns this new node a transmission schedule. The network manager may allocate more $slots$ to a node to increase throughput. Due to clock drift, nodes are also required to re-synchronize regularly. The joining and synchronization process may take some time, depending on the size of the network, external

interference, and the length of the slotframe. As such, a traditional TSMP network does not lend itself readily to applications which require mobile agents and fast response times. The TSMP routing table, and transmission schedule are updated at most once every slot-frame.

Aside from supporting scheduled communications between neighboring nodes, TSMP also support multihop communications via the Routing Protocol for Low power and Lossy Networks (RPL). RPL is a gradient-based routing algorithm which provides multiple data paths between nodes. Data flows, one hop at a time from one node to another. This permits nodes that would otherwise be out of range to communicate with each other. While TSMP architecture is designed to facilitate low throughput communications, and we will now show how it can be augmented to facilitate sporadic bursts of large data streams. A TSMP-type WSN architecture is assumed for the remainder of this thesis. The following parts of this thesis will layout significant improvements to the MAC layer of TDMA-based architectures, which can bed used to reduce synchronization cost, and improve channel selection.

Part II

Crystal-free Time Synchronization

Motivated by the `Smart Dust` vision [33], the past decade has seen a significant reduction in the size of wireless sensor network (WSN) nodes. While a number of compact platforms (incorporating sensing, computation, and communication) have been developed, the goal of the cubic-millimeter sensor node [34] has yet to be realized. The `System-on-Chip` (SoC) WSN node has been proposed [35] to address such a small form-factor while providing reliable, low-power operations. Such a solution envisions the ability to print all the components of the sensor node (including micro-processor, radio and sensors) on one die of silicon.

The means to fabricate such a SoC appears within reach. Advances in MEMS packaging [36, 37] allow sensors and other components to be bundled efficiently at small scales. On-chip antennas [38] can be combined with square-millimeter radios [39] to facilitate communications, while thin-film battery technology [40] can be employed to power the sensor node. A major last hurdle to the cubic-millimeter SoC relates to reliable clock sources, and oscillator design.

WSN nodes often employ external quartz crystal oscillators as a time-keeping source (see Fig. 9). The use of such crystals facilities node-to-node synchronization, network-wide time-stamping, time division multiple access (TDMA) communications, and frequency hopping techniques. Including such a quartz oscillator in a cubic-millimeter SoC WSN mote is both complex and costly. Alternatively, it is much easier to use a simple resistive-capacitive (RC) circuit as a clock source, as they can be printed on-chip. Their output frequency, however, varies greatly due to manufacturing inaccuracy, as well as environmental factors such as fluctuations in temperature and supply voltage. MEMS-based oscillators present yet another option [41,42]. Such oscillators can be fabricated on-chip, but currently do not provide high enough frequencies to source radio transmissions, while also suffering from high temperature drift, and long-term stability issues.

Rather than placing emphasis on the need for new physical components, this chapter shows that a system-level approach can be employed to utilize ubiquitous low-power digital, RC-type oscillators technology to conduct TDMA-based WSN synchronization. We outline a synchro-

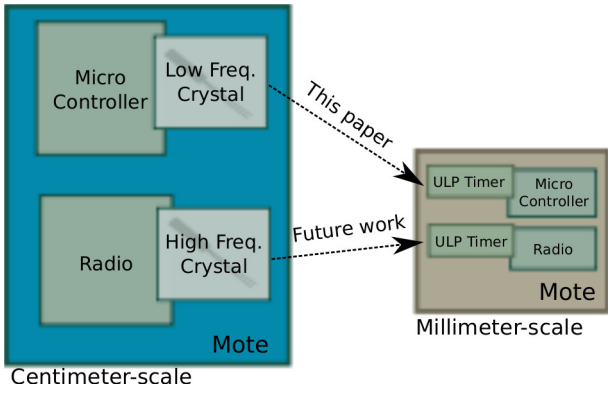

Figure 9: A system-on-chip WSN node demands the removal of typically employed oscillator crystals: a high frequency crystal is used for radio transmissions, while a second, lower-power, crystal is used to facilitate TDMA communications.

nization technique to compensate for the inherent drift and instabilities of RC-based oscillators. This RC circuit thereby becomes an `Ultra Low Power` (ULP) timer, replacing the typically employed quartz resonator (Fig. 9) for WSN synchronization. ULP timers are now capable of operating in a sub-nW range [43], having the potential to significantly reduce power consumption compared to even some of the most power efficient crystal-based solution. Recently, Mehta et al. 2011 [44] also laid the theoretical groundwork to facilitate the removal the higher-frequency crystal typically sourced by a mote's radio, thereby motivating the `crystal-free radio`. Though future work is required to experimentally verify such methods, combined with our synchronization approach, currently existing hardware could be used to create a truly single-chip, crystal-free, cubic-millimeter sensor node.

In this chapter, we experimentally validate our proposed algorithms by operating a multi-node WSN on a crystal-free TDMA schedule, while showing that time synchronization can readily be implemented on low-end nodes. To the best of our knowledge, this work is the first to demonstrate synchronization with nodes not equipped with a crystal oscillator. Through a technique we call `adaptive synchronization`, we achieve synchronization accuracies similar to the ones obtained with a crystal time-source, while utilizing an on-chip digital oscillator which has a drift four orders of magnitude higher. We introduce novel synchronization ideas such as `synchronization bootstrapping`, all of which have been implemented on real-world hardware, and which can be used directly along with existing TDMA synchronization protocols and standards.

Section 2 presented a review WSN synchronization techniques, with a particular emphasis on TDMA-based methods. To motivate the problem, and to understand the limitations of the clock sources at hand (in this case, digitally controlled oscillators internal to a micro-controller), section 5.2 analyzes measurements of inter-device variability, and carries out a study of oscillator clock drift as a function of temperature and supply voltage. Section 6 formalizes the proposed crystal-free TDMA-based synchronization method. The efficiency of this technique is verified experimentally on low-end hardware, by analyzing the performance of a multi-node deployment. Furthermore, in section 7.2, we present optional techniques to improve the convergence speed of `adaptive synchronization` and to decrease the power consumption of network nodes. Section 10 concludes this chapter and outlines the future steps necessary for the realization of the millimeter-cubed sensor node.

5 Hardware Considerations

5.1 Clock Stability and Time-Stamping

A robust implementation of `Time Synchronized Channel Hopping` for WSNs demands reliable hardware resources to ensure accurate time-stamping of packets. Once synchronized, nodes require a stable low-powered clock to ensure that they stay synchronized to their time parent, while being able to keep the majority of resources (micro-controller and radio) powered off to conserve energy. The stability of a clock is quantified by its `drift`, typically measured in parts-per-million (ppm), and corresponding to the difference in frequency relative to another clock source. For example, a drift of $10ppm$, indicates that every $100s$, two clocks will move $1ms$ apart. Crystal oscillators rely on a piece of crystal, typically quartz, resonating at a given frequency. The quality of the cut of that crystal impacts its drift. For typical WSN synchronizations, low power crystal oscillators operate at 32768Hz, with reliable crystals exhibiting a drift of $10ppm$.

In a TDMA-based WSN nodes, these crystals are used to time-stamp packets by incrementing a counter internal to the micro-controller at each clock tick. An interrupt is triggered at the reception of a radio packet, and the current value of the counter (e.g. a 16-bit time stamp) is stored

in memory. The accuracy of the time-stamp depends on the frequency of the clock driving the counter. A 32kHz crystal, for example, is accurate to within $1/32768 \simeq 30us$. This is acceptable for most WSN applications, and offers an adequate trade-off between low-power operation and time-stamping accuracy. Digitally controlled, RC-type oscillators, which can be fabricated within a micro-controller, often oscillate at higher frequencies. As shown in the next section, such oscillators exhibit significantly higher drift, and are much more susceptible to fluctuations in temperature and voltage. This chapter will show, however, that it is possible to effectively synchronize a TDMA-based WSN using those low-end RC-based oscillators.

5.2 Preliminary Measurements

All of the methods presented in this chapter have been implemented on on the eZ430-RF2500 platform [45], a low-cost wireless node containing an MSP430 [46] 16-bit 16MHz micro-controller and a CC2500 [18] radio. This platform was chosen, in part, to showcase the ability to implement a TDMA-based WSN on low-end, limited-resource, out-of-the-box hardware, thus acknowledging that most platforms can provide the same capabilities. The micro-controller only has 1kB of RAM memory (10kB is typical) and 32kB of flash (64kB is typical). A digital bus allows the micro-controller to load packets into, and receive packets from the radio. The platform allows an interrupt to be enabled to signal the beginning of a received packet.

The MSP430 micro-controller features a 16-MHz Digitally Controlled Oscillator (DCO), implemented as a ring oscillator. It also contains a 12kHz Very Low-power Oscillator (VLO) [46]. The eZ430-RF2500 does not have a low-power, on-board 32KHz crystal. The CC2500 radio chip is driven by a 26MHz crystal oscillator. For comparison purposes, the radio was programmed to output a divided version of its crystal output. Using a Agilent 53131A Universal Counter, the inter-device frequency variation, as well as variation over a range of temperature and voltages were monitored for the DCO, VLO, and crystal[2]. In each case, the frequency variation was quantified in parts-per-million (ppm).

Fig. 25 shows the variation of output frequency for 11 different eZ430-RF2500 boards, at a steady ambient temperature of 25C, and input voltage of 3.6V. The crystal oscillator exhibited

[2]For most 32kHz crystals, these values are readily available from manufacturer data sheets.

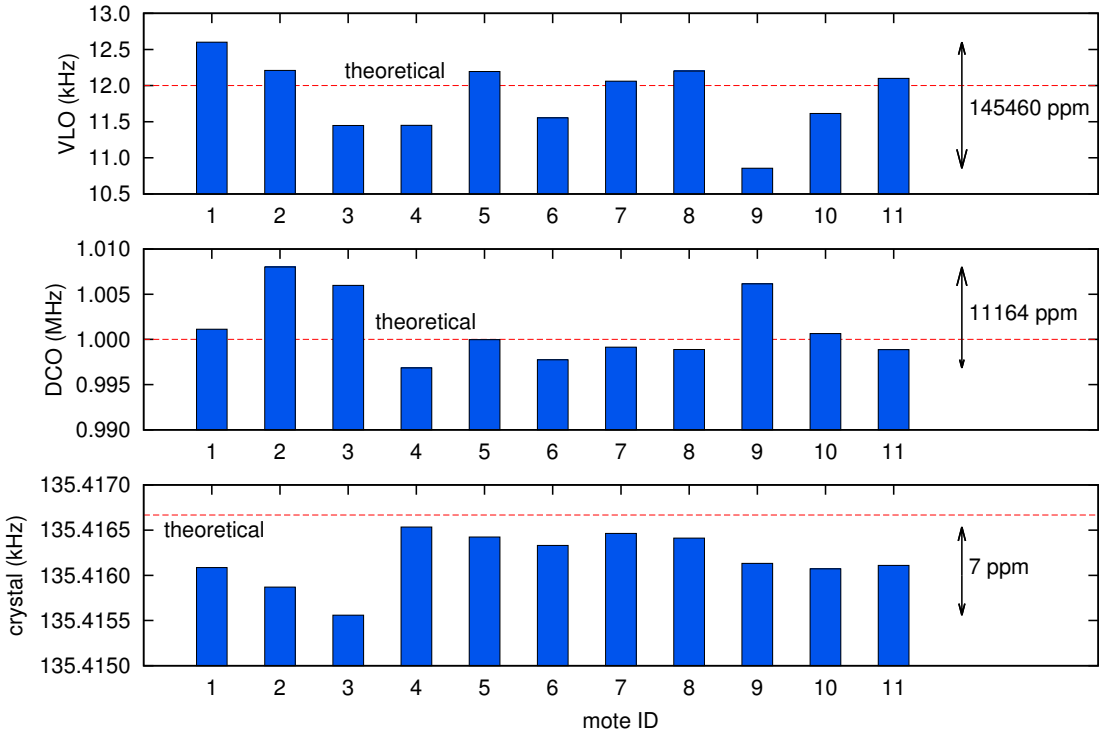

Figure 10: Variation of the clock frequencies over a set of different motes. Temperature was kept constant at 25C, voltage at 3.6V.

the smallest drift ($7ppm$), providing a timer accuracy well suited for conventional synchronization protocols. The VLO showed significant inter-device drift, at an average of $145000ppm$, with a frequencies varying between 11kHz and 12.5kHz. The DCO offered a 10-15 fold improvement over the VLO, but inter-device frequency still varied between 995kHz to 1005kHz.

Fig. 11 shows the effect of supply voltage on clock frequency, for a given eZ430-RF2500 board at a constant temperature. The crystal oscillator remained extremely stable over the range of supply voltage, with an average drift less than $1ppm$, while the VLO and DCO drifted by tens of thousands of ppm.

The third experiment investigated oscillator stability in regard to temperature fluctuations. An eZ430-RF2500 board was mounted on top of a heating/chilling plate, and the the temperature was varied from -5C to 50C, while the voltage was kept at a constant 3.6V. Similar to previous cases, Fig. 12 shows that the crystal remains much more stable relative to the VCO and DLO.

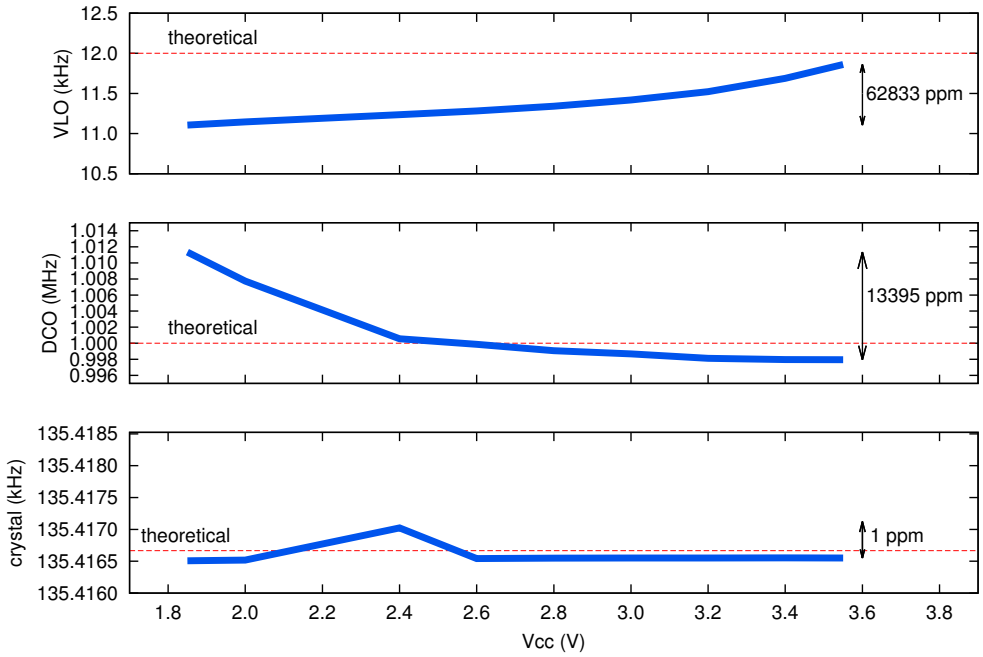

Figure 11: Variation of the clock frequencies over a range of supply voltages `Vcc`. Temperature is kept constant at 25C.

The above measurements further validate why crystals remain the dominant clock sources for WSN synchronization applications. While digitally controlled oscillators such as the DCO and the VLO of the MSP430 exhibit a drift that is orders of magnitude worse than a crystal oscillator, the following sections describe techniques to permit such clock-sources to be utilized for effective TDMA-based WSNs.

6 Problem Definition

The major challenge associated with utilizing inaccurate, high-drift clock sources for TDMA-based WSN synchronization relates to the instability of the clock's frequency (frequency drift). Crystal-free nodes are unaware of the relative speed at which their clock operates. For example, as seen in the previous section, it is entirely feasible that for one node's 1000 clock ticks might translate $9ms$, while for another node it might take $11ms$. Such specifications make the tight time structure required for conventional slotted communications infeasible. Thus, as depicted in Fig. 13 (a) and (b), if the frequencies of two nodes are not tuned correctly, only adapting the `slot phase` is

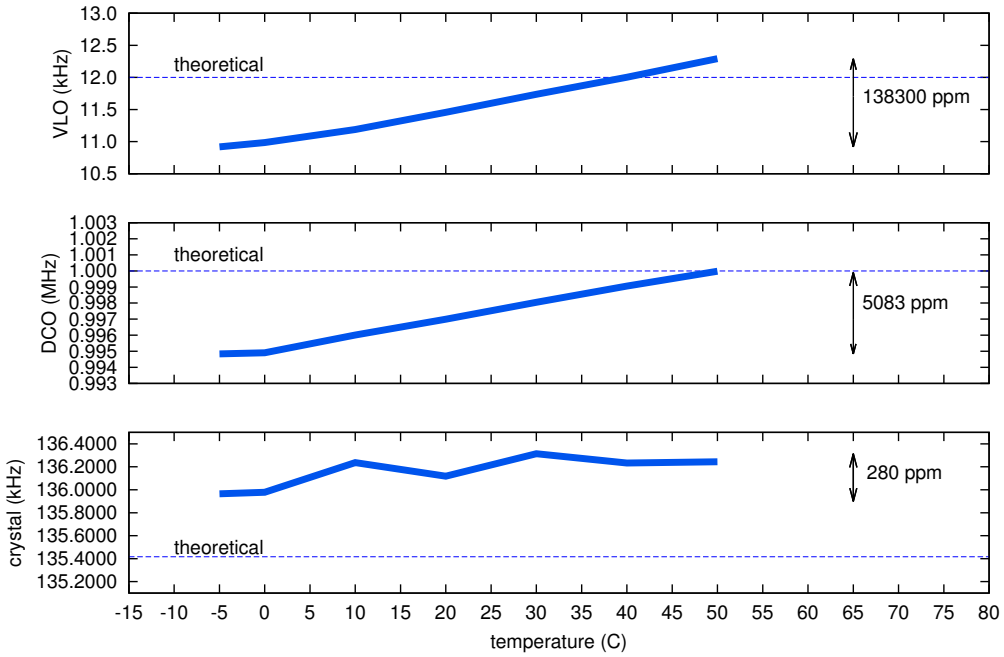

Figure 12: Variation of the clock frequency over a range of temperatures. Supply voltage is kept constant at 3.6V.

not sufficient. In our proposed `adaptive synchronization` approach, the nodes also adapt their `slot duration` to account for frequency drift of the their clocks (see Fig. 13 (c)).

7 Proposed Method

We assume the presence of a time master in the network, against whom other nodes synchronize (possibly using a structure as in Fig. 1). The network is assumed to run a slotted protocol, in our case equivalent to IEEE802.15.4E. This section describes the proposed `adaptive synchronization` procedure, along with multiple enhancements which can be further implemented to achieve more robust performance.

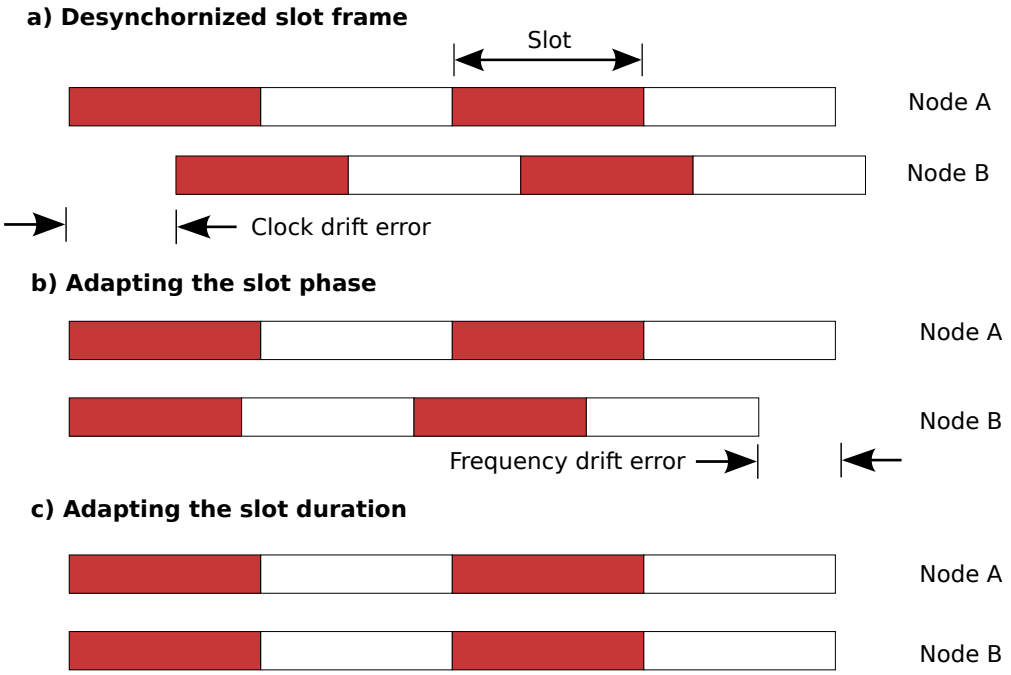

Figure 13: Unlike in conventional TDMA synchronization (see Fig. 5), `adaptive synchronization` involves changing both the slot phase (b) as well as the slot duration (c) to account for unstable clock sources.

7.1 Core Algorithm

`Adaptive synchronization` is built upon the TDMA-based architecture described in Section 2.3. The core of the algorithm is a controller which timestamps received packets and adjusts the `Slot Duration` and `TX Offset` to mitigate a node's clock-, and frequency-drift. A slot resembles that of Fig. 4. As detailed in Section 2.3, every packet is sent exactly `TX Offset` after the beginning of a slot. `Adaptive synchronization` timestamps the received packets to tune the following values:

- `TX Offset`, the duration between the start of the slot and the moment the packet leaves the radio, in clock ticks; this value is used to compute the synchronization error and to re-align a node's slot frame to a time-parent;
- `Slot Duration`, the duration of a single slot, in clock ticks; varying the `Slot Duration` changes the length of the entire `slot frame`, and thus is used to correct effects of frequency drift between two nodes (see Fig. 13 (b) and (c)).

Adapting the value of TX Offset follows the principle described in Section 2.3, i.e. using one-way synchronization 1 to re-align a nodes' slot frame to that of its time parent. If required, this synchronization error is then also used to tune the Slot Duration. As shown in Fig. 13(b), if a synchronizing node's Slot Duration is not correct, at the next synchronization opportunity, it adjusts its slot phase to match that of its parent. Adjusting the phase consists of either delaying or bringing forward the beginning of the current slot (effectively delaying or moving forward the entire slot frame).

Adaptive synchronization increments a counter by 1 each time phase is delayed, and decrements it by 1 when brought forward. During this process, it also keeps track of the the previous synchronization errors in an array $PreviousErrors$ (in the form of a circular buffer). Once the synchronization counter reaches a value of $sync_{thresh}$, the node calculates its Slot Duration error using 3.

$$error_{SlotDuration} = \frac{average(PreviousErrors)}{\texttt{Num of Slots in Slot Frame}} \tag{3}$$

3 is derived assuming a node communicates exactly once every slot frame with its time parent. This can, however, be generalized to compute the slot-duration error given any communications schedule. Once the error is computed, 3 is used to update the Slot Duration, and the $sync_{thresh}$ counter is then reset. Adjusting the Slot Duration also requires the adjustment of the TX Offset value. For ease of implementation, our specific instance of the algorithm defines a constant ratio between slotDuration and TX Offset. In our implementation, this ratio is 5. This means that, if a node decides to increase Slot Duration by 5, it has to also increase TX Offset by 1. A $10ms$ Slot Duration thus gives a TX Offset of $2ms$, which corresponds to the IEEE802.15.4E standard. This constant ratio is not required, and the algorithm can be configured to deal with an arbitrary TX Offset within a slot.

Our implementation defines the constant $sync_{thresh} = 20$. This value could potentially be parameterized via physically-based oscillator properties. For example, the Allan Deviation [47] of an oscillator could be used to evaluate its frequency stability and to quantify its noise properties. A node could then compare these values to a on-line computation of Allan Deviation to detect significant frequency drift, and could thus adjust the Slot Duration accordingly. Such

optimizations are, however, beyond the scope of this chapter.

7.2 Additional Techniques

The methods presented in the previous section facilitate the basics of adaptive synchronization. This section lists a number of additional techniques that can be used to increase the synchronization speed and lower node power consumption.

7.2.1 Tuning `TX Delay`

`TX Delay` is the duration between the moment the micro-controller tells the radio to send the packet, and the moment the first bytes of the packet leaves the radio. On the CC2500, this take a constant duration of $235us$ [18]. Nevertheless, when using low-power oscillators, a node can not know the frequency of its clock, and can thus not convert this duration to a number of clock ticks. `Adaptive synchronization` therefore timestamps, through an interrupt form the radio, when the first packet byte leaves the antenna. It then compares this value with `TX Offset` and adjusts `TX Delay` such that the edge rises exactly `TX Offset` clock ticks after the beginning of the slot.

7.2.2 Synchronization Bootstrapping

In accordance with the IEEE802.15.4E standard, when a node first attempts to join a network, it synchronizes by listening for `advertisement packets` from nearby neighbors. Timing information about these packets is then used in conjunction with the adaptive approach in Section 6 to synchronize to the network. This process can be relatively slow, especially when a node and its time parent are operating at significantly different frequencies, as in the case when using digitally controlled oscillator clocks.

In our proposed `bootstrapping` approach, when a node attempts to join the network, it waits until it hears `two` advertisements from the same time parent. Per the IEEE802.15.4E stan-

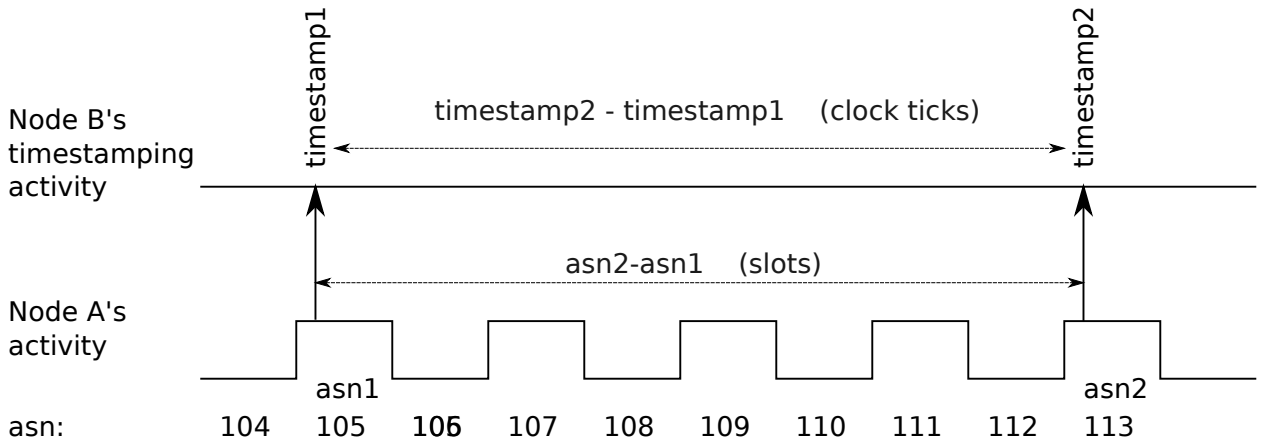

Figure 14: Bootstrapping involves measuring the number of clock ticks between the reception of two advertising packets to calibrate the initial `Slot Duration`.

dard, each transmitted advertisement contains a reference to the number of the absolute slot it was transmitted in. This reference is known as the `absolute slot number` (asn). It is a counter that serves the purpose of keeping track of all slots throughout the lifetime of the network, even if the overall slot frame repeats indefinitely. When the timestamps and asns of each advertisement packet are recorded, as depicted in Fig. 14, the node can derive a good estimate of its initial `Slot Duration` using 4.

$$SlotDuration_{initial} = \frac{timestamp2 - timestamp1}{(asn2 - asn1)} \tag{4}$$

7.2.3 Consecutive Advertisement Slots

A slight extension to the bootstrapping technique described in the previous section involves a time parent transmitting consecutive advertisement packets. A node joining the network hears immediate back-to-back advertisements, allowing it to calibrate its initial `Slot Duration` faster (using 4). Provided that a node hears both advertisements, this approach allows for near immediate calibration of its `Slot Duration`, requiring only the duration of two slots. The phase of its slot frame can then be aligned with that of the time parent, after which the node uses adaptive synchronization to stay synchronized. An implemented version of this modified schedule is shown in Fig. 15.

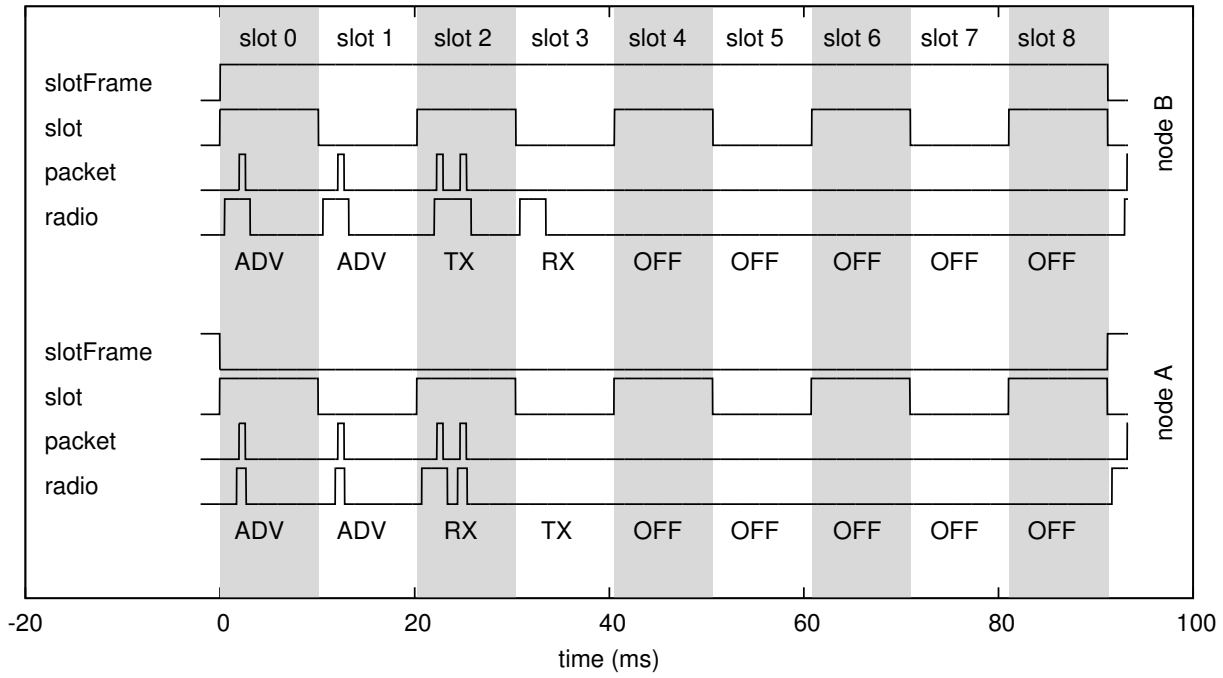

Figure 15: Oscilloscope recording of the fully implemented `adaptive synchronization` procedure. Node A is the time parent.

8 Implementation

`Adaptive synchronization` (including all the additional techniques described in Section 7.2) was implemented on the eZ430-RF2500 platform. The implementation uses the MSP430's DCO clock while providing a slot length of $10ms$. The source code[3], consisting of 973 lines of C code, was developed using IAR Embedded Workbench IDE 5.10.4. To link the micro-controller to the radio, we utilized SPI drivers available from Texas Instruments [4], and modified the readily available CC2500 drivers. The binary compile for the MSP430f2274 micro-controller has a memory footprint of 6532 bytes of flash memory and 383 bytes of RAM. Given the relatively low resources available on the eZ430-RF2500, our implementation still occupies only a fifth of the available flash and a third of the RAM, thus leaving further room for routing and applications.

Fig. 15 shows an oscilloscope-generated snapshot for a simple network of two synchronized nodes, which are running the full adaptive synchronization implementation. Node A represents the

[3]As an online addition to this chapter, the complete source code is available under the OpenBSD license on the first author's website.

[4]http://focus.ti.com/docs/toolsw/folders/print/simpliciti.html

time master. The activity of each of the node is indicated by four digital signals:

- `slotFrame` toggles when a new Slot Frame begins;
- `slot` toggles at each new slot. Each slots is $10ms$ long;
- `radio` is high whenever the radio chip is on, either transmitting or in listening mode;
- `packet` is connected to the interrupt line between the CC2500 and the MSP430. It is high whenever bytes are being sent of received by the radio. The low-to-high transition of this line is used to calculate synchronization error.

Fig. 15 furthermore indicates a sample schedule used by both nodes:

- Slots 0 and 1 are advertisement slots. In those slots, each node either transmits an advertisement packet, or listens. An advertisement contains enough information for a new node to join the network. In this case, node A transmits in both slots, while node B listens; This demonstrates the consecutive advertisement feature described in Section 7.2.3;
- Slot 2 is a dedicated slot for node B to send information to node A. The first pulse corresponds to the data message sent from B to A; the second pulse in the same slot is the acknowledgment sent from A to B;
- Slot 3 is a dedicated slot for node A to send information to node B. In this case, node A has no packet to transmit, thus keeping its radio off; node A listens for $2ms$, and switches off its radio as it received nothing;
- Slots 4 through 8 are OFF, the two nodes do not exchange any packets;

A full 10-node network was deployed in a large office setting over a span of 30 hours to evaluate the synchronization algorithm. This deployment also enabled the use of channel hopping, as described in IEEE802.15.4E. Fig. 16 shows a frequency spectrum analysis of 16 frequencies along the 2.4GHz band, indicating that the 10 node network effectively utilized all 16 channels for packet transmission. To our knowledge, this is the first such implementation of `time synchronized channel hopping` which does not rely on a crystal time source.

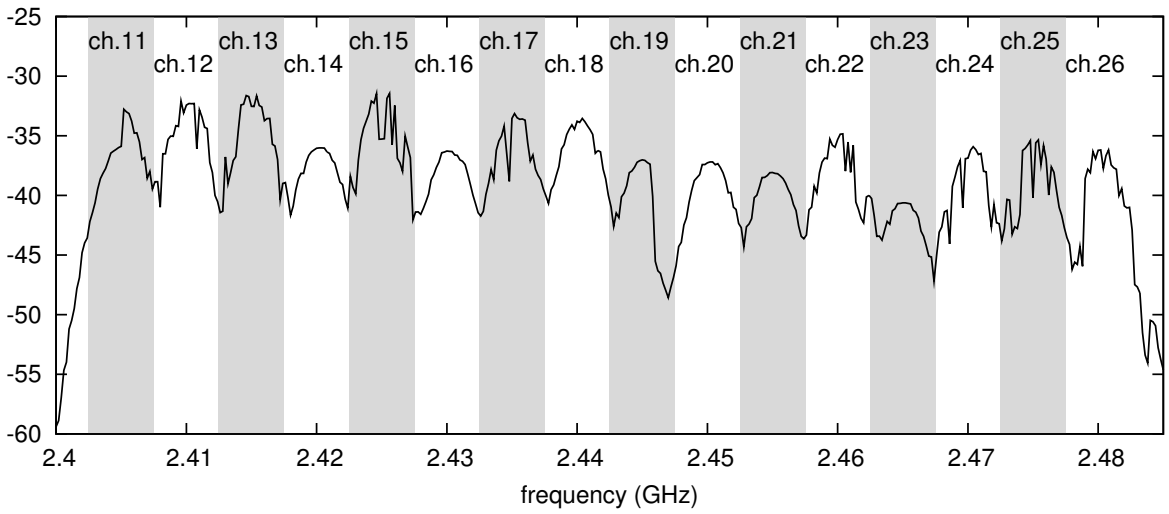

Figure 16: Frequency spectrum recording of a 10-node network running `adaptive synchronization`, showing the use 16 different channels for communication.

9 Experimental Results

9.1 Impact of Temperature

To demonstrate the adaptive nature of the proposed method, the synchronization error, `Slot Duration` and `TX Delay` were recorded, as a node synchronized to a time parent 17. At about $40s$ into the synchronization, an industrial hot-air gun was briefly used to heat up the synchronized node. In accordance with Fig. 12, this caused the node's internal RC-oscillator to resonate at a higher frequency, thus effectively reducing the duration of its slots. The algorithm responded by increasing the node's `Slot Duration`, thus successfully mitigating the frequency drift error induced by the fluctuation in temperature. The detailed rate-of-change of temperature was not recorded, but it can be seen that the algorithm adapted the `Slot Duration` within less than a second of the occurrence of the event, thus providing an initial evaluation of the rapid convergence properties of the proposed approach.

9.2 Long-term Deployment

Figure 18 displays the temporal behavior of the multi-node deployment described in section 8, plotting the average `Slot Duration` of nodes in the network over a 30 hour period. The figure also shows the spread (one standard deviation) of `Slot Duration` among the nodes. High frequency fluctuations are apparent in the mean, indicating that the nodes often adjusted their `Slot Duration` to mitigate frequency drift error. Furthermore, a downward trend can be seen over the 30 hour period, showing that the average network-wide `Slot Duration` was adaptively lowered to retain synchronization to the time parent. A potential cause of this phenomenon could be explained by an increase in the time parent's clock frequency. During the experiment, the time parent was physically connected to a computer for data-logging purposes. This could have potentially led to the heating up of the mote due to heat expelled by the computer, thus causing the time parent's clock frequency to increase in accordance with Fig. 12. Further experiments are required to validate this claim.

9.3 Synchronization Accuracy

Due to clock drift, nodes in a TDMA-based setting need to resynchronize regularly to reset their slot phase. The frequency at which this is required to occur is linearly proportional to the drift of the clock. For example, to retain a low radio duty cycle, the IEEE802.15.4E standard permits a node to be desynchronized by at most $1ms$. If clock has drift of $10ppm$, after $100s$ it will become desynchronized by $1ms$; in the case of slotted communications, resynchronization thus needs to occur at least every $100s$. Similarly, if a clock drifts at $100ppm$, resynchronization needs to occur at least every $10s$ to retain IEEE802.15.4E specified synchronization to the network.

A shown in previous sections, the very large clock drift, and frequency instability, exhibited by the MSP430's DCO, can effectively be compensated in software by `adaptive synchronization`. Even when compensated, the resulting system will still experience some clock drift. Evaluating this resulting drift can be accomplished by forcing the motes to resynchronize at a specific period. Clock drift can then be measured for a number of such periods to characterize the drift behavior of the proposed approach. In our experiment, this was achieved by varying the length of the

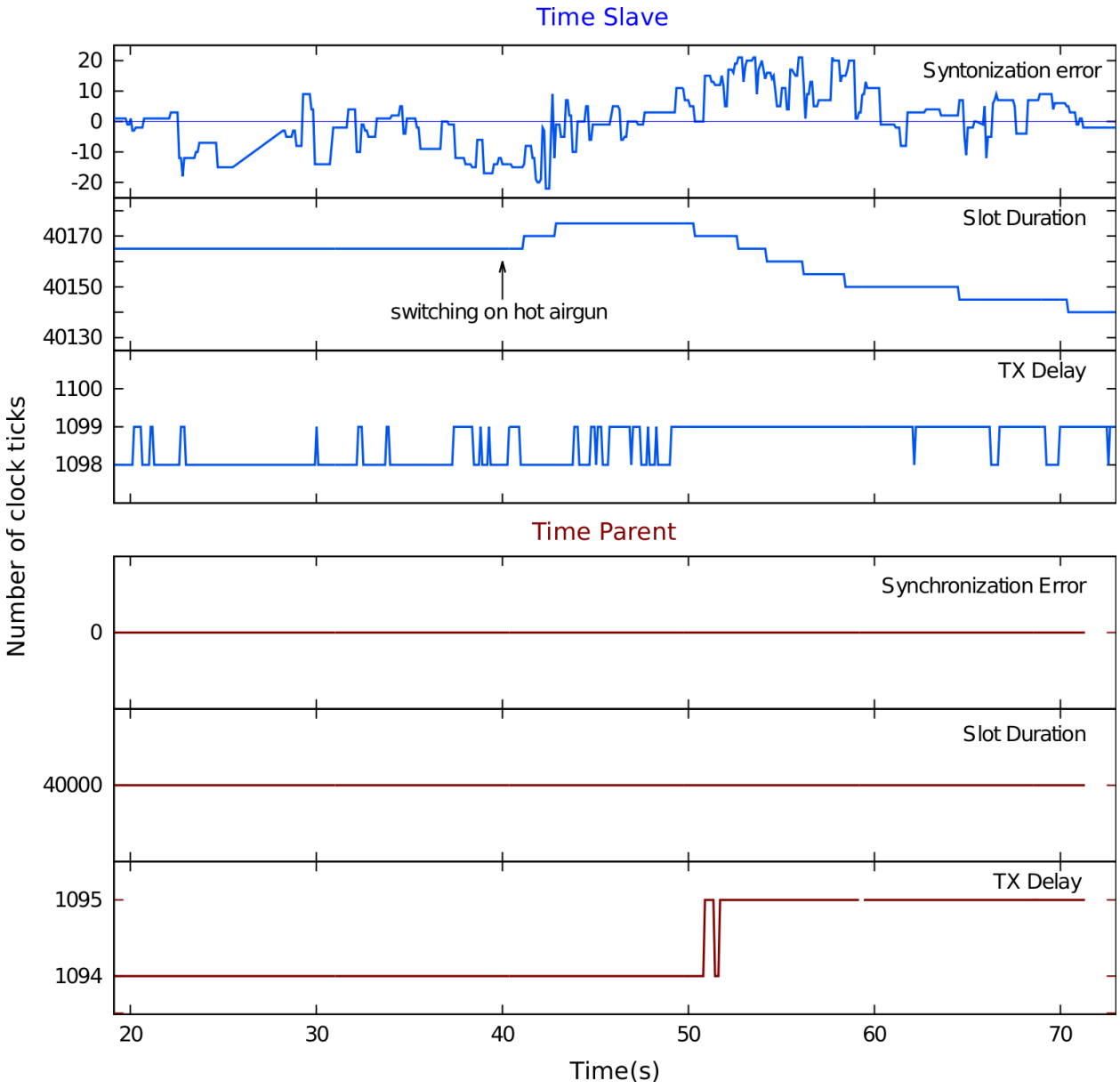

Figure 17: Recorded values of synchronization parameters of adaptive channel hopping while a node is being subjected to temperature fluctuations. The synchronized node reacts by adapting its `slotDuration` accordingly. The time parent is not synchronizing to another node, and thus does not compute a synchronization error, or update its `Slot Duration`.

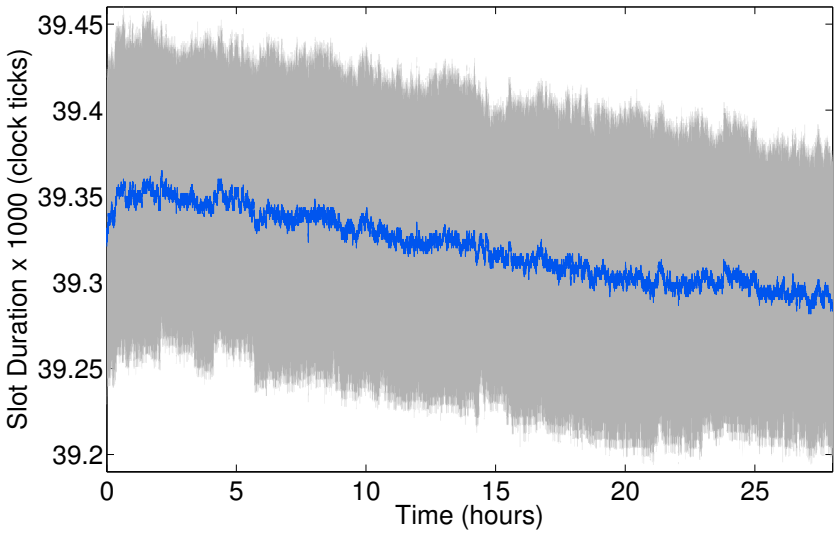

Figure 18: Average `Slot Duration` for a multi-node network over time. The gray area reflects one standard deviation around the mean.

slot frame and forcing motes to synchronize once per slot frame. An Agilent 53131A Universal Counter was used to evaluate the relative phase between the start of consecutive slot frames. The minimum, maximum, and mean of the phase offset were recorded to evaluate the effects of the resynchronization period (see Fig. 19). The results were also compared to the drift of a theoretical $100ppm$ crystal oscillator.

Fig. 19 shows that the software compensation offered by `adaptive synchronization` mitigated the large drift characteristics, and frequency instabilities of the MSP430 DCO, effectively reducing the overall drift to the equivalent of a $100ppm$ crystal. In our particular implementation, nodes utilizing `adaptive synchronization` should resynchronize at least every $10s$ to ensure meeting the $1m$ synchronization error permitted by IEEE802.15.4E. Depending on the bandwidth requirements of a given application, it is reasonable to assume that the majority of this resynchronization can be accomplished through the time-stamping of data packets, which need to be transmitted, in any case, as part of general WSN traffic. Results obtained from data in Fig. 15, indicate that the process of resynchronizing consumes $810\mu s$ of radio-on time at the transmitter, $2.55ms$ at the receiver. For the ex430-RF2500 platform, keeping two nodes synchronized thus translates into a overhead duty cycle of 0.0168%, assuming a $10s$ synchronization period.

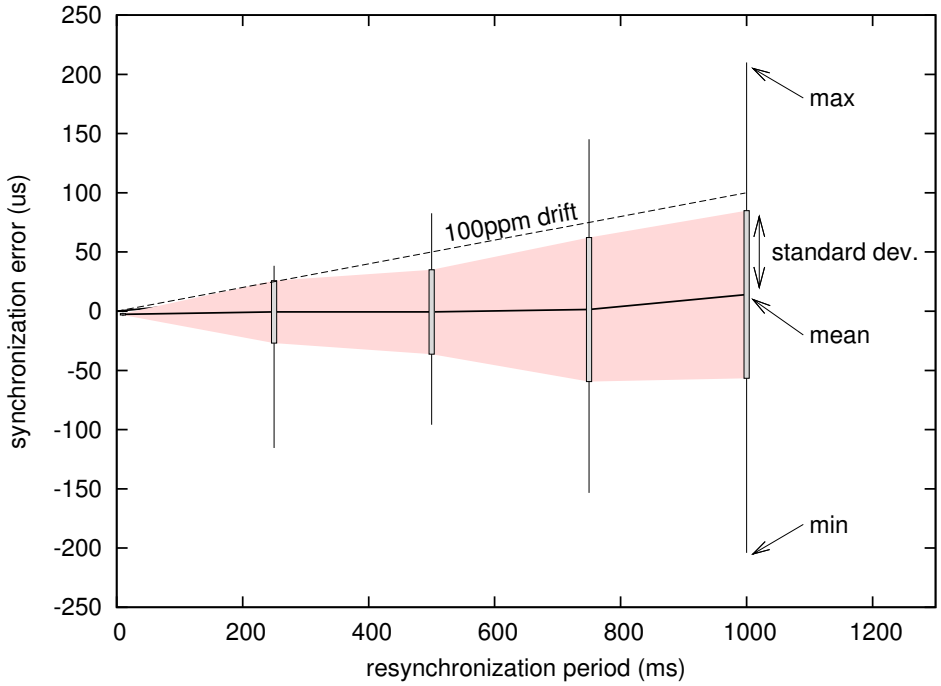

Figure 19: Synchronization error as a function of the resynchronization period (length of the slot frame). 68% of the measurements fall with the shaded area.

10 Conclusions

This chapter demonstrated that time slotted commutations and frequency channel hopping can be implemented on low-end WSN nodes without the use of accurate crystal clock sources. We presented a system-level solution to facilitate network-wide synchronization even when high-drift digital oscillators are used as time sources. Combined with a crystal-free radio [44], currently available fabrication methods could then be used to fabricate an entire mote onto a single silicon die, thus facilitating the greater vision of the millimeter-cubed SoC WSN node. Our solution was implemented on the ez430-RF 2500 platform to motivate the ability to implement TDMA-based crystal-free communications on many other platforms. Superior oscillators, and WSN hardware are currently available, and thus have the potential perform even better when compared to our example implementation.

While our motivation was driven by a SoC mote solution, the implications of this method for power consumption could be significant. It is reasonable to suggest that in a large number of TDMA-based WSNs, the crystal time-source presents a major source of power consumption. To

ensure synchronization, these crystals are fully powered throughout the entire lifetime of the deployment. Thus, coupled with our proposed method, the use of a sub-nW oscillator [43], when compared to today's major low-power crystals, could have the potential to reduce the amount of power required for time-keeping by at least an order of magnitude. Future work should explore the implementation of such oscillators for WSN synchronization. Furthermore, the detailed convergence properties of our approach over a broader set of experiments should be analyzed. The ability to use physical oscillator properties, such as the Allan deviation [47], should also be investigated to guide a more robust detection of oscillator frequency drift. This chapter was the result of a paper, written with Thomas Watteyne, Steven Glaser, and Kris Pister.

Part III

Feasibility Analysis of Controller Design for Adaptive Channel Hopping

As discussed in the introduction, communication reliability in Wireless Sensor Networks (WSNs) is challenged by narrow-band interference and persistent multi-channel fading. While frequency-agile communication protocols have been designed and standardized to increase reliability, these protocols do not adapt the set of channels they hop on to the environment. Rather, single channels are selected, or an *a priori* hopping sequence is established. In this chapter, we evaluate the efficacy of a controller which continuously samples all available frequency channels in order to operate on a channel which performs reasonably well. We wills show that the overall average link Packet Delivery Ratio when using this controller reaches 99.4%, and is higher compared to a single channel solution, on any channel. We will also evaluate the efficiency of this approach by simulating its behavior on connectivity traces gathered during a real-world deployment. This data set is dense in time and sufficiently large in number of nodes and time to be statistically valid.

One approach is to select the channel on which the loss of transmitted packets is minimized; yet, the best channel may vary over time. The option advocated in this chapter is to change channel on a link-by-link basis when necessary, a concept known as **adaptive channel hopping**. In adaptive channel hopping, a mote transmits data on a channel which is predetermined to work acceptably well. Rather than persistently sending information on this stable channel, the pair of communicating motes transmit packets on different channels every now and then. These intermittent transmissions allow for the goodness of other channels to be estimated. When the current channel starts performing unacceptably bad, the mote pair switches to the channel which features the best goodness estimation.

As detailed in Section 3.3, channel hopping has received increased attention in industrial and standardization bodies, driven by the quest for reliability. To the best of our knowledge, however, no work has quantified the performance of adaptive channel hopping. Does adaptive channel

hopping offer an increase in performance over a single-channel solution? Over a blind channel hopping approach? If yes, how often should a pair of nodes evaluate the goodness of the different channels? Is it feasible to design a controller for adaptive channel hopping which is simple enough to be implemented on current hardware? This chapter intendedd to cover channel hopping from a ”best case” perspective, where the best known channel is always known and utilized. This will provide us with an upper bound that can be used when motivating real-world implementations.

Choosing the methodology for answering those questions is non-trivial. On one hand, simulated propagation models do not capture well complex phenomena such as persistent multi-path fading, or the impact of a dynamic environment. On the other hand, a purely experimental study does not provide the ability to evaluate different algorithms under the exact same conditions, as connectivity is time-varying in nature. This chapter proposes to *replay* communication algorithms on *connectivity traces* gathered from a real-world deployment. We believe that this novel approach elegantly copes with the issues raised above, and that it will become commonplace for the performance evaluation of complex environments such as channel hopping protocols.

The contributions of this chapter can be summarized as follows:

- We show that replaying communication algorithms on real-world traces allows for quick and realistic performance evaluations;
- We demonstrate how adaptive channel hopping increases the average link Packet Delivery Ratio $\mathcal{P}$ compared a single channel solution operating on any channel;
- We evaluate the impact of the choice of the channel in a single channel solution, and the impact of the probing period when using adaptive channel hopping.

Evaluating the efficiency of communication algorithms by replaying their behavior on real-world connectivity traces is, we believe, an approach particularly suited for complex systems such as channel hopping Medium Access Control (MAC) protocols. Section 3.3 presented a comprehensive overview of the latest academic and standardization efforts in frequency-agile communication, as well as an overview of the connectivity traces used in this chapter. Section 11 describes the simple controller for adaptive channel hopping evaluated in this chapter, its complexity and limitations.

In Section 17, we present the results obtained by simulating the controller's behavior on real-world connectivity traces; comparison is offered with single channel solutions. Section 18 underlines the research directions stemming from this chapter and discusses the choice of parameters.

Fig. 8, shown at the beginning of this thesis, illustrates the challenges faced by adaptive channel hopping. Overall, every channel undergoes deep fading periods, rendering single channel solutions inadequate[5]. The goal of adaptive channel hopping is to assess the Packet Delivery Ratio $\mathcal{P}$ for all channels, and to use this information to always communicate on a reasonably good channel. Let's assume, for link $24 \rightarrow 17$, that communication starts on channel 15. After a few hours, channel 15 features a low $\mathcal{P}$, and communication can switch for example to channel 26. After day 6, channel 26 experiences poor performances, and the controller may switch to a different channel.

11 Adaptive Channel Hopping

We call **controller** the entity which decides upon the channel to be used.

11.1 Controller Operation

Each node implements an instance of the controller for each link it participates in. We refer to a *slot* as a $15min$ time window[6]. Communication initiates on a randomly chosen channel c_i. Every k slots, the controller switches communication to channel $c_j \neq c_i$ for one slot. By counting the number of attempted and successful transmissions during that slot, the controller assesses the Packet Delivery Ratio $\mathcal{P}$ of channel c_j. We call **probing** the action of switching to a different channel for one slot to assess $\mathcal{P}$. After the probing slot, communication is resumed on channel c_i. k slots later, the controller probes channel $c_k = (c_j + 1)\ \%\mathcal{C}$ where $\%$ is the modulo operator and $\mathcal{C}$ the number of channel ($\mathcal{C} = 16$ for an IEEE802.15.4-compliant radio chip). Each channel is hence probed every $\mathcal{C} \cdot k$ slots.

[5]Strictly speaking, channel 17 on link $24 \rightarrow 17$ features a constant high $\mathcal{P}$. Nevertheless, a single channel MAC protocol assigns the same channel to all links, and, as shown in Section 17, there is no channel which works well at all times, for all links.

[6]This value is somewhat arbitrarily chosen as it is the time interval at which the dataset provides updates on the channels' Packet Deliver Ratio $\mathcal{P}$.

The controller keeps track of the Packet Delivery Ratio $\mathcal{P}$ of the current channel c_i. Whenever $\mathcal{P}$ falls below a threshold $\mathcal{P}_{thres}$, the controller declares the performance of this channel to be unacceptably bad, and switches to the other channel which features the largest $\mathcal{P}$. We call **switching** the action of the controller to change the current channel resulting from a low $\mathcal{P}$.

11.2 Optimality

We employ a controller because $\mathcal{P}$ is an inherently dynamic and unpredictable metric. Since the nature of the proposed control scheme relies on the sporadic sampling of other channels, the approach tends to be suboptimal in the case for which one specific channel is the best option at all times. Additionally, optimality is treated in an informal fashion, realizing that optimal control of a system such as a WSN depends often on mathematical gurantees [48]. Since solutions control theoretic equations in real-time do not appear to be feasible given the computational limitations imposed by the mote hardware, we employ the simplified control scheme presented above.

11.3 Packet Delivery Ratio $\mathcal{P}$ Estimation

The Packet Delivery Ratio $\mathcal{P}$ estimate of a channel is updated using 5, with $\mathcal{P}_{new}$ the updated estimate for a particular channel, $\mathcal{P}_{old}$ is the old estimate, and $\mathcal{P}_{measured}$ the estimate resulting from the probing.

$$\mathcal{P}_{new} = \alpha\mathcal{P}_{old} + (1-\alpha)\mathcal{P}_{measured} \tag{5}$$

The constant $\alpha \in [0, 1]$ is introduced as a weighting factor. A large value of alpha implies that the average Packet Delivery Ratio over time is more important than the one inferred from the last reading.

12 Results

We evaluate the efficiency of the controller described in Section 11 onto the dataset presented in [14][7]. The controller initially picks a channel at random, and is allowed to evolve according to the dynamics described in Section 11.

12.1 Parameters Used

Results are obtained for $\alpha = 0.2$. This puts more weight on the new Packet Delivery Ratio estimate, as Fig. 8 indicates that $\mathcal{P}$ tends to change abruptly. The choice of α depends on how fast a link transitions from high to low $\mathcal{P}$.

We set $\mathcal{P}_{thresh} = 0.9$. $\mathcal{P}_{thresh}$ influences the link hysteresis: a lower $\mathcal{P}_{thresh}$ value leads to less less frequent switching, at the cost of communicating longer on a sub-optimal channel.

k indicates the probing period in slots. The larger the value of k, the less frequently probing occurs, and the less up-to-date the estimate of $\mathcal{P}$ for each channel are. Section 12.3 evaluates the impact of k.

12.2 Witnessing Adaptive Channel Hopping

Fig. 20 depicts the channel chosen by the controller on link $24 \rightarrow 17$. This plot features two components, distinguished by line thickness. The thin line shows the polling, i.e. the periodic evaluation of the different channels' Packet Delivery Ratio $\mathcal{P}$. Polling happens every $k = 20$ slots, or 5 hours. Polling is used to maintain statistics on the goodness of each channel. When the current channel becomes unacceptably bad ($\mathcal{P} < \mathcal{P}_{thresh}$), the information gathered during polling is used to select the best channel among all other channels. Switching is then used to communicate on a different channel. Fig. 20 shows that 9 switches occured on link $24 \rightarrow 17$.

[7] As an online addition to this chapter, the dataset is made available at `http://wsn.eecs.berkeley.edu/connectivity/`.

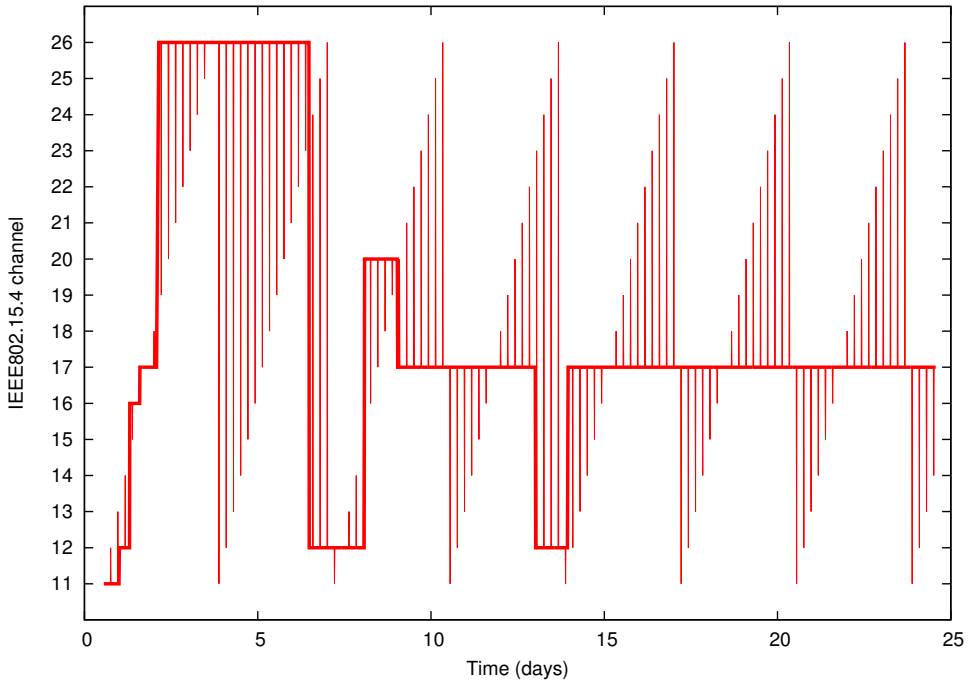

Figure 20: Polling – thin line – and switching – thick line – on link $24 \rightarrow 17$ (colored black in Fig. 6).

12.3 Equivalent Packet Delivery Ratio $\bar{\mathcal{P}}$

The di-graph depicted in Fig. 6 contains 62 links. Over the course of the 28 days of the deployment, a link switches channels multiple times as the Packet Delivery Ratio of the current channel drops below $\mathcal{P}_{thres}$. We refer to the Equivalent Packet Delivery Ratio $\bar{\mathcal{P}}$ the Packet Delivery Ratio averaged over all links and over the 28 days of the deployment. $\bar{\mathcal{P}}$ indicates what fraction of packets one may expect to be sent successfully when choosing a random time and a random link. We determine $\bar{\mathcal{P}}$ for single channel and Adaptive Channel Hopping. A higher value of $\bar{\mathcal{P}}$ indicates a "better" channel.

Fig. 21 shows $\bar{\mathcal{P}}$ for single channel operation (left) and Adaptive Channel Hopping (right). As an example, when $k = 20$, Adaptive Channel Hopping features a $\bar{\mathcal{P}} = 99.4\%$. Note that Adaptive Channel Hopping performs better than a single channel solution, on any channel.

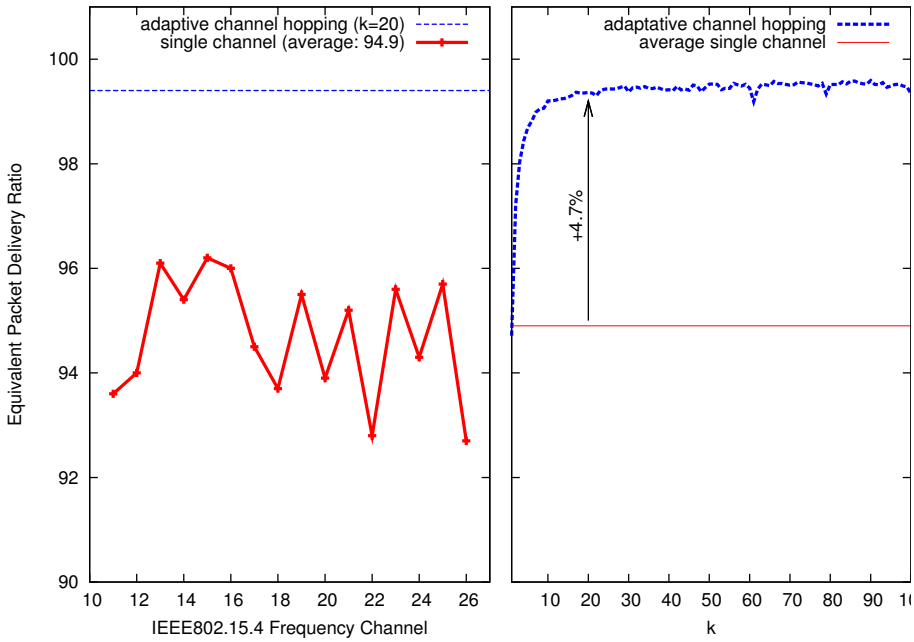

Figure 21: Equivalent Packet Delivery Ratio $\bar{\mathcal{P}}$ for a single channel operation (left) and adaptive channel hopping (right). Adaptive channel hopping performs better than single channel operation, on any channel.

13 Discussion

The controller designed in this chapter is simple enough to be implemented on current hardware. It requires each node to maintain statistics for each channel and for each link it participates in. Considering a node which participates in 5 links with 16 available channels, if each statistic is an integer between 0 and 255, the controller requires 320 bytes of RAM memory. This is an acceptable overhead for a micro-controller which typically features 10kB of RAM [49]. We are currently working on implementing Adaptive Channel Hopping in an open-source version of TSCH [26].

The results presented in Section 17 indicate that Adaptive Channel Hopping performs better than using a single channel. These results are obtained over a set of data sufficiently large in number of nodes (44 nodes) and time (28 days) to be statistically valid. However, connectivity traces were gathered in a printing facility in which there is no external interference from technologies operating on the same frequency band (WiFi, Bluetooth). As a result, channels are stable for hours or days.

The results presented in this chapter may not hold for very dynamic channels. This is the case when there is external interference (e.g. cause by a nearby 802.11 – WiFi – network) which causes *microscopic* dynamics and channels which are stable for only $500ms$ [50]. [4] shows that, in this case, blind channel hopping can be used. Blind Channel Hopping is implemented in TSMP [12], ISA100a [21], IEEE802.15.4E [25] and WirelessHART [20] and causes a link to switch channels every tens of ms, hopping evenly on all available channels.

A real-world deployment most likely features both macroscopic and microscopic dynamics. An issue worthwhile investigating is whether blind channel hopping can be coupled with adaptive channel hopping. This could be done through the use of whitelists. In such a system, the controller described in this chapter could be used to identify the n best channels (the whitelist) over which a blind channel hopping protocol operates.

14 Conlusions

This chapter advocates the use of Adaptive Channel Hopping, in which nodes continuously evaluate the goodness of all available frequency channels in order to operate on a channel which performs reasonably well. We show how the use of a simple controller yields performances which are higher than a single channel solution operating on any channel. This controller is simple enough to be implemented on current hardware, and deals efficiently with slow-varying channels.

Results are obtained by replaying the behavior of the controller on traces gathered from a real-world deployment. The data set is sufficiently large in number of nodes (44 nodes) and time (28 days) to be statistically valid, and is dense in time. For such a *replay* approach to be widely adopted, a larger set of traces is needed, with traces collected in different environments. This chapter was the result of a paper, written with Thomas Watteyne, Steven Glaser, and Kris Pister.

Part IV

Comparison and implementation of channel hopping algorithms

The previous chapter showed that further reliability within TDMA-based WSNs can be improved significantly by using a channel hopping controller that continuously monitors the channel space to persistently operate on the best possible channel. While this improves overall PDR, it does rely on a single channel for extended periods of time. The implementation of an ”always best” channel hopping solution may not be feasible given sudden rapid connectivity drops, and may not be exactly optimal, since there may be multiple channels that have 100% PDR at any time. This chapter investigates how a list of multiple best channels can be developed. We will evaluate the efficacy of two channel hopping algorithms. The first is based on the IEEE802.15.4e blind channel hopping standard. The second algorithm follows a similar hopping pattern, but chooses transmission channels based on a dynamically changing whitelist. It will be shown that the latter reduces network radio on-time by 2.5 percent compared to the blind hopping approach.

The contributions of this chapter can be summarized as follows:

- We introduce a channel hopping algorithm which dynamically adjusts its whitelist to ensure optimal communication
- We evaluate the presented algorithms on an extensive data set of traces collected in a real-world setting
- We show that time synchronized channel hopping can be implemented on readily available hardware
- We study the channels selected by these algorithms, and draw conclusions regarding observed behavior

Section 3.3 presented an overview of the current efforts centered around frequency-agile communication. Section 15 describes a number of channel hopping algorithms. Section 16 describes implementation of these algorithm on the MSP430-RF2500 platform. In section 17, we study the behavior of these algorithms, by gathering connectivity traces over a period of 24 hours. A discussion regarding these results is offered in section 18.

15 Channel Hopping Algorithms

In this section we present two channel hopping algorithms. As described in the introduction, a proper implementation of these algorithms hinges upon a reliable synchronization approach, slotted in this case (see figure 4). The algorithms below assume that a schedule has been assigned for communication between two nodes. As such, the sole purpose of the algorithm is to provide a corresponding frequency for each scheduled transmission. As stated in the introduction ASN is the absolute slot number in a slotted communication scheme.

15.0.1 Algorithm 1: Blind Channel Hopping

This algorithm is a variant of the TSMP method described previously, and is further described by the **IEEE802.15.4E** workgroup. For this algorithm, the network controller schedules ASNs at which communication between two nodes should take place. The controller also initially provides a channel offset $ch0$, on which the communication should take place. Assuming 16 available channels, the next transmission takes place on:

$$channel_{next} = (ch_0 + ASN)\%16 \tag{6}$$

where % is the modulo operator. This technique ensures that all 16 available channels will be used equally, but that consecutive transmissions will take place on different channels. Blind channel hopping is given by algorithm 1.

Algorithm 1 Blind Channel Hopping.

1: initialize ch0
2: **for** each slot $j \in frame$ **do**
3: $channel = (asn + ch0)\%16$
4: Transmit Packet
5: **end for**

15.0.2 Algorithm 2: Adaptive Blind Channel Hopping

This dynamic white-listing algorithm incorporates algorithm 1, where the hopping sequence is determined according to 6, but furthermore collects link-specific statistics to restrict transmission only to a subset of channels. PDR statistics are recorded, and updated continuously for each channel, and a ranked list of best channels is constructed. During a scheduled transmission slot, the transmitting node uses this list to either send a packed on a channel determined by 6, or not transmit a packet at all. The receiving node still listens for incoming packets, but in cases where the transmitting node is aware of an unreliable channel, transmission power is conserved. A simple solution would involve utilizing a fixed-size whitelist, by only transmitting packets on the best few channels, for example. This has the unintended effect of wasting potentially good channels (as well as wasting power listening for incoming packets that never arrive), if the whitelist is too small. Similarly, a whitelist which is too large can include unreliable channels, leading to dropped transmissions. We thus seek to find a dynamic means by which to augment the whitelist, adding and removing channels to ensure that radio up-time is reduced. Radio **up-time** is the time the radio is kept on, transmitting, or listening to incoming packets. Reducing radio up-time reduces energy consumption.

To further understand the effect of algorithm 2 in regard to power consumption, the IEEE80215.4e standard can be used to isolate three general sources of radio up-time:

- The transmitting node does not transmit a packet, and the only energy consumed is by the receiving node, which is listening for the incoming packet at the beginning of a scheduled slot. We call this up-time T_{rx};
- The channel on the scheduled slot is in the whitelist, and the transmitting node sends a packet. We call this up-time T_{tx};

- A successful transmission culminates in an acknowledgement (ACK). This entails the receiver sending an ACK packet to the transmitter, who keeps the radio on to listen to the incoming ACK. We combine this transmission and reception, and refer to the corresponding up-time as T_{ack};

Thus, a successful transmission requires an amount of radio up-time equivalent to the sum of T_{tx}, T_{rx}, and T_{ack}. The amounts for these values will differ depending on hardware and implementation details, but it is possible to derive a realistic values of these numbers using the IEEE IEEE80215.4e standard:

$$T_{tx} = 4.25ms \;\; T_{rx} = 5.06 \;\; T_{ack} = 5.60ms \tag{7}$$

Given periodically gathered PDR statistics on each channel, these values can then be used to determine the optimal size of a whitelist. Let PDR_{avg} denote the average PDR of all channels in a whitelist. On average, a transmitting node will require $\frac{1}{PDR_{avg}}$ transmissions to successfully send a packet to the receiver. Given the size of the whitelist, a receiver will not always hear a packet on every scheduled slot. Thus, on average, a receiver will have to listen $\frac{1}{PDR_{avg}} \times \frac{16}{|whitelist|}$ number of times to receive the packet, where $|whitelist|$ denotes the length of the whitelist and 16 is the total number of available channels. Additionally, one ACK is expected during this period as well.

We introduce the cost function $\mathcal{J}$ to optimally, and dynamically adjust a transmitter's whitelist, such that overall radio up-time is minimized (or, energy is conserved). Assuming that channels are added to the whitelist according to PDR, and sorted starting with the highest, the optimal size of a transmitter's whitelist is given by maximizing the following const function:

$$\mathcal{J} = \frac{1}{PDR_{avg}} \left(T_{tx} + \left(\frac{16}{|whitelist|} \right) T_{rx} \right) + T_{ack} \tag{8}$$

$\mathcal{J}$ can also be viewed as the expected amount of radio time that will be required to successfully transmit a packet. The algorithm below ensures that the transmitter only sends packets if the channel on the scheduled slot is in the whitelist. Additionally, to ensure continuous aggregation of PDR statistics packets are transmitted periodically over channels which are not in the whitelist.

Algorithm 2 Adaptive Blind Channel Hopping. Does not transmit unless channel is in whitelist. Once every $framesPerStat$ number of frames, a packet is transmitted to update PDR statistics, even if a channel is not on the whitelist. $framesPerStat$ should be selected to ensure the number of channels is mutually prime to $framesPerStat \times |frame|$

initialize ch0
framesPerStat
initialize whiteList
for each slot $j \in frame$ **do**
  $channel = (asn + ch0)\%16$
  if $channel \in whiteList$
  or $asn\%(|frame| \times framesPerStat) == j$ **then**
    Transmit Packet and wait for ACK
    Update PDR values
    Minimize $\mathcal{J}$ and adjust whitelist
  end if
end for

16 Implementation

A playback simulation was carried out on an extensive data set collected in an in-door setting (see figure 22). The data set contained PDR values at hourly intervals for all links in the network. This permitted for the implementation of the previously mentioned algorithms. The cost function (2) was evaluated at each time step for each algorithm. A best- and worst- case cost was also evaluated for comparison. This served as a baseline set of values to which the efficacy of the channel hopping algorithms could be compared.

The channel hopping algorithms were implemented on the MSP430-RF2500 platform, a readily available and inexpensive system-on-chip that incorporates the MSP430 micro-controller, and a low-power CC2500 radio. The assembly does not ship with an on-board crystal. As such, a crystal-free time synchronization method was used to synchronize the devices. More information on this method is offered in a previous chapter. Communication was scheduled on a three-slot frame, each slot having a 10ms duration. The first slot was used for synchronization purposes, while slots two and three were reserved for directional communication between the motes. Per-channel PDR statistics were gathered on the motes, and written to a UART interface.

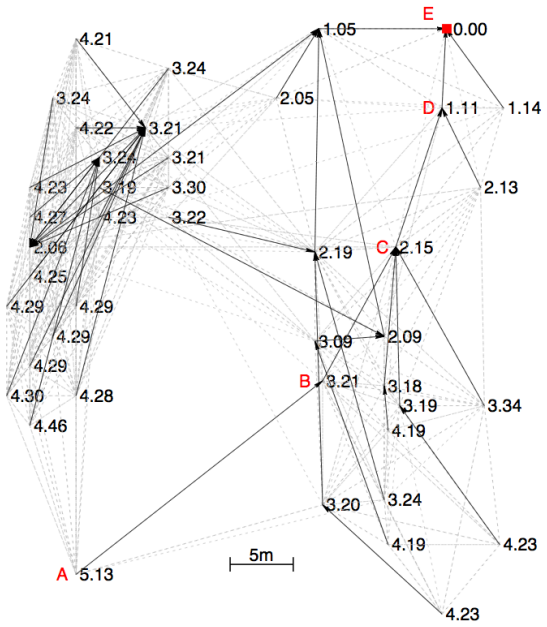

Figure 22: Indoor deployment which served as the data-set for evaluating channel hoping algorithms.

17 Results

Evaluation of this algorithm was conducted on the detailed Soda-Hall trace data set gathered by David Culler's group at UCB. The connectivity graph of the network is shown in figure 22. The results of playing back both algorithms on the trace data set are summarized in table 17. The table shows the average values of the cost function in (8) over the lifetime of the data set, evaluated at every time step for each link in the network. It shows the average amount of radio up-time that can be expected for the transmission of a single packet in the network. The upper bound for radio-up time was calculated assuming the worst possible channel selection at each time frame (e.g. choosing to transmit on the channel with lowest PDR at every step). Similarly, the lower bound was computed assuming transmission occurred at all times only on the channel with highest PDR. This lower bound sets a limit to the reduction in radio up-time that can be achieved by any given channel selection method for the particular data set. The table shows that algorithm 2 reduced average radio up-time by 2.5%, compared to the blind hopping approach of algorithm 1. There appears to be a margin of about 12% that can be gained by utilizing an even more efficient controller.

Table 1: Expected per-packet radio uptime

Channel Selection Method	Mean (ms)	Gain over Blind (%)
Upper Bound (worst channel)	215.6	-550
Lower Bound (best channel)	28.4	14.4
Blind Hopping	33.18	-
Adaptive Blind	32.33	2.56

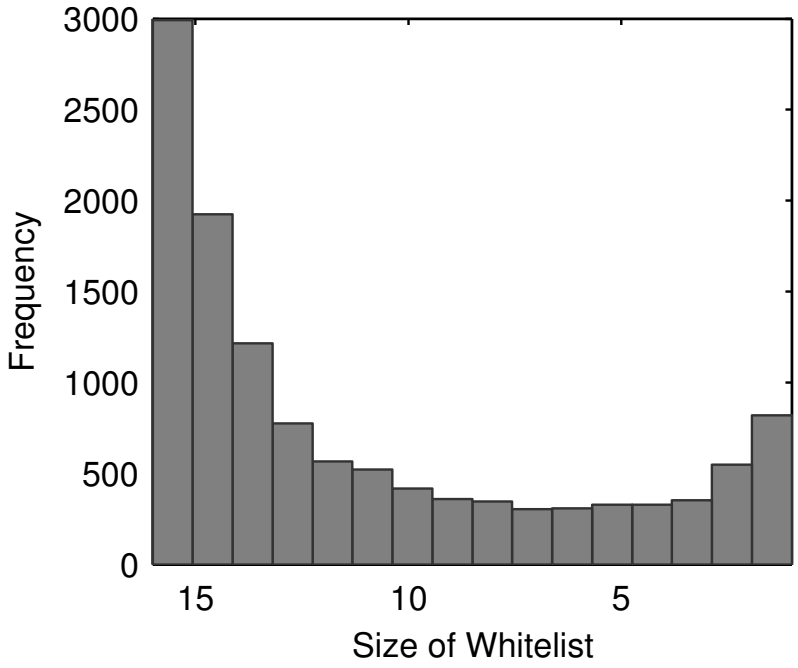

Figure 23: A histogram of whitelist sizes selected by algorithm 2 during the playback simulation.

Figure 17 shows a histogram of whitelist sizes, obtained by algorithm 2 over the lifetime of the data set. A large portion of whitelist sizes contained all 16 of the available channels. There were, however, still a significant amount of whitelists which relied only on a few key channels for transmission.

Figure 23 shows a histogram of average improvements for each time slot that are provided by algorithm 2 over algorithm 1. Algorithm 2 was never outperformed by the blind the channel hopping method, and although in the majority of cases algorithm 2 offered reductions in radio up-time by a few percent, a number of links existed, for which the use of algorithm 2 reduced overall radio up-time by 5-20% over algorithm 1.

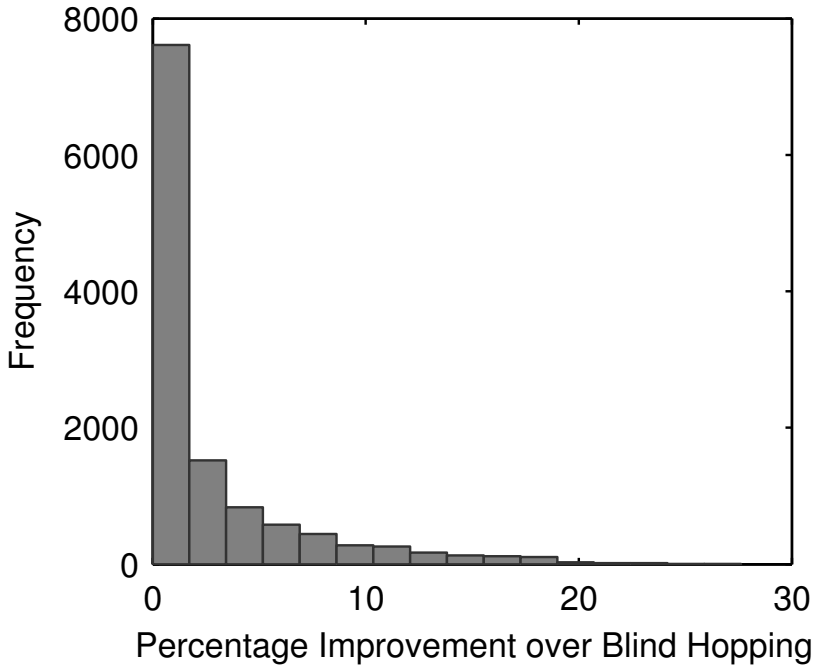

Figure 24: Histogram of the percent of radio uptime reduced by algorithm 2 over algorithm 1.

Figure 24 shows the evaluation of the cost function (2) on a sample of links in the playback data set. It should be noted that for a large part of the links the minimum of the cost function was evaluated at a whitelist size of 16, implying that the optimal whitelist contained all possible channels. However a number of links existed for which a minimum cost was achieved with a smaller white list size. The figure shows a range of network links, where dips in the cost function indicate optimal whitelist size.

The implementation of both algorithms on the MSP430-RF2500 motes performed according to design. Two motes were able to successfully synchronize and transmit packets based on both channel hopping algorithms. The motes were subjected to external WiFi interference, and adjusted their whitelist accordingly.

18 Discussion

A large number of the links in the playback data set reflected relatively high PDR values on all channels. In such cases, channel hopping could only provide a marginal improvement over a

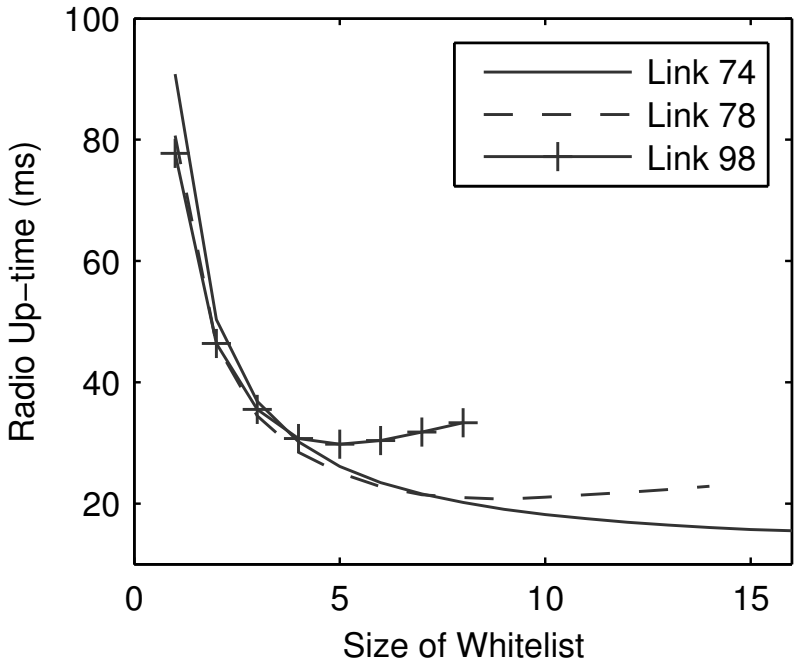

Figure 25: Cost function(2) evaluated for three sample links in the network. Minima in the cost function indicate optimal whitelist size.

single channel approach. A number of links, however, experienced fading on a subset of channels. If these channels were selected for transmission, this behavior would have had a significantly adverse impact on a single-channel solution. The blind channel hopping approach (algorithm 1) provided an advantage in such cases, as motes could still communicate on other channels. Since most links in the playback data set contained very high PDR values, the true benefit of algorithm 2 could best be noted on those links which only had a subset of reliable channels. Figure 23 shows that communication on a number of links reduced radio up-time by 5-20 percent over blind hopping when controlled by algorithm 2. As such, it is expected that algorithm 2 could best cope with very unreliable links in a network. Additionally, the time resolution of the playback data set was very coarse. A major benefit of algorithm 2 is its ability to rapidly augment its whitelist to cope with changes in the environment. Such sudden changes were not capture by the data set. Overall, however, an average 2.5 percent improvement over blind channel hopping, suggests that adaptive channel hopping methods could play a significant role in improving network efficiency and reliability.

19 Conclusions and future work

This chapter evaluated the performance of two channel hopping techniques for WSN. The first was based on the IEEE802.15.4e MAC standard, and blindly transmits packets, covering all 16 available channels over time. The second method performed a similar approach, but packets were only transmitted if they were contained inside an approved whitelist. The implementation of the second algorithm does not conflict with the IEEE802.15.4e standard, since it does not directly affect the transmission protocol, or the pattern at which nodes wake up to listen to incoming packets. It thus presents an elegant solution towards improving network connectivity without much overhead. A playback analysis of link traces, showed that the adaptive algorithm reduced network radio up-time by 2.5 percent, compared to blind channel hopping. Its impact can appreciated most on links where only a small set of channels have reliable packet delivery. In such cases the adaptive approach was able to reduce link-specific radio up-time by 20-30 percent. The adaptive channel hopping algorithm was also implemented on real-world hardware, thus validating the channel hopping concept, and showing how a transmitter's whitelist can be adjusted rapidly, and in real-time to reduce drain on radio resources. This chapter was the result of a paper, written with Thomas Watteyne, Steven Glaser, and Kris Pister.

Part V

Application Note: TDMA-based communications for mobile agent networks

The previous chapters motivated the need for time synchronized channel hopping for WSNs. Such TDMA-based communications are often designed with low-throughput applications in mind, where relatively small payloads are transmitted once every few seconds. Furthermore, nodes are required to synchronize to neighbors, a process which may take some time depending on the frequency of advertisement beacons. This makes it somewhat infeasible for mobile nodes to easily move around the area covered by the WSN, as well as to transmit large data streams. This chapter explores how the previously presented TDMA-based architecture can be modified to permit for multi-agent mobiles nodes to cooperate with a low-throughput stationary sensing backbone.

Small, cheap, highly mobile robots can be used to solve a wide variety of problems [51]. The availability of off the shelf components for such micro air vehicles (MAVs) has made them an active topic of current research and even commercial development (e.g. [52–54]. In particular, MAVs are well suited for exploration of unknown or difficult to access indoor environments, such as buildings and caves. They can be used to map out the space, or search within for items of interest. MAVs can also be used to deploy a wireless sensor network (WSN) for long-term persistent sensing of that environment.

Though there are many details differentiating specific mission scenarios, at the most basic level they all involve gathering data throughout an unknown environment and passing it to a remote base station. Only in a very narrow range of specifications can such a mission be accomplished without any communication at all (e.g. by storing data on board then physically returning to the base station); otherwise, multihop wireless network design becomes an integral consideration in such missions. Different mission parameters require different tradeoffs, and so there is no single network design that will satisfy all MAV requirements. However, there are a number common traits, which leads to a general framework that can be used as a starting point.

In this chapter we present a design for a network infrastructure that can accommodate typical MAV missions. By building on existing solutions for the various subproblems, we can create a hybrid network system that can effectively operate in a mode appropriate to the task at hand. In section 20 we will describe typical MAV missions, and break it up into subproblems. A description of previous research, along with hardware used for such missions will follow. The communications requirements inherent to a MAV will be discussed in section 22, followed by the state of the art protocol for communications in a wireless mesh network in section 23. These two modes will be combined into a single hybrid protocol, as described in section 24. Finally, we will identify areas where this implementation is still insufficient or unclear, and propose various solutions to complete the design in section 25. We will close with a discussion of where future research needs to be focused to properly design a network infrastructure to support MAV missions.

20 Missions and Requirements

A major motivating example of the need for MAV networks is shown in figure 26. In this example, a MAV has dropped a number of nodes in an unknown environment, and is wirelessly communicating with a base station over the resulting multihop mesh. The stationary network nodes serve the purpose of relaying information between the MAV agent and the base station, while also acting as a long-term persistent sensing platform, which can be left in place indefinitely to collect information about its environment. The challenge of designing such a system thus lies in adapting the wireless network to both long-term, low data rate reliable sensing missions, while accommodating the burst of short-term, high data rate transmissions of MAV agents.

The specific mission for which a MAV will be used will define the tradeoffs necessary to design its underlying network. However, many of those design decisions can be accomplished by tweaking various parameters in a general framework. At its most general, then, a MAV mission has the following components:

- Navigate through the environment
- Deploy sensors and repeater nodes

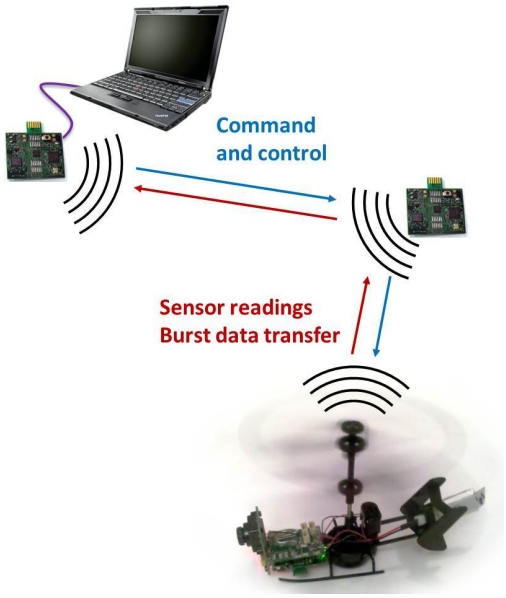

Figure 26: This schematic shows a deployed wireless sensor network infrastructure for a MAV mission. The laptop represents a base station, and acts as a network manager coordinating communications while serving as a sink for data generated by the MAV and sensor mesh. Data generated by the MAV is sent through the multihop mesh.

- Relay data back from MAV and deployed sensors

The scope of an MAV mission is greatly reduced without allowing multi-hop communication to the outside world. The arbitrary topology of the environment means that line-of-sight communication from the MAV to the outside world is often impossible. Thus, the MAV must carry along and deploy a payload of repeater nodes along its flight path. These drop-off nodes must establish a wireless network over which the MAV can communicate to the outside world. The nodes can be equipped with sensors, or simply serve as relay devices.

The objective of the general MAV mission is then to relay data from inside the unknown environment to the outside world. The data can come from the sensors on the MAV itself, using the deployed network nodes merely as a communications infrastructure. Data can also be generated by the deployed nodes. In the case of reconnaissance-type missions, data from MAV sensors will be composed of short, high volume bursts (camera feeds, microphones, etc). The stationary network nodes will generate low rate data over the long-term for purposes of anomaly detection or the monitoring of slowly-changing environmental phenomena (footstep detection, smoke detection, etc).

The most significant design constraints in such missions are weight and power consumption; the two go hand-in-hand. The lower the mass of the MAV with its payload, the less power is needed to fly, and so the longer a mission can be. Similarly, the lower the power consumption, the smaller the battery needed for a given mission, and so the lighter the weight. Since the deployed network is initially dead payload weight on the MAV, minimizing network power consumption minimizes the battery requirement on the drop off nodes (or eliminates in entirely in favor of energy scavenging sources), thus increasing the scope of possible missions.

Guidance of the MAV can either be handled remotely, via a central base station, or autonomously through on board processing. Remote control simplifies flight processing by offloading guidance and navigation decisions to a central operator. However, it imposes high throughput and low latency requirements on the network. Onboard data must be passed rapidly to the operator to facilitate real-time control of the MAV. Due to possible radio interference and limited centralized radio resources, this tends to limit the ability to utilize multiple MAVs during a mission. Autonomous MAV operation, on the other hand, greatly increases on-board sensing and processing requirements, but reduces required sustained data rates. Autonomous operation can also enable multiple MAVs in a single network, which increases complexity of the network routing and general network resource allocation.

Given such tradeoffs, there are a number of important parameters to consider when evaluating a MAV system. The potential capability of the system is directly related to the mission duration, which in turn depends on power consumption. The efficacy at which the MAV can accomplish the goals of the mission tie into the data transfer out of the system, which can be measured both by data throughput and network latency. Finally, it is important to evaluate how the system can scale in size and scope.

21 Sample hardware

Our MAV platform combines a miniature helicopter with an ultra-low-powered wireless node (figure 26). The GINA board [55] consists of a 2.4 GHz 802.15.4 radio, TI MSP430 microprocessor, and a number of inertial sensors. The GINA platform can be used directly as a MAV flight con-

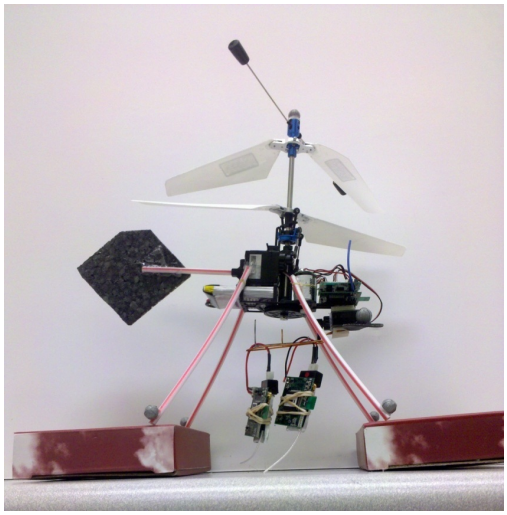

Figure 27: GINA hardware is used to implement an integrated MAV + WSN system.

troller, or as a relay for control signals from a central location [56]. The OpenWSN project [57] uses the GINA hardware to implement time syntonization, frequency channel hopping, mesh networking, and multihop communications. A MAV controlled by a GINA can be used to deploy additional GINA nodes for the stationary drop-off mesh infrastructure, as shown in figure 27. With standards compliant networking interfaces, this hardware forms a testbed for the implementation of the protocols described in this chapter.

22 MAV communication

A MAV interacts with its communication system in two major ways, which are tied to its two major functional subsystems: flight and sensing.

22.1 Flight

Different levels of autonomy require different amounts of communication. If the on-board controller is solely responsible for stability, then a human operator must remote-control the flight. In that case, the base station needs to get sensor data of sufficient fidelity to understand the environment in order to direct the MAV. This is generally at least video at a few frames per second, perhaps

augmented with additional inertial sensor data. This typically requires a throughput on the order of 100kbps. Then, the operator must send control signals back to the MAV. This is generally a much smaller amount of data, on the order of tens to hundreds of bits per second.

The data flowing in both directions needs to be sent in real time – minimizing latency is key. Without intelligence on board the MAV, a network delay on the order of tenths of a second could drive the MAV into a wall or obstacle. Slower, more stable MAVs may be able to tolerate the delay, but they waste energy in hover in low speed movement. As MAV functionality is optimized, round trip latency becomes even more critical.

The required data rates and latencies are typically not difficult to achieve in a point-to-point link, and can even be attained over a short multi-hop path. However, the performance rapidly degrades as the distance between the MAV and its basestation increases. Each additional hop along a network path adds both latency and network traffic, and as network traffic increases, the constant total bandwidth results in lowered end to end data throughput.

This saturation can be alleviated by requiring autonomy of the MAV. This can range from merely obstacle avoidance along a user-defined waypointed path to full autonomous path planning and guidance decisions. Autonomy thus minimizes or eliminates intervention from a human operator, and similarly, reduces the data that then needs to be sent to the base station for flight control. Nonetheless, data throughput and latency are still important parameters that impact the ability of human operators to influence the MAV mission.

22.2 Sensing

MAVs necessarily carry a number of sensors. At minimum, they include those required for stability (e.g. inertial sensors, LIDARs, etc.), but additional sensors may fall into one or more categories:

- Sensors that require significant mobility, e.g still and video cameras;
- Sensors that are too heavy to carry multiple, e.g. chemical sensors;
- Sensors that need to be used only once per location, e.g. thermal sensors, PIRs.

Many of these, such as thermal or chemical sensors, generate data at low rates, on the order of a few bytes per sample. Also, since the measurement often doesn't have high spatial or temporal variability, samples are infrequently generated. In this case, throughput is of minimal concern. If the gathered data does not impact the real time mission status, then latency requirements can also be relaxed.

Other sensors, however, can generate large volumes of data. Of these probably the most widely applicable and thus most common are cameras, which can generate anywhere from kilobytes to megabytes per sample, at rates from minutes per sample up to samples per second. Again, if the gathered data does not impact the mission, latency may not be so much of a concern, but maximizing data throughput is critical. High latency can also affect the throughput, though, if data flowing through the network saturates node capacities, resulting in dropped data. It is thus imperative that the network containing the MAV support both rapid data streams, and well as more sporadic, smaller size packets.

23 Stationary sensor network

23.1 Low power persistent sensing

Reliable communication with mobile MAV agents demands a stationary wireless network, which routes information to and form MAV agents, while serving as a persistent sensing platform to processes information about its surrounding environment. While MAV agents are deployed for short-term mission-specific purposes, this stationary network is expected to have deployment lifetimes on the order of days to months. As such, low power consumption is key for maximizing battery and deployment lifetime. Compared to the requirements imposed by mobile MAV agents, the data throughput requirements of stationary network nodes are expected to be low, generating packets on the order of seconds to minutes. Fro example, in mission critical settings the stationary network would be responsible for environmental-sensing tasks such as proximity sensing or footstep detection.

Table 2: Differences

Metric	Stationary network	MAV comms
Duration	10^6 seconds	10^3 seconds
Data rates	10^2 bits / second	10^5 bits / second
Latency	10^2 seconds	10^{-2} seconds
Network routing	stable	dynamic

24 Hybrid network

The time synchronized mesh protocol (TSMP) architecture was presented earlier as a means of facilitating low-power, reliable communications for a stationary WSN deployments. Table 2 shows the drastic, yet realistic differences which are imposed when comparing stationary network performance, to that of the MAV.

Rather than try to build a single network architecture that compromises on all the metrics, we instead propose to have a dual-mode network. The deployed stationary network by default operates as a standard TDMA-based mesh. As new nodes are dropped off by the MAV, they join the mesh and carry on as necessary. A modification is added to allow communication with the MAV; this communication is carried out on an independent set of channels. One set of channels is reserved for low throughput stationary mesh communications, while the remainder is only used when required by MAV agents.

24.1 TSMP modification

The stationary network dropped off by the MAV can implement a low power low duty cycle TSMP mesh. As nodes are dropped off, they join that mesh, with the routing configurations updating accordingly. The frame length must initially be set short enough to incorporate new nodes into the network on a rate comparable to the deployment of the nodes by the MAV. After the MAV mission, this can be slowed down as necessary.

Because the MAV is highly mobile, it will not maintain a constant set of neighbors, and so it is not useful as a member of the mesh. It still listens for advertisement (ADV) packets to maintain

synchronization with the network, but never sends out its own ADVs. Instead, it can send an acknowledgement packet (ACK) to an ADV to initiate communication into the network. After sending an ADV, stationary nodes listen to receive the ACK from a MAV. The advertisements are scheduled inversely proportional to neighbor count as usual so that on average, the MAV would be able to hear one ADV every frame without packet collisions. The frame length again must be short enough so that typically the MAV would remain within range of the node for the duration of the frame.

An extra slot is added to the frame after the ADV slot for communication from the network to the MAVs. If a node received an ACK to its ADV in the previous slot, it can then confirm with the MAV that it will switch to MAV mode for the frame, and also pass along any data it received from the base station for the MAV. If two MAVs respond to an ACK at the same time, a packet collision may occur. In this case, a randomized $back - off$ timer is used to retry communication at a later time. Once an MAV receives a $clear - to - send$ signal during the second slot of each frame it enters one of two throughput modes.

24.2 Low throughput MAV mode

When ACKing an ADV packet the MAV specifies the kind of data it is intending to transmit. If the the data to be transmitted can be fit into a few packets, without needing stringent latency requirements, it can send this over the reserved MAV channels to the stationary node in the mesh. The stationary node responds to the MAV ACK with slots in the TSMP schedule frame during which it is free and can receive data from the MAV. These slots can be predetermined prior to the deployment. The MAV then communicates with the mesh as would any other node during its scheduled slot, and again the data would pass through the network as normal. This mode is outlined in figure 28. This scheme also permits multiple MAVs to communicate with neighboring motes at the same time.

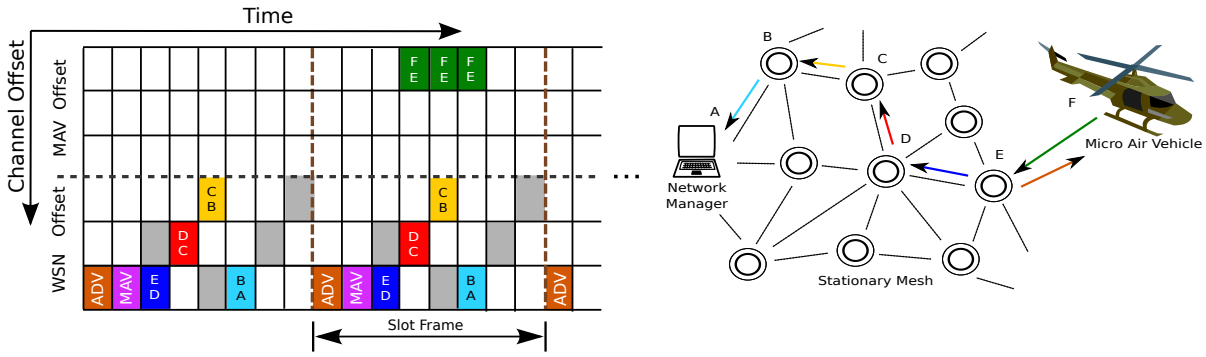

Figure 28: If the MAV has a small amount of data to send, it can request a number of slots to communicate to its nearest neighbor in the mesh. These slots can be predetermined by the manager. That data then gets sent over the mesh in the following several slot frames as would any other data generated in the mesh.

24.3 MAV burst mode

When the MAV has high volumes of data to transmit, or when latency is a primary concern, sending packets through mesh traffic becomes insufficient. In this case, the MAV requests a mode switch in its ADV ACK. If this is confirmed by the stationary node in the MAV slot (see figure 29), then a dedicated channel will be opened up between the MAV and the base station. The route from MAV to base station is determined during standard operation of the mesh, and is unlikely to change over the duration of a slot-frame. Starting from the node in contact with the MAV, then, each node in the mesh informs its direct parent of the mode switch, and then shifts to the burst mode.

This upstream notification can happen in two ways. The standard TSMP communication could be used, in which case the notification would reliably propagate upstream over the course of a slot-frame. Alternately, a separate slot following the MAV slot could be dedicated to this function, in which every node would listen for a mode-switch notification, and immediately turn around and forward such a message to its parent when one is received.

The burst mode communication happens over an independent set of channels from the main mesh. During the streaming mode slot-frame, nodes not involved in the direct path from the MAV to the base station carry on network operations as usual. Meanwhile, the direct path nodes establish a separate communications schedule over its channels. The burst communication schedule is similar to the standard TSMP communication; however there are a number of important differences.

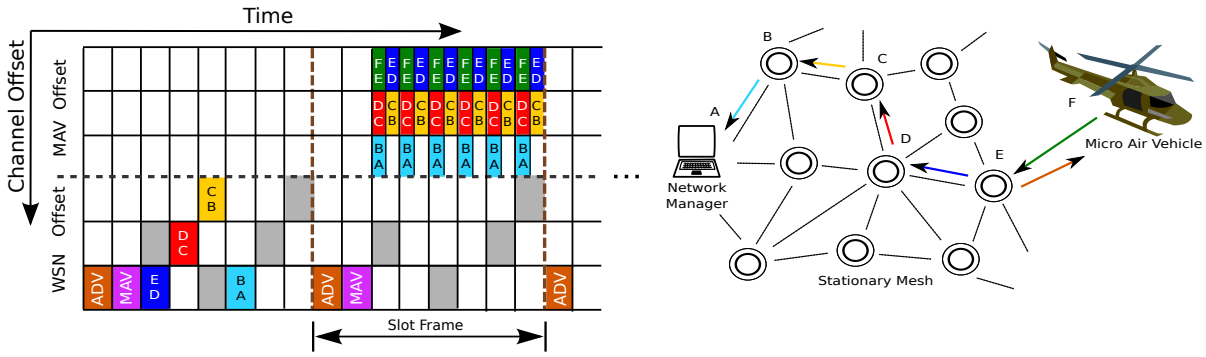

Figure 29: In burst mode, the nodes along the direct path from the MAV to the base station remove themselves from the TSMP network and communicate in a separate high throughput, low latency mode.

Since the data is a single, unidirectional, burst mode stream over a predefined path, most of the headers can be stripped out of the packets to yield a higher data throughput. For similar reasons, the slot length can be narrowed (halved to about 5ms in an IEEE802.15.4E setting) to facilitate higher packet density. Link level ACKs are eschewed in favor of higher bandwidth; typically this sort of data doesn't require very high reliability. This mode is diagrammed in figure 29.

This burst mode subnetwork persists for one slot-frame. On the next frame, the network returns to its original state. Data still in transit along the path can either be discarded or stored for transmission over the default channels. If the MAV has additional data it needs to transmit, it can request burst mode from the next ADV packet it hears.

25 Analysis and conclusions

25.1 Overview

Extensive analysis on TDMA networking, in particular TSMP, has demonstrated its importance for low power wireless networking [58–60]. The low throughput MAV mode is effectively just a slight extension to a standard IEEE802.15.4E TSMP network, with out-of-band communication reserved for direct point-to-point communication with the mobile MAV. Network maintenance and the remaining traffic is left to the primary unmodified IEEE802.15.4 mesh, thus ensuring the

robustness, reliability, and scalability inherited from the original standard.

Burst mode communication, on the other hand, augments the underlying TSMP MAC layer with an additional independent set of channels for the MAV burst data. This additional network is effectively a separate, stripped down TSMP network, derived from the original mesh but operating on orthogonal channels. Given a single unique path from a MAV to the base station a MAV can not simply send one packet after the other. Since its parent node must forward the packer, the MAV must pause until the parent node is ready to receive again. This leads to the transmission schedule seen in figure 29.

25.2 Tuning network parameters

The hybrid mode networking requires dividing the available network space into separate channels for stationary mesh and MAV communication. With one unique path to the base station, the optimal scenario involves the ability of the MAV to transmit a packet every other slot. If not enough slot offsets are available, the paths to the base station will fill the transmission schedule in a way that does not permit the MAV to transmit a packet every other slot. This is considered sub-optimal.

As a rule of thumb, the number of hops (paths to the base station) can only be twice the size of the reserved channels to ensure that the MAV can transmit a packet every other slot. As a MAV gets more hops away from the base station, channels should be allocated to ensure that burst mode is carried out at the fastest possible rate. As such, to achieve high throughput, the number of allocated MAV channels must be at least half the number of hops that the data must traverse (see figure 30).

With half-duplex radios, each node along the pipe must spend half its time receiving and half its time retransmitting the data along the pipe, thus limiting the throughput to half the available data bandwidth. However, with a sufficiently dense network, this limitation could be lifted. If two completely independent paths can be found from the MAV to the base station without overlapping intermediate nodes, data can be alternately sent along both pipes, maximizing the throughput to the full available data rate. In this case, the number of available channels would need to be at least the number of hops from MAV to base station.

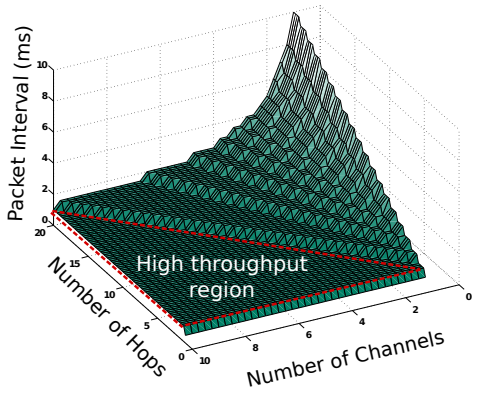

Figure 30: Given the expected maximum network distance from the MAV to the base station, a number of channels can be allocated to the MAV burst mode. As long as the number of channels is at least half the number of hops, the MAV will be able to communicate at the maximum supported throughput of the network given one unique path to the base station.

As there are only 16 channels available in the IEEE802.15.4 standard, allowing a more extensive network requires more channels to be allocated for MAV use, and fewer for the underlying stationary mesh. However, as shown in [60], the robustness and efficiency benefits of channel hopping become evident with only a small subset of channels. Peak performance can be achieved by allocating six channels to the stationary mesh; ten channels can thus be available for MAV communications, allowing for a reliable network up to 20 hops deep. This chapter was the result of a paper, written with Ankur Mehta, Steven Glaser, and Kris Pister.

Part VI

Conclusions

This thesis investigated a novel synchronization algorithm for TDMA-based WSNs. Furthermore, a number of frequency channel hopping methods were investigated to evaluate their impact on overall network reliability. It was shown how a slotted synchronization protocol could be implemented even using low-cost, high-drift, on-chip oscillators. This opens the door to a truly crystal-free mote, and enables another path towards the one-mm-cubed WSN node. The ability to carry out such a slotted communications scheme will be central to preserving power on future mote platforms, since radio up-time will be significantly reduced, permitting for significantly extended battery life.

Furthermore, it was emphasized that channel hopping is essential when considering adverse communication effects due to external interference or multipath fading. It was shown that channel hoping controllers can be designed to optimally utilize a subset of the allotted frequency band. The major benefit of such an approach relates higher PDR, and thus much higher reliability. Power consumption may not however be drastically improved due to adaptive channel hopping when compared to a blind hopping approach. Low PDR links may require retransmissions, but since the mote spends most of its time (99%) in a low power mode due to the slotted synchronizations scheme, retransmissions will only account for a smaller increase in energy use. This should hold true for most links in most applications, where blind channel hopping will be sufficient to facilitate robust communications. As mentioned in the previous chapter, the benefit of more advanced channel hopping methods applies mostly to a few links in the network that may contain mostly unreliable, low PDR links.

Power consumption is also a function of latency requirements. Nodes in TDMA based networks typically only transmit a few packets during a slot frame, thus conserving radio resources. It was show, however, that such network protocols can be modified to reliably support moving agents as well as high-throughput data streams, while not affecting overall network functionality and reliability.

This thesis leaves a number of possible future research questions open, primarily relating to improving synchronization accuracies and channel hopping algorithms through various bootstrapping methods. The major open research question, however, still relates to developing a crystal-free radio solution, which will permit the removal of the final crystal component on a mote, thus opening the door to the truly crystal-free WSN. Such a solution should not be too far in the future from becoming reality.

Part VII

References

References

[1] D. Networks, "Technical Overview of Time Synchronized Mesh Protocol (TSMP)," Dust Networks, Tech. Rep., 2007.

[2] H. Karl and A. Willig, *Protocols and Architectures for Wireless Sensor Networks*, H. Karl and A. Willig, Eds. John Wiley & Sons, Inc., Hoboken, New Jersey, June 2005.

[3] T. Watteyne, S. Lanzisera, A. Mehta, and K. Pister, "Mitigating Multipath Fading Through Channel Hopping in Wireless Sensor Networks," in *IEEE International Conference on Communications (ICC)*. Cape Town, South Africa: IEEE, 23-27 May 2010.

[4] T. Watteyne, A. Mehta, and K. Pister, "Reliability Through Frequency Diversity: Why Channel Hopping Makes Sense," in *6th ACM International Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks (PE-WASUN)*, Tenerife, Canary Islands, Spain, 26-30 October 2009.

[5] B. Kusy, P. Dutta, P. Levis, M. Maroti, A. Ledeczi, and D. Culler, "Elapsed time on arrival: a simple and versatile primitive for canonical time synchronisation services," *International Journal on Ad Hoc and Ubiquitous Computing*, vol. 1, no. 4, pp. 239–251, 2006.

[6] D. Mills, J. Martin, and W. Burbank, *Network Time Protocol Version 4: Protocol and Algorithms Specification*, IETF Std., June 2010, RFC5905.

[7] *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, IEEE Instrumentation and Measurement Society Std. IEEE Std 1588-2008, 24 July 2008.

[8] J. Han and D.-K. Jeong, "A Practical Implementation of IEEE 1588-2008 Transparent Clock for Distributed Measurement and Control Systems," *IEEE Transactions on Instrumentation and Measurement*, vol. 59, no. 2, pp. 433–439, February 2010.

[9] S. Lanzisera, D. Zats, and K. S. J. Pister, "Radio Frequency Time-of-Flight Distance Measurement for Low-Cost Wireless Sensor Localization," *IEEE Sensors Journal*, 2011, to appear.

[10] K. Pister and L. Doherty, "TSMP: Time Synchronized Mesh Protocol," in *International Symposium on Distributed Sensor Networks (DSN)*. Orlando, FL, USA: IASTED, 16-18 November 2008.

[11] A. Tyrrell, G. Auer, and C. Bettstetter, "Emergent Slot Synchronization in Wireless Networks," *IEEE Transactions on Mobile Computing*, vol. 9, no. 5, pp. 719–732, May 2010.

[12] K. Pister and L. Doherty, "TSMP: Time Synchronized Mesh Protocol," in *Parallel and Distributed Computing and Systems (PDCS)*, Orlando, Florida, USA, 16-18 November 2008.

[13] T. Watteyne, A. Mehta, and K. Pister, "Reliability through frequency diversity: why channel hopping makes sense," in *PE-WASUN '09: Proceedings of the 6th ACM symposium on Performance evaluation of wireless ad hoc, sensor, and ubiquitous networks*. New York, NY, USA: ACM, 2009, pp. 116–123.

[14] L. Doherty, W. Lindsay, and J. Simon, "Channel-Specific Wireless Sensor Network Path Data," in *16th International Conference on Computer Communications and Networks (ICCCN)*. Turtle Bay Resort, Honolulu, Hawaii, USA: IEEE, August 13-16 2007, pp. 89–94.

[15] B. Kerkez, T. Watteyne, M. Magliocco, S. Glaser, and K. Pister, "Feasibility analysis of controller design for adaptive channel hopping," in *VALUETOOLS '09: Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools*. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009, pp. 1–6.

[16] W. Thomas, L. Steven, M. Ankur, and K. Pister, "Mitigating multipath fading through channel hopping in wireless sensor networks," in *IEEE International Conference on Communications (ICC)*, 2010.

[17] *IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements. Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)*, IEEE Std., Rev. 2006, 8 September 2006.

[18] *CC2500, Low-Cost Low-Power 2.4 GHz RF Transceiver (Rev. B)*, Texas Instruments, Inc., 19 May 2009, data Sheet SWRS040C [available online].

[19] *IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements. Part 15.1: Wireless medium access control (MAC) and physical layer (PHY) specifications for wireless personal area networks (WPANs)*, IEEE Std., Rev. 2005, 14 June 2005.

[20] *HART Field Communication Protocol Specifications, Revision 7.1, DDL Specifications*, HART Communication Foundation Std., 2008.

[21] ISA, "Isa-100.11a-2009: Wireless systems for industrial automation: Process control and related applications," 11 September 2009.

[22] L. van Hoesel and P. Havinga, "A lightweight medium access protocol (LMAC) for wireless sensor networks: Reducing Preamble Transmissions and Transceiver State Switches," in *International Conference on Networked Sensing Systems*, 2004.

[23] O. D. Incel, S. Dulman, and P. Jansen, "Multi-channel Support for Dense Wireless Sensor Networking," in *First European Conference on Smart Sensing and Context (EuroSSC) Lecture Notes in Computer Science 4272. Springer.*, Enschede, the Netherlands, 25-27 October 2006, pp. 1–14.

[24] Y. Kim, H. Shin, and H. Cha, "Y-MAC: An Energy-efficient Multi-channel MAC Protocol for Dense Wireless Sensor Networks," in *International Conference on Information Processing in Sensor Networks (IPSN)*. St. Louis, Missouri, USA: IEEE, April 22-24 2008, pp. 53–63.

[25] C. S. Kang, K. H. Chang, R. Enns, K. Pister, L. Winkel, C. Powell, and J. Gutierrez, "MAC Enhancements for Time Slotted Channel Hopping," IEEE, Tech. Rep., 15 August 2009, IEEE P802.15 Working Group for Wireless Personal Area Networks (WPANs) [work in progress].

[26] T. Watteyne, D. Zats, and K. Pister, "TSCH: Time Synchronized Channel Hopping," under review.

[27] K. Sohrabi and G. Pottie, "Performance of a novel self-organization protocol for wireless ad-hoc sensor networks," in *Vehicular Technology Conference, 1999. VTC 1999 - Fall. IEEE VTS 50th*, vol. 2, 1999, pp. 1222 –1226 vol.2.

[28] G. C. Bell and C. Federspiel, "Demonstration of datacenter automation software and hardware (dash) at the california franchise tax board," *Internal Report, Prepared for the California Energy Commission*, 2009.

[29] D. Culler, L. Doherty, J. Hill, M. Holden, C. Kiers, S. Kumar, J. Kusuma, S. Morris, K. Pister, K. Ramchandran, B. Robbins, J. Scholtz, M. Scott, R. Szewczyk, B. Tang, and A. Woo. (1997, June) 29 palms fixed/mobile experiment: Tracking vehicles with a uav-delivered sensor network. [Online]. Available: http://robotics.eecs.berkeley.edu/ pister/29Palms0103/

[30] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson, "Wireless sensor networks for habitat monitoring," in *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*. New York, NY, USA: ACM, 2002, pp. 88–97.

[31] "IEEE std 802.15.4-2006: Wireless medium access control (MAC) and physical layer (PHY) specifications for low-rate wireless personal area networks (WPANs)," http://ieeexplore.ieee.org/servlet/opac?punumber=11161.

[32] IEEE P802.15 Working Group for Wireless Personal Area Networks (WPANs) Std. IEEE Std 802.15.4e, Rev. D0.01/r3, 13 September 2009.

[33] J. M. Kahn, R. H. Katz, and K. S. J. Pister, "Next century challenges: mobile networking for smart dust," in *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, ser. MobiCom '99. New York, NY, USA: ACM, 1999, pp. 271–278. [Online]. Available: http://doi.acm.org/10.1145/313451.313558

[34] B. Warneke, K. S. J, and S. Dust, "Smart dust: Communicating with a cubic-millimeter computer," *Classical Papers on Computational Logic*, vol. 1, pp. 372–383, 2001.

[35] B. W. Cook, S. Lanzisera, and K. S. J. Pister, "SoC Issues for RF Smart Dust," *Proceedings of the IEEE*, vol. 94, pp. 1177–1196, 2006.

[36] L. Lin, "Mems post-packaging by localized heating and bonding," *Advanced Packaging, IEEE Transactions on*, vol. 23, no. 4, pp. 608 – 616, nov 2000.

[37] H. Reichl and V. Grosser, "Overview and development trends in the field of mems packaging," in *Micro Electro Mechanical Systems, 2001. MEMS 2001. The 14th IEEE International Conference on*, 2001, pp. 1 –5.

[38] B. Floyd, K. Kim, and O. Kenneth, "Wireless interconnection in a cmos ic with integrated antennas," in *Solid-State Circuits Conference, 2000. Digest of Technical Papers. ISSCC. 2000 IEEE International*, 2000, pp. 328 –329.

[39] R. Su, S. Lanzisera, and K. S. J. Pister, "A 2.6psrms-period-jitter 900mhz all-digital fractional-n pll built with standard cells," in *ESSCIRC (ESSCIRC), 2011 Proceedings of the*, sept. 2011, pp. 455 –458.

[40] V. L. Pushparaj, M. M. Shaijumon, A. Kumar, S. Murugesan, L. Ci, R. Vajtai, R. J. Linhardt, O. Nalamasu, and P. M. Ajayan, "Flexible energy storage devices based on nanocomposite paper," *Proceedings of the National Academy of Science*, vol. 1041, pp. 13 574–13 577, Aug. 2007.

[41] M. Lutz, A. Partridge, P. Gupta, N. Buchan, E. Klaassen, J. McDonald, and K. Petersen, "Mems oscillators for high volume commercial applications," in *Solid-State Sensors, Actuators and Microsystems Conference, 2007. TRANSDUCERS 2007. International*, june 2007, pp. 49 –52.

[42] B. DeMartini, J. Rhoads, K. Turner, S. Shaw, and J. Moehlis, "Linear and nonlinear tuning of parametrically excited mems oscillators," *Microelectromechanical Systems, Journal of*, vol. 16, no. 2, pp. 310 –318, april 2007.

[43] Y. Lee, D. Sylvester, and D. Blaauw, "Synchronization of ultra-low power wireless sensor nodes," in *Circuits and Systems (MWSCAS), 2011 IEEE 54th International Midwest Symposium on*, aug. 2011, pp. 1 –4.

[44] A. M. Mehta and K. S. J. Pister, "Frequency offset compensation for crystal-free 802.15.4 communication," in *Advanced Technologies for Communications (ATC), 2011 International Conference on*, aug. 2011, pp. 45 –47.

[45] *eZ430-RF2500 Development Tool User's Guide*, Texas Instruments, April 2009, SLAU227E [available online].

[46] *MSP430x2xx Family User's Guide*, 2008, SLAU144E [available online].

[47] D. Sullivan, D. Allan, D. Howe, and F. Walls, "Characterization of clocks and oscillators," *NIST Tech Note 1337*, vol. 1337, 1990.

[48] F. L. Lewis and V. L. Syrmos, *Optimal Control, 2nd Edition*, F. L. Lewis and V. L. Syrmos, Eds. Wiley-Interscience, 1995.

[49] J. Polastre, R. Szewczyk, and D. Culler, "Telos: Enabling Ultra-Low Power Wireless Research," in *International Conference on Information Processing in Sensor Networks: Special track on Platform Tools and Design Methods for Network Embedded Sensors (IPSN/SPOTS)*. Los Angeles, CA, USA: IEEE, April 2005.

[50] K. Srinivasan, M. A. Kazandjieva, S. Agarwal, and P. Levis, "The beta-factor: Measuring Wireless Link Burstiness," in *6th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, Raleigh, NC, USA, 5-7 November 2008.

[51] V. Kumar and N. Michael, "Opportunities and challenges with autonomous micro aerial vehicles," aug 2011.

[52] "AeroVironment, inc. (AV) : Nano air vehicle (NAV)," http://www.avinc.com/nano.

[53] "AR.Drone parrot," http://ardrone.parrot.com.

[54] "Ascending technologies, GmbH," http://www.asctec.de/.

[55] A. M. Mehta and K. S. J. Pister, "WARPWING: A Complete Open Source Control Platform for Miniature Robots," in *International Conference on Intelligent Robots and Systems (IROS)*. Taipei, Taiwan: IEEE/RSJ, October 2010.

[56] A. Mehta and K. Pister, "Warpwing: A complete open source control platform for miniature robots," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, oct. 2010, pp. 5169 –5174.

[57] "Berkeley's OpenWSN project," http://openwsn.berkeley.edu/.

[58] "Technical overview of time synchronized mesh protocol (TSMP)," http://www.dustnetworks.com/docs/TSMP$_W hitepaper.pdf$.

[59] T. Watteyne, S. Lanzisera, A. Mehta, and K. S. J. Pister, "Mitigating multipath fading through channel hopping in wireless sensor networks," in *Communications (ICC), 2010 IEEE International Conference on*, may 2010, pp. 1 –5.

[60] T. Watteyne, A. Mehta, and K. Pister, "Reliability through frequency diversity: why channel hopping makes sense," in *Proceedings of the 6th ACM symposium on Performance evaluation of wireless ad hoc, sensor, and ubiquitous networks*. ACM, 2009, pp. 116–123.