DISS. ETH NO. 24379

Adding more PHY to the MAC: Exploiting Physical Layer Effects in Wireless Networks

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES of ETH ZURICH

(Dr. sc. ETH Zurich)

presented by

Michael König

M. Sc., ETH Zurich, Switzerland

born on 06.08.1990

citizen of
Germany

accepted on the recommendation of

Prof. Dr. Roger Wattenhofer, examiner
Prof. Dr. Olaf Landsiedel, co-examiner

2017

TIK-Schriftenreihe-Nr. 171

Abstract

Traditional wireless algorithms all too often ignore the special qualities of the wireless medium. In this thesis, we propose new wireless transmission primitives for various applications and evaluate each of them on a wireless sensor network. In particular, we focus on integrating two properties into our primitives: the availability of transmission power control and the capture effect.

First, we consider the problems of traffic prioritization and running multiple wireless algorithms in parallel. We propose a technique allowing to simultaneously run multiple algorithms of different priorities, with minimal overhead in terms of bandwidth and latency. This is done by assigning each priority a range of admissible received signal strengths at each node, and employing the capture effect to automatically enable reception of only the strongest incoming signal. The setup is transparent to the algorithms: each appears to have complete access to the network's resources as long as no algorithm of a higher priority wishes to use them. We discuss which properties of the network graph and the wireless hardware are beneficial to our technique.

Second, we demonstrate the feasibility of achieving constructive interference using commodity wireless sensor nodes. In contrast to previous work, our technique does not rely on global external events as reference, but instead aims to minimize the errors in clock synchronization and transmission timing. Our evaluation shows that our technique is able to achieve constructive interference in over 30% of cases, even after multiple minutes of sleep.

Third, we propose a class of transmission primitives which decouple packets' synchronization headers from their payloads, such that two or more different senders may contribute to a single received packet. We explore 2 applications: 1) enabling reception attempts of very weak packets, e.g., across a network chasm, and 2) the injection of shorter packets into longer ongoing transmissions. We investigate ways to vastly reduce the problems incurred by using a mismatching synchronization header for reception. In practice, we are able to successfully decode up to 30% of cross-chasm packets and up to 70% of injected packets.

Fourth, we examine how transmission power control can improve wireless schedules. Based on the classic RAND scheduling algorithm we develop a version employing power control called PowerRAND. The schedules generated by PowerRAND are 20–25% shorter, i.e., achieve a 25–33% higher throughput than RAND. Our practical evaluation shows that these schedules are just as feasible in practice. Further, we discuss how power control provides flexibility to schedules in the face of changing environments.

Zusammenfassung

Traditionelle drahtlose Algorithmen ignorieren allzu häufig die speziellen Qualitäten des drahtlosen Mediums. In dieser Dissertation stellen wir neue drahtlose Übertragungsprimitiven für diverse Anwendungen vor und evaluieren jedes auf einem drahtlosen Sensornetzwerk. Wir konzentrieren uns insbesondere auf die Integration von zwei Eigenschaften in unsere Primitiven: die Verfügbarkeit von regulierbaren Sendestärken und den Erfassungseffekt.

Erstens betrachten wir die Problemstellungen von Verkehrspriorisierung und dem parallelen Ausführen mehrerer drahtloser Algorithmen. Wir schlagen eine Technik vor, die es erlaubt zugleich mehrere drahtlose Algorithmen verschiedener Prioritäten auszuführen, mit minimalen Extrakosten an Bandbreite und Latenz. Dazu wird jeder Priorität ein Intervall von zulässigen Signalempfangsstärken an jedem Knoten zugewiesen, und der Erfassungseffekt eingesetzt, um automatisch nur das Empfangen des stärksten eintreffenden Signals zu ermöglichen. Dieser Aufbau ist den Algorithmen transparent: jeder scheint vollständigen Zugang zu den Netzwerkressourcen zu haben, so lange kein Algorithmus einer höheren Priorität sie zu nutzen wünscht. Wir erörtern welche Eigenschaften des Netzwerkgraphen und der drahtlosen Geräte vorteilhaft für unsere Technik sind.

Zweitens demonstrieren wir die Realisierbarkeit des Erzielens von konstruktiver Interferenz mittels handelsüblicher Sensorknoten. Im Kontrast zu vorigen Werken verlässt sich unsere Technik nicht auf globale externe Ereignisse als Referenz, sondern sie versucht stattdessen die Fehler in Uhrensynchronisation und Übertragungszeitpunkt zu minimieren. Unsere Evaluation zeigt, dass unsere Technik in über 30% der Fälle konstruktive Interferenz erzielen kann, sogar nach mehreren Ruheminuten.

Drittens stellen wir eine Klasse von Übertragungsprimitiven vor, die bei Paketen die Synchronisationskopfteile von den Nutzlasten trennt, sodass zwei oder mehr verschiedene Sender zu einem einzelnen empfangenen Paket beitragen können. Wir untersuchen 2 Anwendungen: 1) das Ermöglichen von Empfangsversuchen von sehr schwachen Paketen, z.B. über eine Netzwerkschlucht, und 2) die Injektion von kürzeren Paketen in längere laufende Übertragungen. Wir erforschen Wege, um die Probleme, die man sich durch die Verwendung falscher Synchronisationskopfteile zum Empfang einhandelt, erheblich zu reduzieren. In der Praxis können wir bis zu 30% der Querschluchtpakete und bis zu 70% der injizierten Pakete erfolgreich dekodieren.

Viertens prüfen wir wie regulierbare Sendestärken drahtlose Sendepläne verbessern können. Basierend auf dem klassischen Sendeplan-Algorithmus RAND entwickeln wir eine Version, die regulierbare Sendestärken einsetzt, namens PowerRAND. Die von PowerRAND generierten Sendepläne sind

20–25% kürzer, d.h., erzielen einen 25–33% höheren Durchsatz als RAND. Unsere praktische Evaluation zeigt, dass diese Sendepläne in der Praxis gleichermaßen durchführbar sind. Ferner erörtern wir wie regulierbare Sendestärken Sendeplänen Flexibilität in Anbetracht sich ändernder Umgebungen verleihen.

Acknowledgements

My time as a PhD student in the Distributed Computing Group was a very interesting and formative experience. I gained a lot of insight into the inner workings of universities as well as the academic research machinery – and even got to contribute myself, which I greatly enjoyed doing. However, the thesis that you are now reading would not have been possible without the help of many people that I would like to thank in the following.

First, I want to thank my supervisor Roger Wattenhofer for giving me the opportunity to write my thesis in his group and supporting me throughout the time. While allowing me to freely pursue my research, he was still always able to offer helpful advice and ideas. His sense for viable paper topics was invaluable to my publication efficiency and kept me going when the going got tough.

Then, I would also like to thank my co-referee Olaf Landsiedel for taking the time to review this thesis and to serve on my committee.

Furthermore, there are also the other people that made my time in the Distributed Computing Group a wonderful experience; my colleagues and co-workers. I want to thank (in alphabetical order) Barbara Keller for semi-regularly surrendering Jara and her living room to StarCraft, Beat Futterknecht for eating administrative problems for breakfast, Benny Gächter for spreading the gospel of C++, Christian Decker for emanating Bitcoin education, Conrad Burchert for providing a new unit of measure and modding a game for me, Darya Melnyk for being one of the few reliable tögglers, David Stolz for always being ready to strike up a conversation, little Georg for trying to be rude and being a reliable cake supplier, Gino Brunner for carrying the student burden for most of the group, Jara Uitto for his Finnish approach to everything, Jochen Seidel for being a pragmatic Byzantine and keeping StarCraft alive, Klaus-Tycho Förster for being a beacon of kindness and a partner in crime providing the group's pun supply, Laura Peer for bringing up many a programming puzzle, Manuel Eichelberger for unlocking my office every morning, Pankaj Khanchandani for staying late but still being available for töggeli, Pascal Bissig for not compromising on taste and his evil töggeli shots, Philipp Brandes for turning on the lights and being my most dedicated subject, Roman Lim for his excellent wireless testbed support, Samuel Welten for supervising many a ridiculous student thesis involving smartphones, Sebastian Brandt for both splitting and wearing lots of hair, Simon Tanner for being a reliable office mate, Stefan Schindler for being a Rust evangelist, Stephan Holzer for being a relaxed office mate, Thomas Ulrich for his creative ideas, Tobias Langner for showing genuine passion in most things he does, Yuval Emek for being a great nanny, and Yuyi Wang for always being helpful.

Last but not least, I want to thank the people most important in my life: my parents Kirsten and Stefan, my sister Sandra, as well as my grandparents for always having supported me in my computer science education and doctoral studies, and my dear friends Marcel, Patrick, Till, Tim and Wolfram for always being up for a game or a silly side project when I needed to take my mind off work.

Contents

1 Introduction **1**

2 Protocol Layering **5**
2.1 Introduction 5
2.2 Related Work 8
2.3 The Capture Effect 10
2.4 Layering Protocols 14
2.5 Example Application 21
2.6 Test Results 25
2.7 Summary and Future Work 27

3 Maintaining Constructive Interference **33**
3.1 Introduction 33
3.2 Related Work 34
3.3 Experiment Setup 36
3.4 Timing Requirements 38
3.5 Clock Synchronization 40
3.6 Transmission Synchronization 44
3.7 Constructive Interference 48
3.8 Summary and Future Work 53

4 Capturing Attention Using the Capture Effect **55**
4.1 Introduction 55
4.2 Related Work 58
4.3 Concepts 59
4.4 Experiment Setup 64

4.5 Transmission Synchronization . . . 70
4.6 Mapping Symbols . . . 71
4.7 Results . . . 72
4.8 Summary and Future Work . . . 76

5 Tempering Wireless Schedules **79**
5.1 Introduction . . . 79
5.2 Related Work . . . 82
5.3 Link Model . . . 84
5.4 Prediction Model . . . 88
5.5 Tempering RAND . . . 90
5.6 Experiment Setup . . . 95
5.7 Results . . . 98
5.8 Summary and Future Work . . . 104

6 Conclusion **107**

1 Introduction

Over the last two decades, wireless technology has become a mainstay of our everyday lives. Mobile devices connect to the Internet over Wi-Fi and communicate locally using Bluetooth. Cell phones connect to wide-reaching cellular networks, and modern navigational systems are unthinkable without the position estimations obtained from GPS signals. It is estimated that over 60% of the population of both Western Europe and North America uses a smartphone at least once a month [59]. But not only mobile devices are able to profit from wireless technology: battery-powered sensor nodes may be deployed completely without any wired infrastructure, yet still are able to form large multi-hop sensor node networks for easy sensor data aggregation. Due to these networks' self-reliance, they are found in various applications, but especially in long-term monitoring. Examples include the monitoring of structural integrity in buildings [50], wheel wear in trains [29], wide-area weather conditions [71] and avalanche risk [30].

Wireless communication faces unique challenges and opportunities stemming from the shared nature of the wireless medium. Most notably, all transmissions are broadcast transmissions by default, i.e., they are heard and may be received by all receivers "in range" of the sender. One particular implication of this property is that, unlike in most modern wired networks, the signals of simultaneous transmissions will compete at each prospective receiver, acting as interference to each other. If two links (sender-receiver pairs) are able to transmit data reliably individually, they may not be able to do so concurrently: depending on the environment's geometry, both, only one, or even neither may be able to transmit successfully.

In spite of the widespread usage of wireless technology, these special properties are still frequently abstracted away. This is usually done in favor of simplicity, but comes at the cost of less efficient medium usage. For example, multi-hop wireless networks are often modeled as a graph in which two nodes are connected by an edge if and only if they are able to communicate. A transmission is then expected to be successful only if only a single neighbor – namely the intended sender – of an intended receiver is transmitting for the duration of the transmission. In reality, a link may be feasible even in presence of interfering links, if the interfering links' signals are sufficiently weaker at the receiver: due to path loss and obstacles a link's signal may be strongly attenuated once it arrives at the receiver. Another option rarely considered by today's wireless protocols is *power control*, i.e., the ability to configure different transmissions to use different transmission powers. This option can, for example, be used to reduce the transmission power for already strong links, such that they cause less interference, and to ensure that the desired transmission is received at a contested receiver.

Breaking these imperfect abstractions and simplifications will be the general theme of this thesis. We propose several protocols and techniques drawing benefits from doing away with the above restrictions, and verify each of them in practice. We conduct all our experiments on TelosB wireless sensor nodes [62] (often also referred to as "Tmote Sky") deployed in the FlockLab testbed [46], which is situated in an office building and as a result experiences typical background noise as well as an ever-changing environment. Hence, all our practical results are based on the low-power and low-bandwidth IEEE 802.15.4 wireless standard. However, we believe each of our approaches to be applicable to a majority of today's wireless standards.

We begin by considering the problem of priority traffic in Chapter 2. Most wireless systems suffer from the lack of a dedicated control plane: high-priority control messages have to contend for medium access the same as any other message. Existing methods providing quality of service guarantees in this setting rely upon opportunistic sending or on scheduling mechanisms, and as a result incur undesirable tradeoffs in either latency or impact on regular non-priority traffic. We present a technique to simultaneously execute multiple protocols of different priorities, without compromising bandwidth or latency of regular traffic not affected by priority traffic. Using power control and moderately tight synchronization we exploit the capture effect to give each protocol almost complete access to the network's resources as long as no protocol of higher priority wishes to use them. We examine which impact the properties of the network graph and the capabilities of the wireless hardware have on the effectiveness of our technique. We suggest an example scenario of a wireless sensor network of fire detectors, with a low-priority protocol collecting statistical data and confirming aliveness, while a high-priority protocol wishes to report fire alarms to a base station as quickly as possible. Our testbed implementation of this example application achieves near optimal latency and bandwidth for priority traffic while not disturbing low-priority traffic where it is separated sufficiently in space or time.

We investigate achieving constructive interference (CI) on sensor nodes in Chapter 3. I.e., the ability to avoid interference of two or more identical incoming signals by synchronizing the signals well enough. As an added benefit, the resulting signal has a higher signal strength than any single of the senders alone would have been able to create. Traditionally, achieving CI required specialized timekeeping hardware. Recently, the ability and interest to employ CI distributedly at any time using groups of ordinary single antenna wireless sensor nodes have grown. The IEEE 802.15.4 wireless standard we are working with uses a chip frequency of 1 MHz. This means signals need to be synchronized with an error below 0.5 µs to allow for CI. Hence, excellent clock synchronization between nodes as well as precise transmission timing are required. We implemented and tested a prototype addressing the implementation challenges of synchronizing the nodes' clocks up to a precision of a few hundred nanoseconds and of timing transmissions as accurately as possible. Our results show that, even after multiple minutes of sleep, our approach is able to achieve CI in over 30% of cases, in scenarios in which any influence from the capture effect can be ruled out. This leads to an increase in a packet's chance of arrival to 30–65%, compared to 0–30% when transmitting with either less synchrony or different data payload. Further, we find that 2 senders generally increase the signal power by 2–3 dB and can double the packet reception ratio of weak links.

In Chapter 4, we propose a new class of wireless transmission schemes decoupling synchronization headers from payloads to create new transmission primitives involving a second sender. By transmitting a synchronization header only, we can let nearby nodes receive fragments of a packet without having to receive that packet's synchronization header, and by using the capture effect we can overwrite portions of the payload of longer ongoing packets. We explore two scenarios potentially benefiting from such schemes. A) First, we consider crossing a network chasm over which all links are of poor quality: by broadcasting a fabricated packet header on the receiving side of the chasm all receiving nodes are informed to record the packet to the best of their ability. B) Second, we investigate the insertion of short high-priority packets into longer lower-priority transmissions from a different sender. This has the advantage that high-priority senders do not need to wait for the medium to become free but can begin sending at once, while receivers lose only the affected portion of their already incoming low-priority packets. Further, we examine two techniques to reduce the amount of symbol decoding errors caused by using a mismatching synchronization header: 1) careful transmission timing and 2) correction of deterministic symbol decoding errors. In scenario A) these techniques improve the chance of every part of a packet being received successfully by some node on the receiving side of the chasm from 5% to up to 30%. In scenario B) we reach successful decoding of the injected packet in up to 70% of cases.

Lastly, in Chapter 5, we demonstrate that the often overlooked feature of power control can in fact be lucrative for wireless algorithms to incorporate. In particular, we present PowerRAND, an extension of the simple yet reliable classic RAND scheduling algorithm. Further, we explore how to deal with the challenges of a changing environment which all scheduling algorithms face. For such problems, power control provides a unique flexibility. To optimize the utilization of a wireless channel, time slotting and scheduling tailored to the traffic demands have proven to be one of the most efficient methods in high load networks. Traditionally, the work in this area was focused merely on finding sets of links that could send simultaneously. As transmission power control is a widespread feature in hardware, this adds an additional degree of freedom to schedule creation: how strongly should each sender transmit? Theory results show that it is possible to construct scenarios in which power control allows creating shorter schedules. In practice, theory is not the same as practice. For example, the range of possible transmission power values is limited and not arbitrarily fine-grained. Our results show that using power control we can obtain 20–25% shorter schedules, which is equivalent to an increase of 25–33% in overall throughput. Additionally, by rewarding links being scheduled a second time within the same schedule, we are able to further improve slot utilization, i.e., achieve a higher average throughput per slot. In our experiments we confirm that these schedules are just as reliable as schedules not employing power control.

We conclude in Chapter 6 with a short summary of what we learned and an outlook towards possible future work.

2 Protocol Layering

2.1 Introduction

While the deployments of wireless networks continue to grow in number and size year by year, protocol designers are still struggling to understand how to most efficiently use the wireless medium. Not only is operational disruption through environmental noise caused by other networks or microwave ovens oftentimes unpredictable in urban settings, but due to the broadcast nature of wireless transmissions, nodes participating in a network frequently experience interference with nearby nodes of the same network.

To tackle this medium access problem given a fixed frequency spectrum, generally one of two approaches is used: (1) opportunistic sending (CSMA) or (2) scheduling (TDMA).

(1) Opportunistic sending follows a first-come-first-serve philosophy and hopes for a free channel at the time of traffic emergence, sending immediately, asking questions later. After sending, an explicit or implicit acknowledgment from the recipient is required to determine whether the transmission was successful or needs to be repeated. Variants of opportunistic sending such as clear channel assessment (CCA) and request-to-send/clear-to-send (RTS/CTS) have been proposed to reduce the number of haphazard collisions, but suffer from hidden or exposed terminal problems and intro-

duce overhead in terms of packets sent and latency, which especially in the case of RTS/CTS may quickly grow very noticeable [48, 64, 77].

(2) The alternative is to employ one or more nodes with the role of scheduling authorities. These nodes manage the permissions to send packets in their immediate network neighborhood. Each node is either allotted a periodically recurring time slot for sending, or may need to first explicitly request a reservation for the channel from its local scheduler(s). This approach may completely avoid collisions during regular operation but incurs latency and possibly also bandwidth penalties. While generally opportunistic sending is preferred for its simplicity and flexibility in most scenarios, the scheduling approach has also found its way into widely deployed systems such as Bluetooth [34].

Clearly, neither approach is optimal in all scenarios, nor is every scenario served well by either approach. For example, consider the case of an emergency signal needing to travel to a destination node in as little time as possible. Using opportunistic sending, progress may stall almost indefinitely when the network is under heavy load, while a scheduling approach might reserve every second slot for emergency messages which guarantees arrival in twice the minimum possible time at the cost of halving the number of slots available for regular non-emergency traffic. Hybrid MAC layers such as Z-MAC [65] have been proposed, with the goal of combining the advantages of CSMA and TDMA. Z-MAC realizes this by dynamically switching between CSMA and TDMA based on network load.

Network mechanisms designed to ensure fairness or more specifically offering guarantees about network performance are grouped under the term quality of service (QoS). While QoS research for wireless networks is an area with a wealth of history, certain aspects of wireless networks are yet to be fully understood and utilized. Among these is the so-called *capture effect* (also known as *physical layer capture*). For a long time, protocol designers tried to avoid collisions whenever possible, working under the assumption that any collision of wireless packets at a receiving node inevitably leads to the failure of that node to decode any of the messages. This loss rate has been shown to have been significantly overestimated [73, 76]. This is due to the capture effect, a phenomenon which oftentimes allows the receiver of a wireless transmission to continue correctly decoding the transmission, in spite of interference caused by other transmissions starting during the original transmission.

We consider a “protocol” to be a self-contained distributed algorithm using the network to transmit messages, coping with lost messages and usually avoiding collisions where possible. In this chapter, we propose a technique to “layer” such protocols of different priority levels on top of each other using the capture effect, effectively enabling a priority process to use almost all the resources of a network, while at the same time allowing lower level processes separated from the priority traffic in space and/or time to use the network at no additional overhead. Note that giving unrestricted resource access to a protocol necessarily implies that it may starve all lower-priority protocols. We also propose a mechanism to only administer a share of the resources to a protocol, but this unavoidably introduces a latency overhead.

We require a certain degree of clock synchronization (clock difference below 160 µs between any pair of nodes with distance at most two hops) to be able to make best use of the capture effect, and impose the notion of time slots on the network. Hence, we assume that at least one of the layered protocols contains a component periodically resynchronizing all nodes. Furthermore, we require the wireless hardware to offer transmission power control, which is a common feature even among older hardware.

The basic idea is to cause the capture effect whenever a node would receive multiple packets in the same time slot. This is done by choosing the transmission power of each node such that it falls into one of multiple precomputed bands of reception power at the intended receiving node. Given these bands are separated well enough, the receiving node will almost always (in over 98% of cases) be able to decode the packet in the strongest band without error. This implicit prioritization lets us avoid the overhead caused by more explicit measures such as schedules. On the other hand, there are some inherent disadvantages tied to our technique, namely the need for time slotting and the predetermination of transmission powers, removing the ability to intentionally save energy on short links and to save hops with long links requiring the highest possible transmission power.

We specifically target wireless sensor networks (WSNs), which typically form networks with a relatively large connectivity graph diameter and favor node quantity over advanced wireless capabilities. Our technique accommodates these conditions particularly well. For example, as higher layer protocols only disturb their immediate neighborhood in the connectivity graph, more spread out networks are more likely to benefit.

We tested our technique on an example alarm reporting protocol. Our implementation is based on Contiki [12] and achieves near optimal alarm reporting latency and almost no packet loss on the high-priority layer, while when under load indeed causing comparatively little disturbance to the underlying low-priority traffic which we use to ensure node liveness and keep the nodes' clocks synchronized.

2.2 Related Work

Already in 1976, the capture effect in FM receivers was modeled by Leentvaar et al. [40]. To combat it they proposed using bandlimiting at the receiver. The capture effect is not a phenomenon limited to FM transmissions. Ash [1] showed that it is possible to obtain an equivalent and even stronger effect in AM receivers.

While the capture effect had at first been considered undesirable, it was soon ascribed inadvertent performance boosts in common wireless scenarios such as slotted ALOHA [9] and everyday 802.11 traffic [48, 76]. Soon, a number of environmental influences like noise, path loss, shadowing and fading were identified to be contributing to the capture effect's potency as a general packet reception enhancer [6, 44].

Other research was conducted on the details of packet timing, as common transceiver hardware does not facilitate switching reception from one packet to another mid-demodulation. Thus, if a much stronger packet starts during the reception of a weaker one, both packets are lost (save for the leading portion of the weaker one). A well-studied quirk of the capture effect is that it may occur even when the stronger signal arrives after the weaker one, as long as it still arrives before the end of the synchronization header of the weaker signal [36, 73, 76, 78].

The obvious solution to the "stronger packet arrives too late" problem is to continuously scan the medium for synchronization headers, even during packet reception. This requires more specialized hardware support, but has nevertheless already been thoroughly investigated [36, 39, 52, 76]. To make best use of the capability to switch to stronger packets during reception (also known as "message in message"), Manweiler et al. [52] discuss how careful ordering of transmissions enables the parallel utilization of traditionally conflicting sender-receiver pairs.

When it comes to low-power wireless networks such as those comprised of sensor nodes, the meticulous study by Son et al. [73] provides a solid foundation. While they find that occurrence of the capture effect can be guaranteed given a large enough SINR value, they find a significant gray region of up to 6 dB to exist in practice. Further, they find the SINR threshold to be heavily dependent on the transmitting hardware and the selected transmission signal strength. Yuan et al. [78] continue this study and propose a packet reception model for concurrent transmissions, including the special case of constructive interference.

Nyandoro et al. [60] consider the scenario of an 802.11 access point and several clients split into low-priority and high-priority clients. They propose using a significantly higher sending power for the high-priority clients. Due to the capture effect collisions between packets from high and low-priority clients will then always be solved in favor of the high-priority client. Patras et al. [61] go as far as to suggest deliberately fluctuating sending power levels in order to make the capture effect more likely to occur in case a collision takes place. They show that in practice this can translate to throughput gains of up to 25%. As links become more heterogeneous, though, this effect decreases, and instead an increase in fairness can be observed.

Lu et al. [49] proposed the *Flash* flooding protocol, in which a flooding schedule is forgone in favor of letting every reached node simply broadcast a few times. The capture effect enables correct reception at nodes receiving packets from different neighbors at sufficiently different strengths. The protocol also implements fallback mechanisms to ensure flooding is able to proceed at nodes which experience only destructive interference due to the arriving signals being too similar in strength. This approach was shown to reach flooding latencies close to the theoretical optimum, reducing previous latency values by up to 80%.

Liang et al. [45] created *RushNet*, a data delivery framework harnessing the capture effect to achieve low-overhead prioritization similar to the work in this chapter. RushNet distinguishes low-priority bulk transfer and latency-sensitive high-priority traffic, which exactly matches our example application (see Section 2.5). In contrast, our technique is designed for use with arbitrary wireless protocols and supports a larger number of traffic priorities where link qualities permit.

Various approaches to coordinate simultaneously running protocols competing for medium access have been suggested. Flury et al. [20] proposed "slotted programming", dividing time into slots and assigning every protocol a fixed portion of the slots. This framework is implemented in a fashion transparent to the protocols, effectively making them independent and modular building blocks for larger systems. Note that in contrast to the method presented in this chapter, slotted programming incurs a significant penalty on the total throughput when protocols are unable to make use of the scheduled slots assigned to them.

The same work also includes a proposal for an alarm mechanism: Flury et al. suggest alarmed nodes transmit a specific waveform at maximum power. Other nodes, upon detecting the waveform, become alarmed and start transmitting the waveform as well, thus spreading the alarm. The authors find that, even without synchronization between the nodes, the collision of the signals is not detrimental to the spread of the alarm. However, extending this scheme to alarm signals carrying more information than the alarm's presence itself appears to be difficult. Additionally, these alarm signals are undirected and will prevent any regular traffic in the network.

Cidon et al. [8] propose establishing a control plane for Wi-Fi networks by inserting high-power "flashes" into regular packets. These flashes are a waveform of far higher amplitude than the rest of the signal and are added to regular data symbols, effectively erasing those symbols. By exploiting the underlying OFDM encoding, which sends multiple redundant copies of each data bit either separated in time or frequency, they are able to insert on the order of 50,000 flashes per second without causing a packet loss rate of more than 1%. The occurrence and spacing of these flashes may then be chosen to represent out-of-band data. While this approach does not require additional frequency bands or time slots for control messages, its main disadvantage is its reliance on specialized hardware.

For the specific scenario of time-critical alarm message propagation, Li et al. [42] propose incorporating slots allocated for emergency messages into a regular scheduling mechanism, but to employ slot stealing to avoid wasting network bandwidth in the absence of emergencies. A short while after the start of a slot assigned to emergency messages, if the slot is detected to remain unused, nodes may steal and use the remainder of the slot to send regular traffic. They further provide a simulation framework tailored to such wireless alarm systems. In constrast to our work, their method relies on an explicit scheduling mechanism to designate recurring slots for emergency messages. This incurs an overhead in latency and does not scale well to a larger number of distinct priorities, as the time required to detect slot use grows and thus further erodes the concept of time slotting.

A different approach, employed to great effect by cellular networks technologies such as LTE, is to send and receive on multiple different frequency bands, allowing to use a subset of them as an independent control plane [22]. In contrast, we do not consider the use of multiple frequency bands and restrict ourselves to simple wireless transceivers able to send and receive on a single band at a time only.

2.3 The Capture Effect

In this section, we will go into detail about the capture effect and its inner workings, as it is integral to the method we propose. Furthermore, we will discuss the exact parameters for its occurrence we measured on the hardware and testbed we will be using for our example implementation.

The *capture effect* is a term describing the general phenomenon of wireless receivers being able to decode the strongest of multiple signals without error, effectively completely ignoring the weaker signals. Standard wireless hardware is designed to send and receive wireless data strings in the form of discrete packets, which may be inserted into the noisy carrier medium at arbitrary points in time. Due to this, harnessing the capture effect on such hardware is limited to a certain set of scenarios.

Typically, wireless receivers use specific pre-defined chip patterns to detect the start of a transmission, so-called *synchronization headers.* They serve multiple purposes: For one, they allow to, with a high probability, identify a starting transmission amongst the environmental noise and hence avoid mistaking noise for a transmission even in settings where transmissions create signals barely stronger than the noise. Another purpose, whose side-effect is particularly important for summoning the capture effect, is the aligning of the receiving wireless transceiver's internal clock with the phase of the signal. This essentially means that once a receiver has detected a synchronization header, it can configure itself to easily decode the following data symbols and stream their values into some kind of memory.

As a result, upon hearing a synchronization header, receivers effectively commit to receiving a particular transmission, locking their clocks to that transmission's phase and often also its length (which in many physical layer protocols is transmitted amongst the first few data symbols). This behavior is especially desirable when bursts of noise frequently occur in the environment, but partially corrupted packets may still be valuable, either due to error correcting codes or simply full data integrity not being a critical requirement.

When two or more signals can be heard at a receiver simultaneously, they act as noise to each other, i.e., cause interference for one another. Generally, it is impossible to decode the weaker signal(s), unless the hardware is capable of more advanced techniques, such as decoding and subtracting the stronger signals first or using a coding scheme such as CDMA. However, our proposed method is aimed at scenarios employing simple sensor nodes and does not rely on such functionality. Hence, we will assume that when a stronger transmission starts during the reception of a weaker transmission, the weaker transmission is certain to become corrupted.

On the other hand, when the stronger transmission starts first, it can nearly always be received completely without error. This is in spite of the fact that a prediction based purely on the signal powers and the classical SINR model will conclude that data corruption would occur for a significant range of power differences. The locking onto the phase of the signal effectively diminishes the influence of competing transmissions and lowers the SINR threshold required to be met for correct reception.

Due to the nature of the use of synchronization headers, the requirement, that the stronger transmission comes first, is significantly eroded: The weaker transmission may come first, as long as its synchronization header is not completely received before the synchronization header of the stronger transmission begins. This happens because the stronger synchronization header destroys the end of the weaker synchronization header, hence, the receiver no longer considers the weaker signal to be a valid packet. Further,

	Packet A (strong)	**Packet B (weak)**
$\Delta < -d_A$	correct	correct
$-d_A < \Delta < 0$	correct	not at all
$0 < \Delta < \tau_1$	correct	not at all
$\tau_1 < \Delta < \tau_2$	tapering chance	not at all
$\tau_2 < \Delta < d_B$	not at all	partially corrupted
$d_B < \Delta$	correct	correct

Table 2.1: Listing of the possible outcomes of two packets arriving at the same receiver simultaneously. Δ specifies the time difference $t_A - t_B$ between the arrival times of the two packets and d_i denotes the duration of packet i. τ_1 and τ_2 denote two thresholds between which the probability of a correct reception of packet A tapers off.

in some scenarios the window for the stronger transmission to arrive has been reported to extend even into the first 3 bytes of the weaker transmission [78].

A summary of the outcomes in each possible scenario for 2 packets can be seen in Table 2.1. Of special interest here are the constants τ_1 and τ_2, which dictate the timing thresholds required for the capture effect to occur. We expect these to closely match the length of the synchronization header.

To find the exact values of τ_1 and τ_2 for our specific hardware and testbed setup, we conduct a few experiments pitting two senders against each other to try to successfully transmit a packet to a single receiver. To mimic actual protocol performance as close as possible we only use the hardware's innate ability to keep time and synchronize clocks. Hence, the receiver periodically sends a packet both senders use to synchronize their local clocks to the receiver's. The period is chosen to keep the clock error strictly below $\pm 0.5\,\mu s$, which is close to optimal considering the clocks' frequency of 4 MHz. We will discuss the details of achieving such synchronization precision in Chapter 3. In the remaining slots, both senders send packets with varying transmit power levels or delays.

Figure 2.1 shows the results for one sender using a significantly larger sending power and, due to links of similar quality being used, significantly higher received signal strength. We observe the probability of the receiver capturing the stronger packet to fall to zero roughly over the interval from $\tau_1 \approx 150\ \mu s$ to $\tau_2 \approx 165\ \mu s$. At the bitrate of 250 kbit/s 160 µs correspond to 10 symbols or 5 bytes, which matches the length of the synchronization header (4 bytes of preamble plus the immediately following SFD (start of frame delimiter) byte). The spread $\tau_2 - \tau_1 \approx 16\ \mu s$ matches the length of

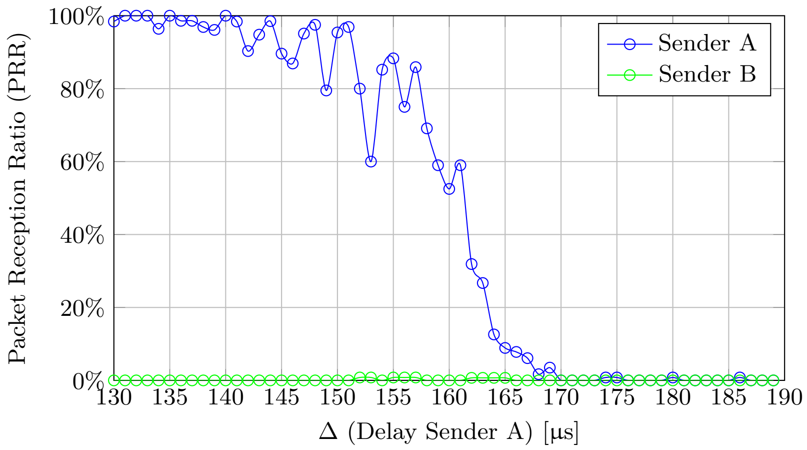


Figure 2.1: Occurrence probability of the capture effect at a receiver plotted against the time difference between the two incoming packets. Sender A's signal was significantly stronger averaging at −69 dBm RSS (Received Signal Strength), compared to Sender B at −82 dBm RSS. 130 samples were taken per delay value.

one symbol and is roughly centered around $\Delta = 160$ µs.

It is worth noting that the study by Yuan et al. [78], also using TelosB motes, measured the transition window to be $\tau_1 \approx 128$ µs to $\tau_2 \approx 224$ µs, resulting in a spread of roughly 6 symbols or 3 bytes. We are not completely certain about the cause of this discrepancy, but presume the difference in path quality to play a significant role: we used 3 nodes part of the testbed, located in different rooms, while they had placed the nodes in clear line of sight at a distance of 1 meter.

We also measured the effect of signal strength difference on capture probability. Figure 2.2 shows our results for the case of both senders sending simultaneously ($|\Delta| < 1$ µs), but one sender varying its transmission power. To determine the difference in RSS and counteract the fluctuation of the link qualities over time, we additionally measured each sender's RSS value within at most 0.5 µs of each sample. We observe the gap $thres_{power}$ between either of the senders having their packets captured with a high probability, say $PRR > 90\%$, to be about 5 to 7 dB. This roughly matches the values cited in existing literature, e.g., [73]. It is worth noting, however, that the gap size $thres_{power}$ likely depends on the environment, since in most existing models a higher noise floor implies needing a higher signal power to beat the SINR threshold required for successful capture. Further, we find that

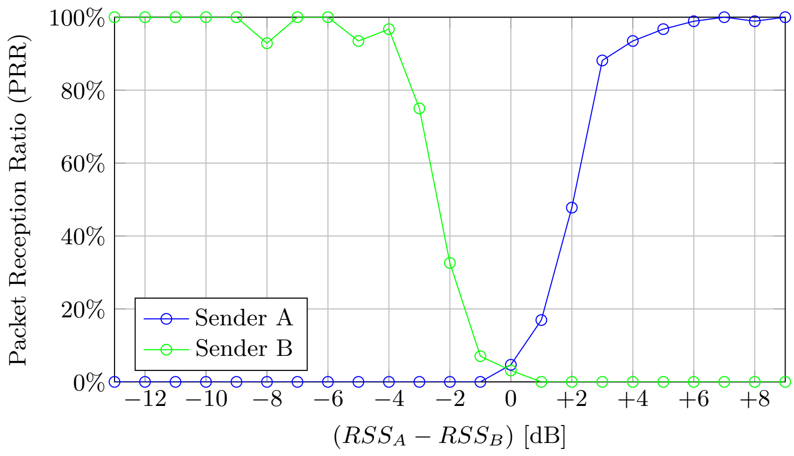


Figure 2.2: Occurrence probability of the capture effect at a receiver plotted against the power difference between the two incoming packets. Sender A's transmit power was varied over its whole available range, producing RSS_A values from −89 dBm to −68 dBm, while Sender B's transmit power was kept constant, producing RSS_B values from −79 dBm to −76 dBm. 40 samples were taken for each of the available 32 transmit power values, resulting in a total of 1108 comparable instances, i.e., instances in which we were able to measure both senders' RSS values.

the probability of either packet to be received successfully drops to around 5% when the signal strengths are identical.

In conclusion, we observe that in order to be able to reliably call on the capture effect in a slotted setup (see next section), competing senders should have a clock difference below 160 µs and a power difference above 5 dB. This level of synchronization we can achieve by synchronizing about once every few minutes.

2.4 Layering Protocols

2.4.1 Slot Logic

Traditionally, executing multiple protocols in parallel is likely to incur a penalty on the utilization of the network resources and/or the performance of the protocols themselves. For example, if two protocols access the medium alternatingly using time slots, up to half the slots may be wasted, while one of the protocols is idle and the other has demand for more than its share of time slots. Further, in this scenario, information propagation latency is doubled, as no information can leave a node sooner than 2 slots after its arrival. Using more opportunistic approaches, such as when using CSMA/CA (Carrier Sensing Multiple Access with Collision Avoidance), the problems mentioned above do not occur: any number of idle protocols do not influence the network's utilization or the performance of the other protocols. However, when the load on the network becomes too large, unfairness and starvation become threats to effective operation.

In this section, we will detail our proposed method of parallelizing, or *layering*, k protocols with the aim of combining the benefits of the approaches mentioned above: no unused network resources under load, while also offering fairness and prevention of starvation. Further, our method allows prioritization of protocols, giving higher-priority protocols almost complete access to the network's resources at the cost of possible starvation of lower-priority protocols in areas of the network not sufficiently separated from the high-priority traffic in space or time. Finally, this section will discuss how network topology and environment influence the number of layers our method can support.

The core idea is to deliberately provoke the capture effect at every node whenever it is destined to receive multiple packets at the same time. To do so, we enforce time slotting and require the clock difference between any two nodes within a node's immediate neighborhood to be below τ_1 (see Section 2.3). By ensuring all potentially competing packets start within a time interval of length τ_1, we obtain that which packet is to be received in a time slot is solely dependent on the arriving packets' signal strengths, but not on their relative timings. Given a sufficient spread in signal strengths, the capture effect is almost certain to enable successful reception of the packet with the strongest signal. We will discuss why this assumption is a reasonable one to make below.

The next piece of the scheme is to use transmission power control at each sender to specify the "layer" of each packet. As nodes may have varying distances and link qualities to each other, the transmission power cannot simply be derived from the layer of the packet to be sent, but must consider the destination node. In effect, for every receiving node a set of incoming signal strength intervals needs to be chosen, different enough to be distinguishable by the capture effect, but similar enough to fit within the range of signal strengths each of the neighboring nodes can produce. Thus, for every sending node, for each of its neighbors and for each of the layers the correct sending power needs to be determined.

Finally, every protocol is uniquely assigned to a layer. We label the layers $1, \ldots, k$, where the protocol of layer k has the highest priority and the protocol of layer 1 has the lowest. Every slot, every node executes

Algorithm 1: Pseudocode for Slot Logic

```
out ← $\varnothing$
foreach protocol $\mathsf{P}_l$ with layer $l \in \{1, \dots, k\}$ do
    $\mathsf{P}_l$.compute_slot()
    if $\mathsf{P}_l$.outgoing_packet $\neq \varnothing$ then
        out ← $\mathsf{P}_l$.outgoing_packet
        $\mathsf{P}_l$.outgoing_packet ← $\varnothing$
if out $\neq \varnothing$ then
    Transmit out this slot (using the correct power for out's target
      and layer).
else
    Listen this slot.
    in ← incoming_packet
    if in $\neq \varnothing$ then
        $\mathsf{P}_{\mathsf{in.layer}}$.process_packet(in)
```

Algorithm 1: First, it performs each protocol's slot computation separately, while storing the packet the highest layer protocol wants to send (out) and discarding all others. If any packet was chosen this way, it is sent at the sending power corresponding to its destination and protocol layer. If no packet was chosen, the node listens for the duration of that slot and delivers any received packet to the correct protocol.

This setup attempts to give each protocol the illusion of being the only protocol present. This is achieved by protocols experiencing a "packet loss" if a protocol of higher layer is active at the same time: if a higher layer is overriding the sending of a lower layer packet, that lower layer packet simply appears to have been lost in transit; conversely, if a packet of a higher layer is overriding the reception of a lower layer packet, that packet's fate appears indistinguishable from true packet loss as well. As occasional packet loss is a common occurrence in almost every environment due to noise bursts or interference, most wireless algorithms are innately capable of recovering from a loss of packets. Hence, they are perfectly suitable to be used as lower layer protocols. The highest layer protocol experiences no packet loss due to the presence of other layers (with one exception noted below), but is still subject to the usual environmental impediments. If the environment is in fact controlled enough to not suffer any such packet loss, as might be the case in clinical settings such as perhaps data centers, a protocol relying on a low packet loss ratio may be used as the highest layer.

We do not consider queuing packets from multiple protocols desiring to send from the same node in the same slot, as this would tamper with possible protocol-internal slot schedules of protocols whose packets have been delayed. This would damage the illusion, and require significant changes to the way protocols for use in the lower layers are designed, such that common known protocols can no longer easily be used. Simulating packet loss is hence a cleaner solution, while the option of allowing layering-aware protocols to immediately know if their packet was dropped, such that they may queue it for the next slot if desired, is still available.

There is one scenario, however, in which the illusion inevitably breaks down. As we are assuming that the wireless hardware is not capable of receiving while transmitting, a problem occurs when a node is choosing to send in a slot due to a protocol of layer i, but would in the same slot receive a packet on layer $j > i$. Here the protocol of layer j will experience packet loss due to a lower layer protocol. Unfortunately, it is impossible to prevent this scenario from occurring without also introducing significant overhead to all other scenarios: if the traffic demand on layer j can occur spontaneously, for instance, to propagate an alarm event, every node's layer i protocol may be in any state, including having chosen to send in that particular slot. If one forces layer i a priori to not send in certain slots, the latency and bandwidth penalties tied to TDMA are inevitable.

We found that for applications, in which the highest level protocol aims to achieve the lowest latency possible (such as the example application discussed in Section 2.5), a reasonable workaround is to send every high-priority packet twice in successive time slots. Note that while this does double the amount of packets sent, the latency only increases when the described scenario indeed occurs. Running common algorithms, a node that is sending in slot t is unlikely to send again in slot $t + 1$ as there would not have been any input in slot t to instigate another outgoing transmission. This is even true if multiple layers wanted to send in slot t, as all their packets would have been either sent or discarded in slot t. Hence, a high-priority packet sent in two successive time slots is bound to arrive in at least one of the two slots.

2.4.2 Power Choices

The main difficulty now lies in ensuring a good spread in the signal strengths of all packets received at each node in the same slot. One assumption we make is that the individual protocols avoid causing multiple of its packets to collide at the same node. This is reasonable especially for protocols following the traditional school of thought, which dictates all simultaneous packet arrivals to be fatal collisions. Given this assumption and the fact that every protocol runs on its own unique layer, all packets arriving at a node in the same slot belong to different layers and should thus have sufficiently different signal strengths for the capture effect to be able to enable reception of the strongest packet.

There exists a tradeoff between the number of available layers (and thus number of parallelizable protocols) and the achievable spread of received signal strengths at each node. The network topology and in particular the homogeneity of the network's link qualities play a large role in enabling a higher number of layers to be well separated at each node. "Well-separatedness" requires the difference in received signal strength between every pair of layers to exceed the threshold of $thres_{power}$ (which we found to be at least 5 dB on our testbed, see Section 2.3). We found that in a perfectly homogeneous setting where every link is either of high quality (high range of powers usable for successful transmissions) or not of significant power, the number of available layers becomes maximal. Using our hardware we found up to 4 or 5 sending powers to be distinguishable at a receiver, see Figure 2.3(a). Such a high number of layers (4 or more) is likely only feasible in settings with a high degree of control over node positioning and environmental influences.

More commonly, networks contain varying levels of heterogeneity, with some areas containing only long-distance/low-quality links, some areas more tightly packed with low-distance/high-quality links, and many areas being cases in between. For our hardware, especially these in-between cases spell trouble due to the granularity of selectable sending powers decreasing sharply as power values decrease, see Figure 2.4. As a direct result, a bottleneck for the number of layers forms at nodes with both long-distance and short-distance links. In the example of Figure 2.3(b), the low-quality link of Sender D can only provide receive powers in the range from -89 to -78 dBm, a range which Sender A cannot reach with any power setting. In Figure 2.3(c), while no extremely high-quality links are included, the higher-quality links still offer only a very low power granularity in the range feasible for lower-quality links.

We find that for our hardware 2 clearly distinguishable layers are possible in essentially all topologies, but identify the occurrence of both "long" and "short" links at a single node as the main bottleneck. Note that the cause for the bottleneck is not present in nodes which have only long or only short links, as in these cases the links' power ranges overlap very well. Essentially, the smaller the upper bound on the difference between the longest and shortest link at any node in the network, the higher the number of available layers.

When faced with the problem of the network supporting too few well-separated layers, there are several possible solutions. For one, the problem may be addressed directly by excluding or repositioning such mixed-link nodes or some of their neighbors. Another alternative is to employ wireless transceivers offering better-suited transmission power control options.

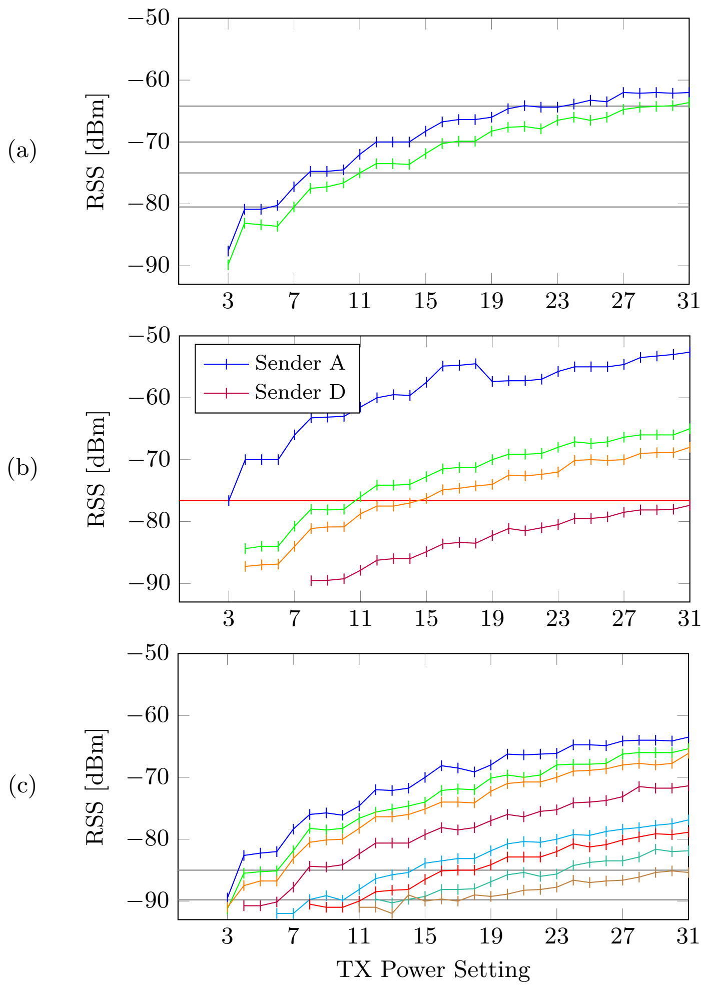


Figure 2.3: Shown are the RSS (received signal strength) values for different output powers for different links at the same receiver.
(a) Two links of similar quality easily allow 4 layers to be differentiated (horizontal lines). A possible fifth layer could identify with powers below -88 dBm.
(b) An example of a node with an extremely short distance neighbor (Sender A). No power setting allows a packet from Sender D to be captured while Sender A is sending as well.
(c) An example of a typical node with no very close neighbors. 2 layers are supported by all links.

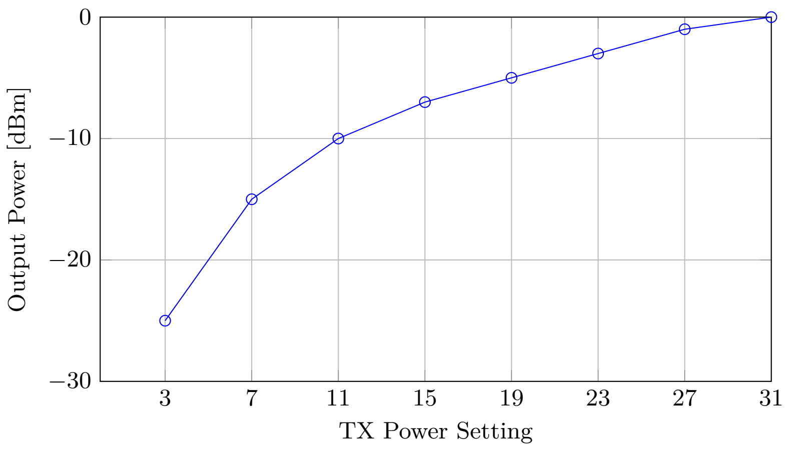


Figure 2.4: The 32 available output power settings on the CC2420 wireless transceiver [75]. Note that half the available values cover only a 7 dB interval and settings below 3 are not usable.

Finally, in some scenarios compromising the quality of the layer separation a bit by lowering the required signal strength spread at each receiver may be feasible, especially if only few or unimportant nodes are affected. Latter may lead to occasional inadvertent inversion of packet priorities and true destructive packet collisions, which some applications may be able to tolerate.

If one wishes to have multiple protocols to have the same priority and be entitled to equal share of the medium, it is not advisable to assign both protocols the same layer of receive powers. Since packets of the same strength arriving at a node will not trigger the capture effect but instead lead to destructive packet collisions, none of the packets would be decoded correctly. Instead, we recommend having a separate layer for each protocol, but rotating through the protocol assignments for the layers on a slot number basis. I.e., for two protocols, simply swap their layer assignments every c slots. We suggest choosing the value of c to be around 50 to 100 to avoid each protocol suffering very frequent packet loss when both protocols are under load.

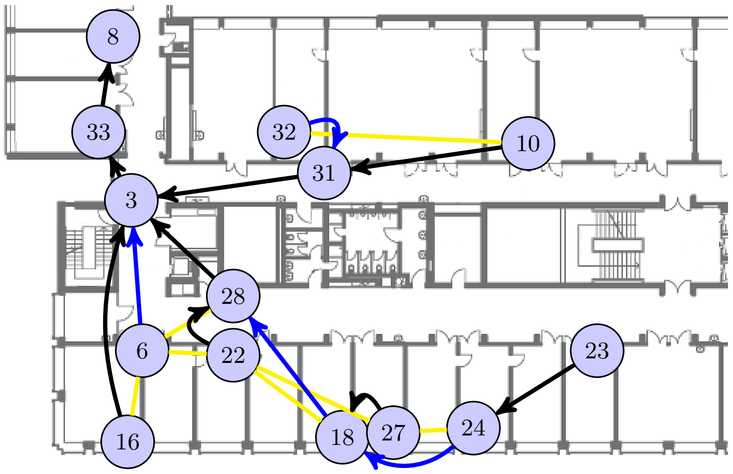


Figure 2.5: A part of the FlockLab testbed which we used to conduct our experiments on. An example of a convergecast tree with node 8 as the root node is shown. Yellow edges indicate links not part of the tree. To avoid collisions within the convergecast layer, no two sibling branches connected by links may execute simultaneously. Hence, every node, once it is woken, first queries all its black edge children in parallel, and then, in a second step, its blue edge children.

2.5 Example Application

To verify and measure the effectiveness of our method on real world wireless sensor networks, we chose an example application highlighting the supposed benefits of our method and implemented it on FlockLab [46] which spans the floor of an office building (see Figure 2.5).

We consider the scenario of fire detectors covering the rooms of a building with the goal of reporting the outbreak of a fire as quickly as possible to a base station, or *root node*, which to the network is just a normal node with a special role. The root node has a wired connection to the building facilities and can escalate the alarm if it is informed of a fire by one of its neighbors. In its wireless capability it matches a regular node and as such it can only hear a local neighborhood of nodes, necessitating the propagation of a fire alarm over multiple hops.

Additionally, we require the liveness and proper functioning of all nodes to be regularly verified so that defunct nodes may be replaced in a timely manner. We regard these liveness tests to be of less urgency than the fire alarms and thus do not mind fire alarms having priority access to the wireless medium. Hence, in our model the liveness tests will be executed as a protocol on layer 1, while the fire alarm propagation will take place as a protocol on layer 2.

We model each fire detector as a TelosB sensor node extended with a smoke sensor, though for this experiment we only simulate a virtual smoke sensor to make alarm generation easier. The TelosB sensor node features a CC2420 wireless transceiver [75], which supports the IEEE 802.15.4 wireless standard [32] and is capable of either sending or receiving on a single frequency at a time. It uses a synchronization header for detecting the start of packets and offers output power control, but does not provide any more advanced features such as message in message or decoding more than one transmission at a time. For all intents and purposes it fulfills the ideal of a "standard" low-power wireless interface as was referred to in Sections 2.3 and 2.4.

2.5.1 Layer 1

We design the layer 1 protocol as a parallelized convergecast on a tree overlaid onto the network connectivity graph. Every node knows of its parent and its children in the tree as well as the adjacency relationships between its child branches. The root node repeatedly initiates convergecasts and will, whenever a node is indicated as missing or defunct by the output of the convergecast, generate an appropriate alert for that node's replacement. This layer will also resynchronize a node's clock whenever it receives a message from its parent in the tree. This ensures the packet transmission synchronization requirements for achieving the capture effect hold over the time of the deployment.

The states each node goes through as it participates in the convergecast are depicted in Figure 2.6. The root node begins a convergecast by waking up itself and skipping to q_{wake} (since it has no parent). Other nodes start in q_{sleep} and wait for a request from their parent. After acknowledging the request ($q_{ackrequest}$) their parent will send some of its children proceed messages (q_{wake}), while the remaining children only receive acknowledgments and will have to wait at first ($q_{waitproceed}$). After "proceeding" a node wakes its children with a request of its own and then proceeds to q_{choose}.

Once a node has woken all its children it will start querying subsets of its children ($q_{sendproceed}$), such that the branches of no two chosen children are adjacent in the same subset. With every proceed message, every addressed child is assigned a recurring slot during which it may confirm the query (not pictured) and then later send a response, once it has gathered all the information from its branch. Once a subset of branches has completed, by each branch either sending a report or not responding for 3 subsequent

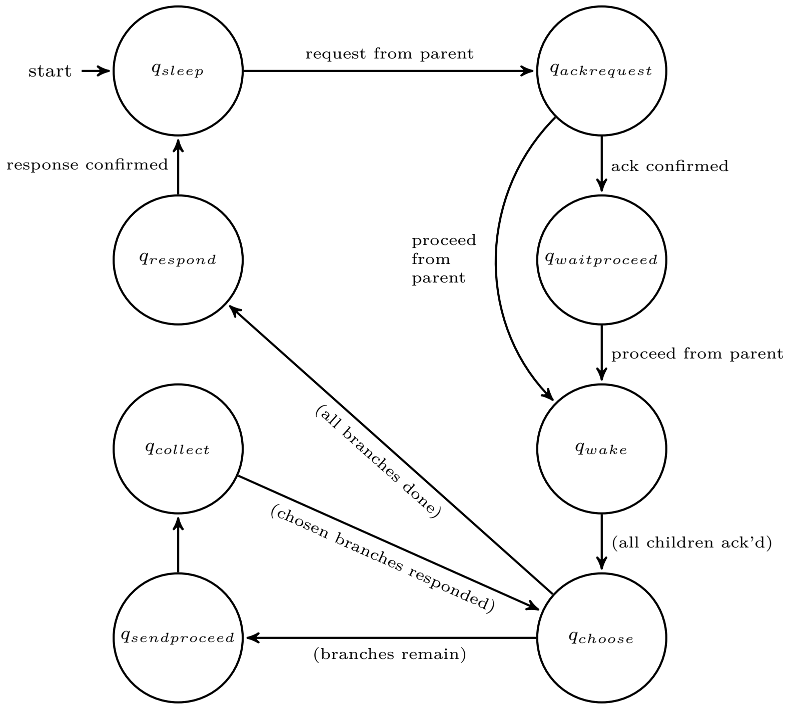


Figure 2.6: A high-level overview of the states each nodes traverses during the parallelized convergecast. Resends and root specific transitions were omitted for clarity.

queries, the process is repeated for another subset of branches. Finally, when a node has gathered all the information from its subtree it will wait for one of its designated response slots and report to its parent ($q_{respond}$).

Anytime a node needs to send messages to multiple of its children at the same time, it will combine these messages into one and send it at the highest power any of the links requires for layer 1. While this may strain the layering system in certain configurations, the effects were negligible in practice. Hence, we followed through with this approach for its efficiency and simplicity. An additional benefit is the compliance with the assumption that no node would ever send in two subsequent slots.

Further, every query and every report is explicitly acknowledged. If an acknowledgment is lost, the query or report will be resent after randomized exponentially growing backoff intervals until an acknowledgment is received or, in the case of a query, the recipient is pronounced dead after 3 attempts. Once a node is determined to be dead, it will be ignored for the remainder of the convergecast and the root node, upon receiving the aggregated node data, can notify building personnel to manually check on the potentially broken node. If a node retrying reports is queried for a new convergecast, it will snap out of its clearly erroneous state and participate as per usual in the new convergecast.

2.5.2 Layer 2

The fire alarm propagation is implemented on layer 2, i.e., a layer with a higher priority than the convergecasts described above. When an alarm event occurs it is propagated along a tree towards the root, a tree similar or even identical to the one used by layer 1. We simulate smoke alarm events occurring randomly and independently with a chance of 1% each slot. In a first test series, we trigger the event at one random node only (SA). In a separate test series we trigger multiple alarm events (MA) at different nodes simultaneously or with a few slots delay as might happen in the case of the outbreak of a real fire.

Multiple simultaneous alarms introduce an additional difficulty to the layer 2 protocol. As mentioned in Section 2.4.1, even the protocol with highest priority needs to deal with packet loss due to addressed nodes possibly not listening as they might be sending out a packet of their own on layer 1. The solution mentioned previously, to repeat all packets of the highest layer once in the subsequent slot, is not sufficient here due to the possible presence of multiple alarms, which in our setting all need to have the same priority (as we only have 2 layers available) and are hence expected to possibly collide destructively. Thus, we additionally implemented implicit and explicit acknowledgments for alarm propagation. When a node receives an alarm packet from a descendant in the tree, it forwards it in both of the next two slots and then listens. If it hears an ancestor in the tree propagate the alarm, it takes this as an implicit acknowledgment and calms down, i.e., no longer spreads the alarm. If it does not hear the alarm propagated, it repeats it another two times after a random exponentially growing backoff period, and listens again. A node which hears the same alarm again (after it had already propagated it) does not propagate it again. If the sender was a direct descendant, it sends back an explicit acknowledgment.

We also considered solving the collision of multiple alarms by tracking the received signal strength indicator (RSSI) when no packet is being received in a slot. We expected to see a signal strength similar to or stronger than that of a layer 2 packet, which would indicate that an alarm had occurred for certain, even if the exact data of the alarm was unavailable. This would allow us to pass on the existence of an alarm without incurring a latency penalty from the alarm collision. Unfortunately, experiments showed the RSSI to not be reliable enough of a measure for the presence of layer 2 packets, producing an unacceptably high rate of either false positives or false negatives.

2.5.3 Discovery

To determine the trees to be used by layers 1 and 2 as well as the different reception power levels for each layer at each node, we initially perform a "discovery" phase. The goal of this phase is to record the quality of each link in the network, compute the trees with the smallest possible height and then inform each node of its parent, its children and all the parameters required for operation as listed above.

While in our experiments we needed to perform this phase only a single time, in practice it would likely be desirable to repeat this phase every so often to deal with changes in the wireless environment, as, for instance, may easily be caused by the closing of doors or the increasing of a room's air temperature or humidity. Such repeated runs may for the most part be executed solely on layer 1 without impacting the operationally critical alarm propagation on layer 2, the potential exception being tests of links at higher sending powers. In our experience, some links' qualities can drastically change every few seconds, while others are stable for days. Reasonably, one would not perform a complete discovery as often as every few seconds or minutes, but rather on the order of hours while testing known fluctuating links more frequently.

2.6 Test Results

We compare the performance of our method to a traditional approach, which does not incorporate the capture effect but for comparability's sake adheres to the time slotting. It will, however, still execute both of the protocols (convergecast and alarm propagation) and is aware of their relative priorities. Hence, it will prefer forwarding alarm packets over sending convergecast related packets, but will use the same transmission power for all packets. Additionally, we compare the latency of the alarms as well as the durations of the convergecasts to the best physically possible values. Since these values usually are not obtainable without a dose of luck with regards to low-reliability long-range links, we do not expect these values to consistently be met.

	Alarms	**Convergecasts**
Traditional (SA)	78%	98%
Layering (SA)	100%	88%
Traditional (MA)	59%	98%
Layering (MA)	79%	88%
Traditional (MA with Acks)	100%	75%
Layering (MA with Acks)	100%	88%

Table 2.2: Experiment A: Percentages of successful alarms and convergecasts.

	Alarms	**Convergecasts**
Traditional (SA)	79%	98%
Layering (SA)	98%	87%
Traditional (MA)	50%	97%
Layering (MA)	65%	53%
Traditional (MA with Acks)	82%	66%
Layering (MA with Acks)	98%	50%

Table 2.3: Experiment B: Percentages of successful alarms and convergecasts.

In the first test series we consider the described algorithm without alarm acknowledgments (and thus without resends) in the scenario of single alarms (SA) only. We do, however, still repeat every alarm propagation packet in the subsequent slot to avoid losing it to a layer 1 packet being sent at the destination node. For the second and third test series we tested the algorithm with and without acknowledgements in the scenario of multiple alarms (MA). We present the results for two representative experiment runs with different topologies. Experiment A used 11 nodes and executed 462 convergecasts and 231 alarms. Experiment B used 14 nodes and focused on an increased alarm density by doubling the probability for an alarm to occur in each slot to 2%, executing 311 convergecasts and 528 alarms. For both experiments, the convergecasts and alarms were divided evenly among the 3 test series and 2 approaches. The experiments took approximately 50 minutes each.

Tables 2.2 and 2.3 list the portions of alarms and convergecasts which were successful in each setting. An alarm is successful if it did not get lost, i.e., reached the root node eventually. A convergecast is successful if it completed correctly and collected data from every single node. The tendency of the layered approach to promote alarms over convergecasts even more than the traditional approach is clearly visible: while our layering approach generally suffers from fewer successful convergecasts, it beats the successful alarm ratio of the traditional approach, reaching an almost certain alarm delivery. The difficulty of dealing with multiple alarms without acknowledgements is also apparent, as alarms inevitably get lost. Of special note is the fact that while the traditional approach reaches the same 100% alarm successes as us using acknowledgements in experiment A, it suffers a larger convergecast success penalty.

Figures 2.7 and 2.8 show the CDF (cumulative distribution function) for the alarm delay. The delay of an alarm is defined as the number of slots it takes to reach the root node minus the physical minimum number of slots required to traverse the multi-hop path from its origin to the root. This allows for a comparison between alarms originating at different nodes. We observe our approach beating the traditional one in each category, not least because it suffers fewer lost alarms, with the exception of "MA with acknowledgements" in experiment A. For single alarms we achieve an alarm delay of 2 slots or less in 85% resp. 94% of cases, which is excellent considering the optimal reference alarm delay taking unreliable long-distance links into account. As was to be expected, overall the maximum delay experienced without acknowledgements is around 4 slots, while acknowledgements allows alarm reporting to be drawn out considerably.

Figures 2.9 and 2.10 shows a similar CDF for convergecast delays, defined as a convergecast's duration minus the minimum amount of slots our algorithm requires for the convergecast even if not a single packet was lost. The considerably worse performance of our approach here is due to it causing the layer 1 algorithm increased packet loss in order to support layer 2. Also of note is the general increase in delay for both approaches as more layer 2 traffic is introduced, both by adding acknowledgements and resends and by increasing the amount of alarms.

2.7 Summary and Future Work

We presented a method to execute multiple protocols in parallel, giving each protocol the illusion of being the only one and having complete access to the network's resources. When a higher-priority protocol uses network resources, such as the ability of a node to send or receive a packet in a specific slot, lower-priority protocols experience this as packet loss, as sending priority or the capture effect drop lower-priority packets.

Further, in theory our method causes no overhead in terms of time slot

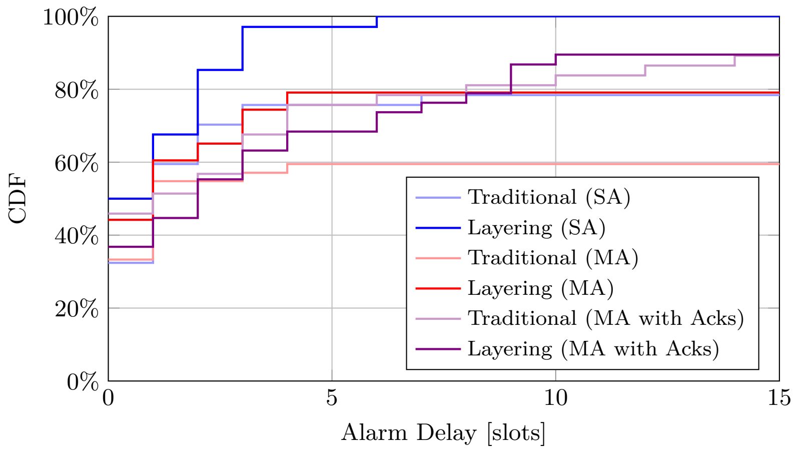


Figure 2.7: Experiment A: Distributions of alarm delay.

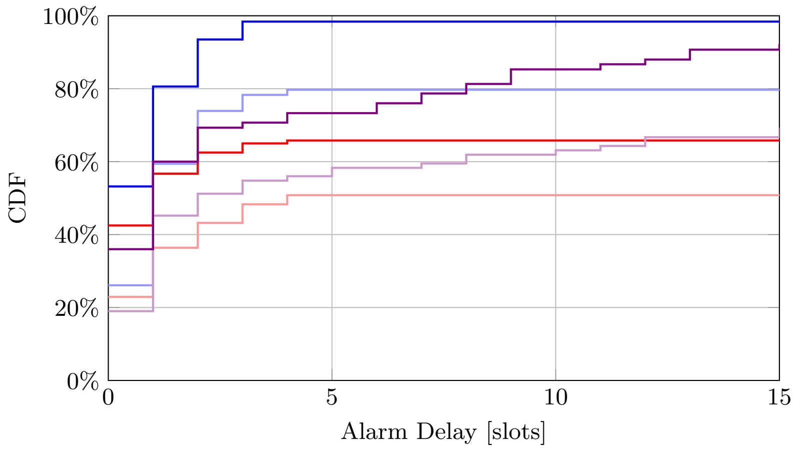


Figure 2.8: Experiment B: Distributions of alarm delay.

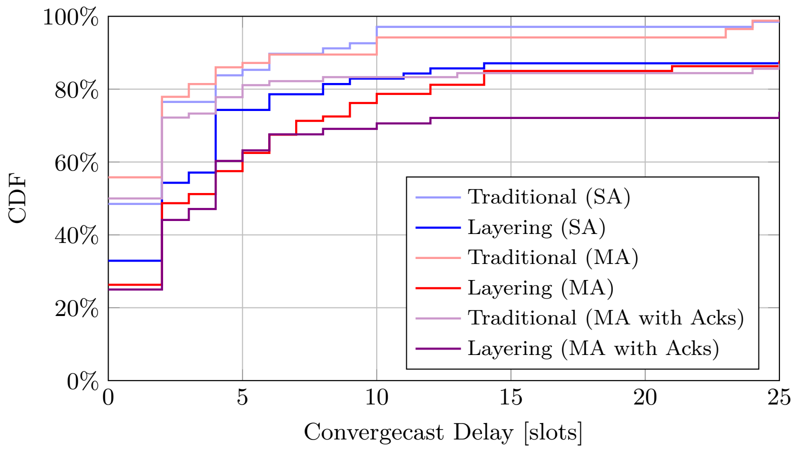


Figure 2.9: Experiment A: Distributions of convergecast delay.

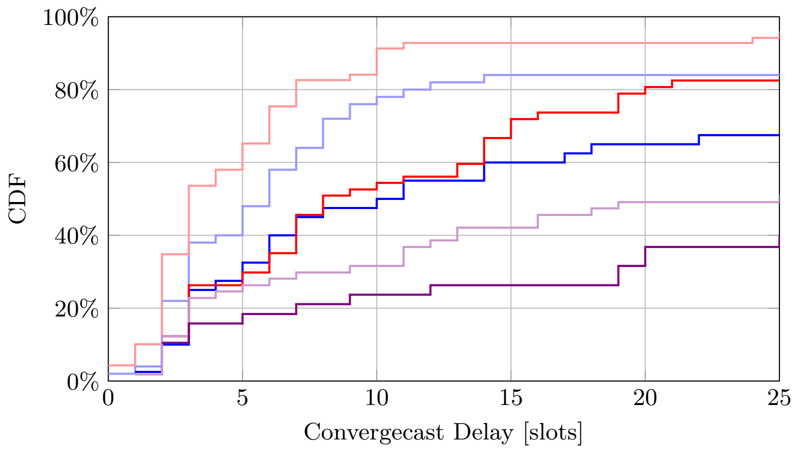


Figure 2.10: Experiment B: Distributions of convergecast delay.

use and latency. To confirm this, we implemented an example application with 2 protocols of different priorities and measured their performance. The results show very few packet losses and essentially optimal latency for the higher-priority protocol, while it causes additional losses to the lower-priority protocol compared to a traditional approach. In the scenario that both approaches can avoid loss of high-priority packets, our method does so while incurring less overhead to the lower-priority protocol.

Our technique does come with some downsides, most notably the removal of the individual protocols' ability to employ some options directly related to the physical layer. These options include power control, knowledge about corrupted packets, the use of the capture effect and the ability to not use time slotting. For many algorithms these features may be non-critical or even completely irrelevant. For others it may make this technique unsuitable or require significant changes. Further, the connectivity graph is likely to lose some edges on one or more layers whose target reception power ranges cannot be met by the respective senders' transmission power ranges. This especially affects the availability of long-range links on the lower-priority layers.

Our technique proves to be flexible concerning hardware and existing protocols, allowing it to be deployed without much additional overhead beyond implementing the discovery phase and basic slot logic. We have shown simple low-power hardware such as TelosB to be sufficient and pointed out how our technique is capable of masking the existence of competing protocols, allowing many protocols to simply be "dropped in" on one of the layers, possibly even on-the-fly. Some protocols may greatly benefit from being aware of the other protocols, for instance, by knowing when their packet was dropped already at the sender, or by even directly communicating with the higher layer protocols on the same node. We believe this work to nevertheless be a useful first step in the direction of exploring such multi-layer protocols.

Incidentally, our technique is also a feasible fairness provider in many situations. While in its basic implementation it will in fact allow any protocol to completely starve all protocols with a lower priority, as discussed in Section 2.4.2, by periodically reassigning the layers to different protocols, each protocol can be assigned an arbitrary share of the network resources when averaged over longer periods of time. However, our approach is likely not suitable for applications with a need for temporally more fine-grained QoS. This is under the assumption that under load each protocol is able to work more efficiently if it is the top active layer for its critical region for longer consecutive time intervals than for many smaller ones, separated by spurs of time with possibly 100% packet loss.

Power consumption was not examined as part of our tests. We believe this issue to be almost orthogonal to the problems discussed here, as duty cycling and rate of packets sent are not affected by the proposed method. If nodes are unable to maintain synchronization through a sleep phase, care needs to be taken that they are resynchronized before attempting to send a packet intended to evoke the capture effect. On TelosB hardware, however, staying below the synchronization error threshold of 160 µs for longer periods of time is very easy due to its 32 kHz crystal oscillator (with a tick length of about 31 µs) being able to power-efficiently and accurately keep time even when the CPU is in sleep mode. While operation on higher-priority layers does require larger transmission powers, for many applications the proposed method may remain a desirable choice, especially if high-priority messages are more of an exception than a rule (as in our fire alarm example).

Promising future work includes applying this method to wireless hardware with a finer power control granularity, especially at the low power end. We expect such hardware to make it significantly easier to support a larger number of layers while still having a clear separation of layers. In general, it is also worth investigating the capture effect parameters for a layering setup on other hardware, as hardware more amenable to exhibiting the capture effect may loosen the requirements on synchronization or layer separation.

If the lack of absolute network resource control for higher layers proves to be an issue (i.e., if lower layers are preventing reception of high layer packets by sending lower layer packets), various solutions may be worth exploring. For example, one could depart a little from strict slotting and have higher layers send their packets a few dozen symbol durations earlier, as to allow prevention of lower layer sending. Alternatively, a mechanism by which higher layers completely reserve some nodes for a period of time is imaginable and may be very effective depending on the application.

3

Maintaining Constructive Interference

3.1 Introduction

In recent years, the benefits of constructive interference (CI) have been discovered for wireless sensor networks: By synchronizing identical signals well enough, the interference caused by the collision is not destructive but rather unnoticeable or even strengthening. On the one hand, this allows the development of algorithms which can expect identical packets not to destructively interfere. On the other hand, more unstable and longer links can be used more reliably through the strengthening of the signal.

Traditionally, the frugal nature of sensor node hardware and timekeeping moved synchronizing clocks and transmissions out of reach for commodity sensor nodes. In the past, several workarounds have been proposed to nevertheless enable the signals of separate nodes to interact meaningfully:

- One way is to provide a highly accurate external global clock source by some means, e.g., GPS, with each sensor node.
- Another approach is to forgo the ability to send fully articulated data packets – and to simply transmit waveforms instead, which are unlikely to fatally destructively interfere [20].

- Most recently, Glossy [19] explored how to leverage incoming reference packets to send a packet immediately afterwards with an almost constant delay. When multiple nodes use the same reference packet (or different well-synchronized reference packets), their outgoing packets are then synchronized enough to achieve CI.

All of these approaches have in common that they try to avoid dealing with the rather unreliable internal clocks of commodity sensor nodes. In this chapter, we explore the possibilities of creating synchrony sufficient for harnessing CI without relying on any of these workarounds. By meticulously keeping time using the available local clocks to extrapolate a global time value, and by minimizing the variance in packet departure delay, we manage to at least partially obtain this goal. In particular, we present a proof of concept implementation achieving CI on the popular TelosB sensor nodes, confirming the viability of our approach: Even for only 2 senders we obtain an increase in signal strength of at least 2 dB in 60% of cases and an increase in packet reception ratio of 20–35% when signal strengths are equal. For ideal results, we need the last synchronization of the senders to lie at most 20 seconds in the past when attempting to send simultaneously, but even after multiple minutes of sleep our approach can benefit from CI. We carefully compare several parameters in order to understand the limits of CI on sensor nodes. For instance, already a synchronization error in the order of 1 µs is problematic. More precisely, as we are using the IEEE 802.15.4 standard, the transmissions need to start within less than 0.5 µs of each other to avoid destructive interference.

Hence, a significant part of this chapter is dedicated to synchronizing the clocks of the sensor nodes, maintaining this synchrony and, finally, precisely dictating the time span of the wireless transmission. Clock synchronization for wireless nodes in general is a well-studied problem, but the synchronization precision requirements by our setup are more stringent than those of the common use cases such as TDMA.

3.2 Related Work

3.2.1 Clock Synchronization

Time synchronization between network nodes has been studied for a long time even before wireless nodes became widespread. The 30 years old Network Time Protocol (NTP) [55] is still in use today for time synchronization between two machines over the Internet, achieving accuracies of about 10 ms. Recently, adjustments have been proposed improving its accuracy to about 1 millisecond [43], which is still off by a factor of over 1000 from the precision we require.

Wireless networks face additional challenges [16] but also have additional options at their disposal due to the different nature of their medium. Römer [67] proposes a synchronization algorithm focusing on sparse ad hoc networks. Dozer [3] minimizes energy consumption by keeping sender-recipient pairs synchronized over long periods of time and only rarely waking them for brief scheduled rendezvous exchanges. To maintain synchrony in spite of clock drift, the drift is modeled and compensated for. We employ a similar scheme, see Section 3.5.3.

RBS [15] makes use of the broadcasting nature of every wireless transmission to synchronize multiple nodes with a single transmission. RBS achieves synchronization accuracy on the order of microseconds. TPSN [21] builds a hierarchical tree structure for synchronization from a root node. TPSN also introduces the concept of MAC layer timestamping, which we build upon in this work (see Section 3.5.2). FTSP [53, 54] floods periodical synchronization waves through the network, improving accuracy to around 1 µs. Further improvements for synchronizing networks of nodes are RITS [68], reactively synchronizing nodes using global events, and RATS [37] and PulseSync [41], which propose rapid network-wide flooding while employing schedules to avoid collisions.

An analysis of the single-hop and multi-hop performance of the different synchronization protocols and a proposal for a non-hierarchical synchronization method improving the multi-hop case can be found in [31].

3.2.2 Constructive Interference

Already early on, primarily flooding algorithms became privy to the possible gains of using non-interfering signals and thus being able to flood the network without the overhead of building and adhering to a flooding schedule. We distinguish single and multiple source flooding: while in single source flooding a node desires to disseminate information or an event in the whole network, in multiple source flooding any number of nodes may raise redundant alarm events which need to be propagated. Lu et al. [49] proposed the single source flooding protocol *Flash*, which relies on the capture effect to resolve collisions between neighboring nodes at the flooding front.

Slotos [20] implemented multiple source flooding in form of an alarm system in which an alarm signal, a simple waveform, was propagated through a network by nodes which received the alarm starting to send out the signal as well. In this case the concurrent signals were not artificially aligned but the resulting amount of destructive interference was acceptable to the protocol, as the signal did not carry any data beyond its existence. Similarly, *Black Burst Synchronization* (BBS) [23] employs non-destructively interfering pulses, so called “black bursts”. Dutta et al. [13] discovered that automatic acknowledgments of multiple nodes having received the same packet will in fact be synchronized so well as to cause constructive interference. Automatic acknowledgments are a common wireless transceiver feature that allows an immediate response to correctly received packets (as verified by the packet’s checksum). As the acknowledgment is completely handled by the hardware, this feature avoids any variable desynchronizing delays the microprocessor software may cause. While essentially all of these approaches have the common disadvantage of having very little control over the data being sent, this is well suited for multiple source flooding, as such events typically are not expected to carry detailed information.

More recently, Ferrari et al. [19] took the idea of synchronizing to the end of a previously received packet one step further and proposed *Glossy*: instead of sending an acknowledgment, which does not allow for the transmission of meaningful data, they prepare a full response packet during the reception of the incoming packet. By immediately issuing its transmission when the transceiver indicates the completion of the reception of the incoming packet, the variable delays microprocessor software code paths and clock skew can introduce are minimized. As the packet contents sent by the different responders still have to be the same to allow constructive interference, flooding is the main application for this approach as well. The constructive interference leads to improved signal strengths, making weaker and longer links more viable. Further, the rapid forwarding leads to exceptionally short flooding durations. Several recent protocols are based on the rapid data dissemination capability of Glossy [11,18,38,79] and various studies on its reliability and the conditions necessary for creating constructive interference have been conducted [78, 80].

To achieve CI, Glossy relies on tying the departure time of each packet to the end of the incoming transmission of a previous packet. One of the main goals of this chapter is to do away with this restriction, yet still attain CI for data packets rather than be restricted to mere events.

3.3 Experiment Setup

3.3.1 Hardware

The TelosB sensor node [62] we use for our experiments is mainly comprised of a 16-bit RISC microcontroller, the TI MSP430, and a TI CC2420 wireless transceiver. The clocks available to the MSP430 are a 32 kHz (2^{15} Hz) quartz crystal as well as an internal digitally controlled oscillator (DCO) which serves as the pulse generator for the instruction execution. The frequency of the DCO is very prone to being skewed through fluctuations in temperature and voltage, but can generally be configured to be anywhere in the range from 100 kHz to 5 MHz.

The CC2420 is tailored to support IEEE 802.15.4 and offers many convenience features such as framing given packet data including automatically including a 16-bit CRC checksum field in the packet footer. We rely on this checksum mechanism as our main way to tell whether a received packet has been corrupted. The CC2420 offers a range of transmit powers from -30 dBm to 0 dBm (see Figure 2.4). We make use of this feature as it is easier to adjust the output power than to physically move the nodes every time changes in the environmental conditions require it (see Section 3.7).

We made use of an installation of 30 of these sensor nodes part of FlockLab, placed at various locations on a floor of an office building near the ceiling, some inside enclosed offices, some on the hallway. This allowed us to consider a variety of real life scenarios.

3.3.2 Software

On the sensor nodes we run code based on Contiki [12] version 2.7. We chose Contiki because it allows making the kind of low-level modifications we required the easiest. In particular, we overhauled the clock module which is concerned with using the MSP430's Timer_A (configured with the 32 kHz quartz crystal as clock source) to count elapsed seconds and periodically test for expired user-defined timers. We decrease the time span between timer interrupts to 2^{-10} seconds, and instead of counting elapsed seconds we count the occurrences of overflow interrupts for the Timer_A timer register.

Contiki also offers a mechanism to periodically re-configure the DCO to best match the desired frequency. We make use of this to keep the DCO frequency near 2^{22} Hz (roughly 4 MHz), the highest supported power-of-two multiple of the quartz crystal frequency (2^{15} Hz). This essentially subdivides each tick of the quartz crystal into 2^7 sub-ticks, i.e., increasing the combined precision by 7 bit (see Section 3.5.1).

The DCO is usually turned off to conserve energy during idle low-power phases of the processor. While our approach allows such sleep phases, we naturally do require the DCO to have been running since at least the last quartz crystal tick to be able to provide accurate timestamps. This incurs an overhead of up to 2^{-15} seconds of DCO operation after each wake-up.

We also configure the CC2420 driver to not use CCA (clear channel assessment) for packet transmission and extend it to support recording 64-bit timestamps on incoming packets and adding 64-bit timestamps to outgoing packets (see Section 3.5.2).

For the process of the experiments themselves see Section 3.7.1.

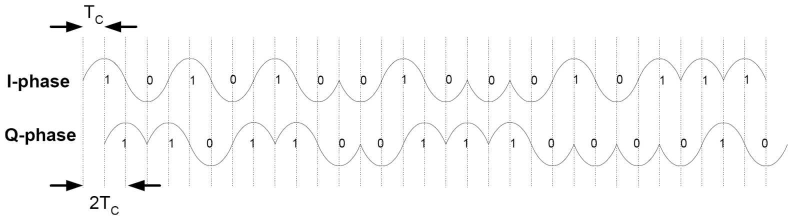


Figure 3.1: A figure from the CC2420 Manual [75] illustrating the pseudo-random modulation pattern (chip sequence) of the '0' symbol on the I and Q phases. T_C is defined to be 0.5 µs, i.e., the chip rate is 1 Chip/T_C = 2 MChips/s.

3.4 Timing Requirements

The IEEE 802.15.4 standard [32] specifies the encoding of the raw logical data stream to electromagnetic signal as follows: First, the data is segmented into groups of 4 bits, called "symbols". Each of the 16 possible symbols is then mapped to a certain pseudo-random noise sequence of 32 binary "chips". Finally, the chip sequences are concatenated and modulated using O-QPSK, i.e., every chip is modulated alternatingly onto the I and Q phases, offset by half a chip duration. At a chip rate of 2 MChips/s this pans out to 62,500 symbols per second or a 250 kbit/s data rate. See Figure 3.1 for an example modulation of the zero symbol (corresponding to 4 zero bits).

In the ideal case all the senders' (identical) chip sequences arrive at the recipient node at the same time overlapping each other at a random (carrier) phase offset, likely causing CI and thus increasing the signal strength. Unfortunately, such alignment does not come easily, and in the remainder of this section we will discuss the different sources of signal misalignment.

As an upper limit for the acceptable signal shift between two signals we can identify $T_C = 0.5\,\mu\text{s}$, half the duration of a single chip within one of the two phases. If the signal shift is any greater, every chip of the second signal will superimpose the chip subsequent to the corresponding chip of the first signal more than the actually corresponding chip. This upper limit of 0.5 µs is certainly not tight as that is merely the point at which chips certainly cannot be unambiguously matched to a signal anymore. In practice, we are aiming for a signal shift of 0.2 µs or lower to ensure a strong CI effect. When using a set of more than 2 senders, we aim to minimize the shift between every pair of signals.

We identify 3 main sources of signal misalignment between the signals of two senders A and B: the clock synchronization error e_{clock}, the transmission timing error $e_{transmit}$, and the difference in the signal travel times

e_{travel}:

$$e_{total} = e_{clock} + e_{transmit} + e_{travel} \overset{!}{\ll} 0.5\,\mu\text{s}$$

Each of the components can be positive or negative and thus they can cancel each other out. They are defined as follows.

The clock synchronization error is simply the difference in the senders' views of the global clock value. We discuss reducing this error in detail in Section 3.5.

$$e_{clock} = clock_A - clock_B$$

The transmission timing error of a node is the delay the node starts the transmission after a given desired local time. The difference in transmission timing errors (delays) at each node forms $e_{transmit}$. We discuss reducing this error in detail in Section 3.6.

$$e_{transmit} = e_{transmitB} - e_{transmitA}$$

Finally, the travel time error is the difference between each sender's signal's travel time.

$$e_{travel} = traveltime_B - traveltime_A$$

The travel time error is of note because at the precision of time we are working with here – tens of nanoseconds – the travel time of the electromagnetic waves becomes significant, even in a testbed located on a medium-sized office building floor. For instance, if one of the senders is 30 meters further away from the receiver than the other the difference in signal travel time is 100 nanoseconds.

We tried measuring these travel times by measuring the round-trip times of links and subtracting the time the responding node measured between the arrival of the incoming and the departure of the outgoing packet. Two factors thwarted this attempt: Firstly, the CC2420 transceiver we are using experiences a "data latency", which denotes the time between the sender and the receiver activating their SFD ("start of frame delimiter") pin, which indicates the start of an incoming or outgoing transmission. This delay is apparently caused by the processing of the non-packet signal in the receiver and specified to be 3 µs by the CC2420's data sheet. Our measurements show that this delay is probably closer to 3.6 µs and that it can fluctuate by several dozen nanoseconds in subsequent measurements. Secondly, we observed the round-trip time to vary by hundreds of nanoseconds over the course of a day. This is likely caused by environmental effects, such as temperature influencing the wireless transceiver's clock and the opening and closing of doors in the office building changing the actual travel times, although we doubt that hundreds of nanoseconds of difference can be explained by travel

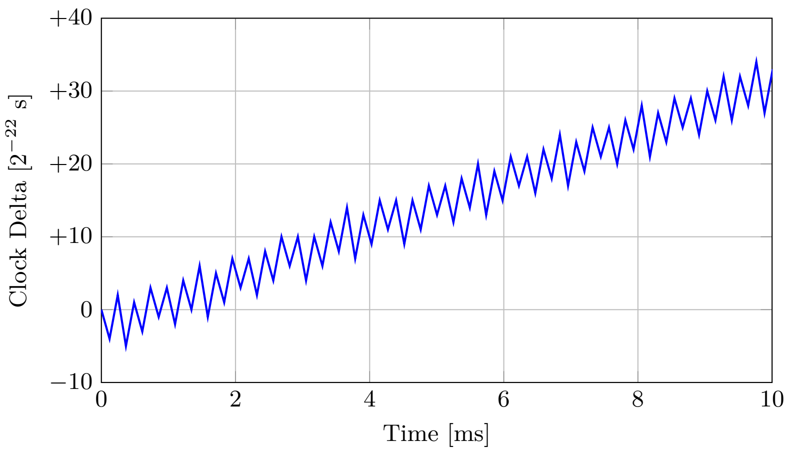


Figure 3.2: The delta between the quartz crystal clock and the DCO clock over the span of 10 ms, sampled at every fourth tick of the quartz crystal.

time changes. In the end, we did not come to a conclusive explanation and decided to simply make best effort to eliminate this error: we pick senders at roughly equal distance to the recipient node and adjust for the "data latency" with a constant time offset of 3.6 µs.

3.5 Clock Synchronization

3.5.1 Merged Clocks

In general, quartz crystals exhibit a more stable and thus more desirable behavior than digitally controlled oscillators (DCOs), both in terms of long term drift as well as short term frequency stability. The TelosB possesses 2 quartz crystals: a 32768 Hz one attached to the MSP430 for general high-accuracy timekeeping, and a 16 MHz one attached to the CC2420 transceiver used for signal processing and as data rate reference. Unfortunately, the latter is not accessible to the CPU, so we need to make do with the MSP430's built-in DCO as sole source for high granularity clock ticks: It offers a resolution of about 0.24 µs, compared to the resolution of the 32 kHz quartz crystal of about 30 µs.

To get the best from both worlds – the stability of the quartz crystal and the precision of the DCO – we build a hybrid clock, similar in spirit to the method proposed by Schmid et al. [70]. The 2^{15} Hz quartz crystal has

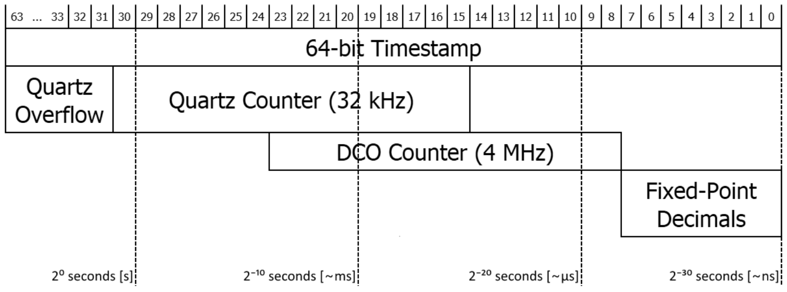


Figure 3.3: The final layout of our timestamps with the different clock sources in different rows. The quartz clock has priority over the DCO clock, so actually only 7 bits of the DCO clock are used. No clock we use is able to specify the lowest 8 bits, but these bits allow for storing extra precision when a timestamp is the result of arithmetic operations.

priority and ensures relatively high stability over the long term while the DCO, set to a speed as close to 2^{22} Hz as possible, fills out the time span in between the quartz crystal ticks with $2^{22}/2^{15} = 2^7$ finer-grained ticks. To accomplish this, we configured the MSP430 to copy the current value of the DCO clock (a 16-bit counter) at each quartz crystal tick into a special capture register. The MSP430 allows doing this in parallel with regular instruction execution at no additional overhead. A complete current time value could then be obtained by subtracting the current DCO counter from the value in the special capture register and using the result as 7 bits of additional precision together with the current quartz crystal counter. Additionally, we keep the number of times the quartz crystal counter overflowed in a variable. This variable is incremented in an overflow interrupt handler causing a negligible overhead every 2 seconds. Figure 3.2 shows how a separate DCO clock would move in relation to the quartz crystal. Not only is there a clear drift and oscillating behavior (both of which could be accounted for to some degree), but there also is a notable amount of randomness that would make relying on the DCO alone as a clock source a poor choice.

As all the MSP430's registers only hold 16 bits, the question of the size for timestamps arises, as for the transmission between nodes they should be unambiguous, at least within a period of several seconds. 16-bit variables do not suffice as the DCO clock alone wraps around 256 times a second. 32-bit variables would only wrap every 1024 seconds or roughly 17 minutes, but since we want to enhance our timestamps by an additional 8 bits of precision when extrapolating (see Section 3.5.3), we settled on 64-bit variables. Such variables are well supported by our compiler (a GCC variant for the MSP430), but special care has to be taken when performing arithmetic operations on variables of this size, as they quickly grow to require hundreds of instructions on 16-bit RISC CPUs such as the MSP430. The final layout of our timestamps can be seen in Figure 3.3. The precision of these timestamps is 2^{-30} seconds, roughly a nanosecond, and the range is sufficiently large (several years).

3.5.2 MAC Layer Timestamping

To synchronize the sending nodes' clocks, a reference node sends a couple of synchronization packets at the start of each round (see Section 3.7.1). To transmit synchronization information we employ MAC layer timestamping [21]. This technique generally involves capturing the local time at the start of an arriving or departing packet using an specialized output pin of the wireless transceiver (the aforementioned "SFD pin") connected to a timer capture input pin of the CPU. The sender writes the captured timestamp to the footer of the outgoing packet while the transmission of the of the packet is ongoing. The receiver in turn can then compare its captured timestamp to the timestamp contained in the footer to obtain an accurate local time value for the shared event of the start of the packet.

Contiki's CC2420 driver already provides this feature, but only supports 16-bit timestamps from the 32 kHz quartz crystal. We extend the driver to support our mixed-source 64-bit timestamps. As mentioned above, assembling such a 64-bit timestamp from its individual components requires a fair amount of instructions on this architecture – a total of 55 in our case. Further, we also already apply drift compensation at this stage, which requires the multiplication of two 64-bit integer variables, which result in an additional few hundred instructions. As our synchronization packets are 60 bytes in length, which require roughly 2 ms or 8500 CPU cycles to transmit, there is still ample time to compute this timestamp and insert it at the end of the packet.

3.5.3 Drift Compensation

Although the properties of quartz crystals are generally rock-solid, their frequencies are not completely set in stone: temperature changes affect their frequency and multiple instances of the same crystal may have slightly different frequencies. In a system with multiple such quartz crystals this leads to observable clock drift: clocks which were synchronized at some point in time may drift apart as a result of their ever so slightly varying speeds. Worse, these speeds change over time as the environment temperature changes.

To combat this, we implemented a drift compensation mechanism based on linear regression with a rolling buffer: For the last k received synchronization packets we store $(local_i, offset_i)$ pairs, where $local_i$ is the local time at which the ith synchronization packet was received and $offset_i = global_i - local_i$ was the clock offset at that time. We then compute on the last k values:

$$drift = \frac{\overline{local_i \cdot offset_i} - \overline{local_i} \cdot \overline{offset_i}}{\overline{local_i^2} - \overline{local_i}^2}$$

$$baseoffset = \overline{offset_i} - drift \cdot \overline{local_i}$$

where $\overline{\exp}$ denotes the average of the expression exp for $i \in \{n, n-1, n-2, \ldots, n-k+1\}$ and n denotes the number of received synchronization packets. To extrapolate an estimate of the global time we compute:

$$globalest(local) = local + baseoffset + drift \cdot local.$$

For the size of the rolling buffer we found $k = 8$ to be adequate, adapting to changes quickly enough, while avoiding wild fluctuations from noise and outliers. Other settings with less frequent synchronization rounds might find smaller values to suit their needs better.

While drift compensation is absolutely crucial in situations where nodes are not re-synchronized for longer periods of time, it can already offer substantial benefits after shorter periods when high precision is required. To be able to store and forward global timestamps computed using the formulas above while preserving the gained precision, we append 8 additional bits to our timestamps (see Figure 3.3).

Figure 3.4 shows the quality of the clock synchronization that we were able to achieve using both MAC layer timestamping and drift compensation. We observe most instances (over 70%) to be spread uniformly between -250 and $+250$ nanoseconds. We cannot hope to improve this by much since our clocks are only granular to 2^{-22} seconds. Hence, each clock reading introduces an error distributed uniformly at random in the interval $[-2^{-23}$ seconds, $+2^{-23}$ seconds$]$ (roughly $[-120$ ns, $+120$ ns$]$).

Figure 3.5 shows the development of the synchronization error between two nodes which were synchronized a couple of times at the start (to allow for drift detection) and then not anymore. Clearly, not employing drift compensation when not re-synchronizing nodes over longer periods of time is fatal to the synchronization error. The almost constant slope shows why linear regression is the right tool to combat clock drift. We also see in this example that, in spite of drift compensation, the synchronization error exceeds 0.5 µs after about 25 seconds. Hence, when attempting simultaneous sending, the most recent synchronization should ideally not lie further than 10 or 20 seconds in the past.

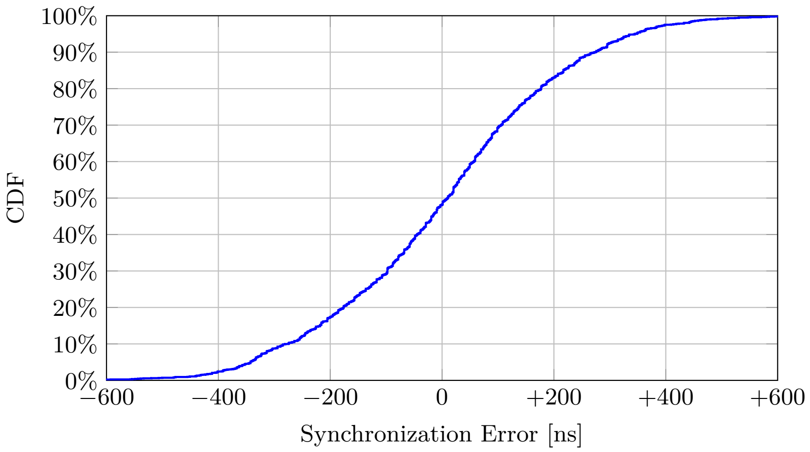
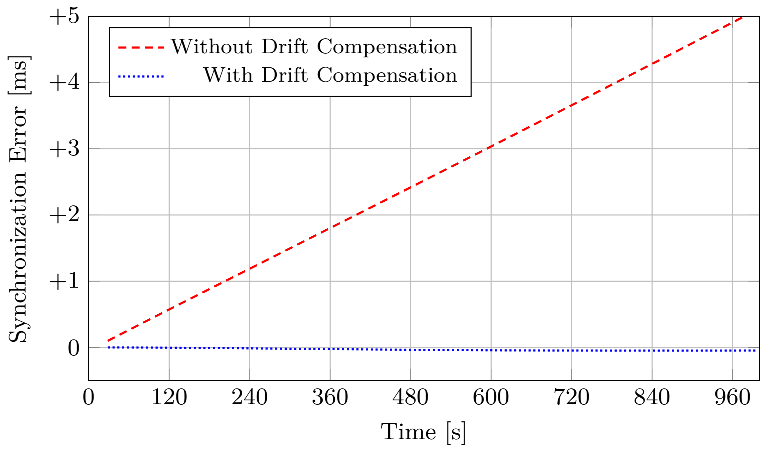


Figure 3.4: The distribution of the clock synchronization error a single node is detecting while being re-synchronized every few seconds. For this plot the sample size was 1934.

In conclusion, we solved the problem of single-hop clock synchronization on TelosB as well as possible but still have to admit an error e_{clock} of up to 250 nanoseconds. However, as the error is almost uniformly distributed, we will hit cases as good as $|e_{clock}| \leq 50$ nanoseconds over 20% of the time.

3.6 Transmission Synchronization

Even if our clock was perfectly synchronized to the global clock, transmitting a packet exactly at some given time is not a simple task. Once the transmission of a packet has begun, its synchronization error will no longer change, since its transmission is now controlled by the CC2420's 16 MHz quartz crystal. However, the delay between reaching the desired departure time and beginning the transmission is not necessarily constant. It can be split into two parts: the time between reaching the desired departure time and issuing the STXON command strobe (requesting the transmission of a previously loaded packet) to the transceiver, and the time between issuing STXON and the actual start of the packet transmission. We measured the latter of these two by comparing the DCO clock values right before issuing the STXON command and at the SFD event (see Section 3.5.2). The distribution of the measured values are shown in Table 3.1. The fluctuations are well within the limits of the frequency noise the DCO experiences (see



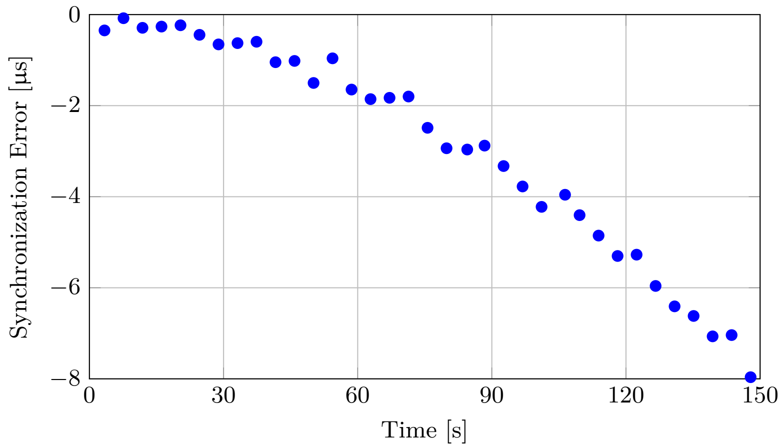


Figure 3.5: Synchronization error of a single node over time.

$t_{\text{SFD}} - t_{\text{STXON}}$	**Instance Count**	**Fraction of Total**
1593	15	0.26%
1594	893	15.71%
1595	3726	65.53%
1596	1049	18.45%
1597	3	0.05%
Σ	5686	100.00%

Table 3.1: Measured STXON→SFD times collected from a single test run with 15 nodes. The time differences are specified in DCO ticks.

Figure 3.2). While it is possible that there is a small variable delay, it is not discernible and we assume this time span to be constant.

It remains to scrutinize the time span between reaching the target time and issuing STXON. Initially, we employed busy waiting: once the target time was close, we stop relinquishing control to the operating system and enter a loop in which all we do is query the time until the target time has been passed. Immediately after, we call the driver routine to start the transmission (one of the first instructions of which is to issue STXON).

Listing 3.1: Simple Busy Wait

```
if (TargetTime - GetGlobalTime() < 10 ms) {
    while (TargetTime > GetGlobalTime())
        ; // do nothing
    cc2420_driver.transmit();
}
```

This approach experiences a large variance in loop exit times and thus transmission times (relative to the target time) due to the large amount of instructions within the loop: assembling the local timestamp requires 55 instructions, computing the global time from it requires at least 110 instructions and comparing it to the target time requires 22 instructions. To address this problem we made 3 changes. The first change was to apply our model of the global time “backwards” to compute the target local time before entering the busy wait loop. The second change was to decompose the target local time into its 3 clock sources (see Figure 3.3) and spin on the 3 16-bit clocks one after another. These two changes reduce each loop to the minimum of 4 instructions. See Listing 3.2 for an approximate implementation. `TAR` is the quartz counter register, `TBR` is the DCO counter register and `TBCCR6` contains the value of `TBR` at the last quartz crystal tick.

Listing 3.2: Busy Wait Split by Clock Sources

```
void await(uint64_t local_target) {
    uint16_t target_tarof = (local_target >> 23) & 0xFFFF;

    while (TAR_overflows < target_tarof)
        ;

    uint16_t target_tar   = (local_target >>  7) & 0xFFFF;
    while ((TAR - target_tar) & 0x8000)
        ;

    uint16_t target_tbr = (local_target & 0x007F) + TBCCR6;
    while ((TBR - target_tbr) & 0x8000)
        ;
}
```

It is worth noting that not every instruction requires the same amount of time to be executed. However, due to the absence of caches and pipelining in the MSP430's architecture, the execution time of any given instruction can be specified exactly in terms of a number of CPU cycles, which directly correspond to the impulses generated by the DCO.

Listing 3.3: Busy Wait Assembly

```
.L36:
    mov &__TBR, r15 ; 3 cycles
    sub r12, r15    ; 1 cycle
    cmp #0, r15     ; 2 cycles
    jl .L36         ; 2 cycles
```

The third and final change we made was to insert NOPs (1-cycle "no operation" instructions) ahead of the final loop to ensure we exactly matched the target time when exiting the final loop. The four instructions comprising the loop execute in exactly 8 cycles (see Listing 3.3), so before the loop we wait for `(target_tbr - TBR)` mod 8 cycles by using a jump table into a series of NOPs. We note that we could save 2 cycles by merging the `cmp` and `jl` instructions into a single `jn` instruction (2 cycles), an optimization which our compiler did not apply. However, such a change would affect the achieved synchronization only in extremely rare cases.

After applying these optimizations, the time offset between the target time and the issuing of the STXON command strobe was constant in over 80% of cases. By artificially delaying the next periodic timer interrupt before entering the last busy-wait loop to a point 1 ms in the future, this number increased to $> 99\%$ of cases.

The constant delay is easily adjusted for, leaving the only remaining transmission error we can observe the delay between issuing STXON and registering the SFD event as measured by the DCO clock. Note that in this case the DCO clock is used only for measuring and does not in any way influence the wireless transceiver preparing for the transmission. Thus, it is a reasonable assumption that the jitter seen in Table 3.1 is merely a product of the DCO's instability, albeit we cannot rule out that the transceiver itself introduces a small variable delay.

Finally, as with clock synchronization, we suffer a transmission error $e_{transmit}$ of up to 250 nanoseconds, the resolution of our most precise clock, independent of the clock synchronization error. However, this error is also distributed uniformly in the interval $[-2^{-23}$ seconds, $+2^{-23}$ seconds$]$ and in a certain fraction of all samples $|e_{transmit}|$ will be acceptably small.

3.7 Constructive Interference

3.7.1 Experiment Procedure

A particular concern when designing the experiments was to ensure we would be able to discern which transmissions were successful due to CI and which were successful due to the capture effect. For the capture effect to occur, the "captured" transmission needs to be slightly stronger than the sum of the remaining transmissions' signal powers. On our testbed we found the required extra signal power to vary between 2 or 3 dB, highly depending on the nodes (see Figure 3.6). As link qualities in real life scenarios and on our testbed can easily vary by ± 3 dB within minutes or sometimes even mere seconds, completely avoiding power settings in which the capture effect can occur is not feasible. To nevertheless be able to detect the capture effect, we thus frequently measure the quality of every link used. For a detailed explanation of the capture effect refer to Section 2.3.

In our experiments we considered several tuples of nodes, where one node of each tuple was designated the receiver and was known to have somewhat stable links to the other nodes, the senders. We proceed in rounds, in which first we let all senders send simultaneously twice: first with the same data packet and then with data packets individual to the senders. Then, as a second stage, each of the senders sends alone once. Finally, the receiver gives the senders feedback on the round and optionally also supplies them with fresh synchronization information. During each round the senders keep their sending powers constant. The purpose of the second stage is to ensure we know the received signal strengths (RSS) for all senders. Because a round takes less than half a second, we make the assumption that the wireless environment does not change significantly during the vast majority of all rounds. Knowing the difference in RSS values is vital for us, as it allows us to discern successful receptions which may have been caused by the capture effect, as discussed above. We fix one of the senders' transmission powers at a medium value, such that the other senders will be able to create RSS values both stronger and weaker than the fixed sender. These other senders iterate over a range of power settings whose limits are regularly adapted based on the feedback received between rounds.

3.7.2 Results

First, consider the case of 2 senders. Figure 3.6 shows the relation between the difference in received signal strengths (RSS) and the packet reception ratio (PRR) for senders sending the same data versus sending different data, in which case the senders would set every symbol of the packet payload to a symbol corresponding to their ID. A large positive RSS difference means sender B, which is iterating over different transmission powers over the rounds, was received a lot stronger than sender A, whose transmission power stays constant. Which sender was received can be seen when different data is sent, as is shown by the dashed lines. These plots are particularly interesting because they highlight the interplay of CI and the capture effect: if either sender is received a lot more strongly than the other, it will be received correctly regardless of it sending a different packet. However, as the signal powers become more similar, destructive interference occurs more often. By sending the same data, we are able to avoid a portion of the destructive interference, strongly implying the occurrence of CI.

We were able to observe CI to varying degrees for any triplet of nodes, as long as the senders are capable of creating signals with similar strength at the receiver. As a result of CI the PRR observed at equal RSS increases by 25–35% of samples. The width of the gap between one sender dominating and the other sender dominating appears to be a property of the hardware and environment of a certain node tuple, but does not appear to be connected to the CI we achieve.

Figure 3.7 displays the distribution of the signal strength gained through CI at equal RSS for 2 senders. We observe a large variation both due to an inherent gray area in link quality and due to the variation in transmission synchrony achieved. However, in over 65% of cases we measure an increase of signal strength by 2 dB or more.

We also conducted experiments with 3 and 4 senders, making an even larger case for CI as the capture effect is known to occur less and less as the number of senders increases [49]. Tables 3.2, 3.3 and 3.4 display the PRRs aggregated in two different ways for one run with 3 senders and one run with 4 senders. Empty cells correspond to configurations with fewer than 5 samples. While there is a significant fluctuation in values due to the unavoidably low number of samples in many cells, as we cannot dictate the RSS values, a few clear trends can be observed. Most visibly, the same data case usually exhibits a PRR 25–40% higher than the different data case.

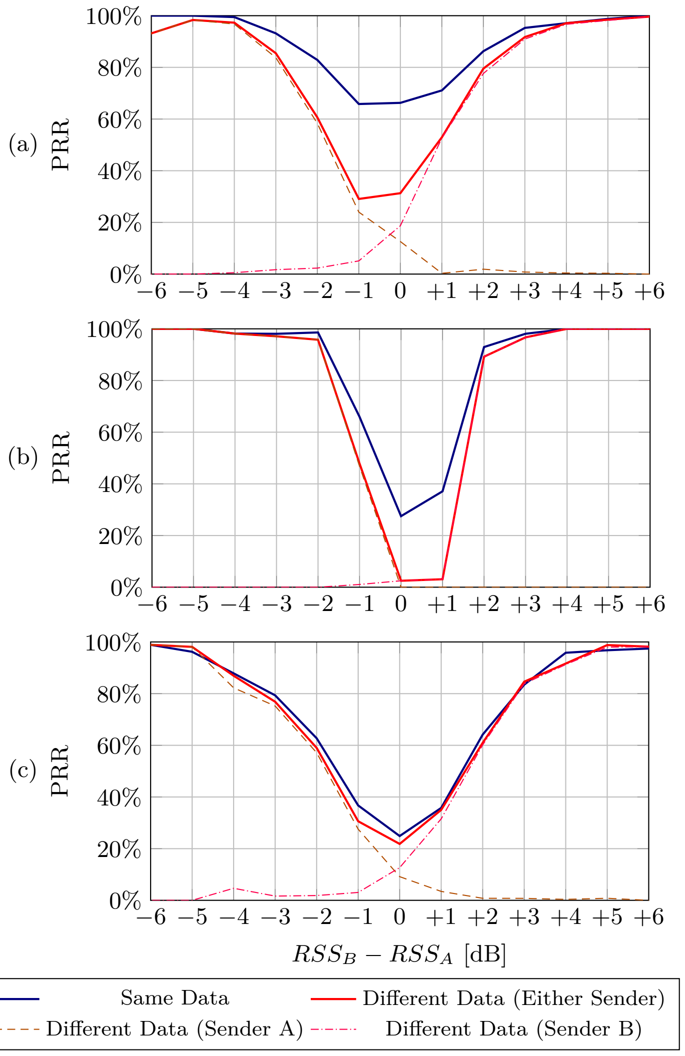


Figure 3.6: The PRR plotted against difference in RSS for two senders for 3 different node triplets, clearly showing CI occurring at a RSS difference close to zero for each triplet, while the capture effect causes larger differences to almost always succeed. (a) and (b) show two pairs of senders with differing behavior. (c) depicts the result of deliberately mistiming one of the senders by 1 µs, preventing any CI from occurring.

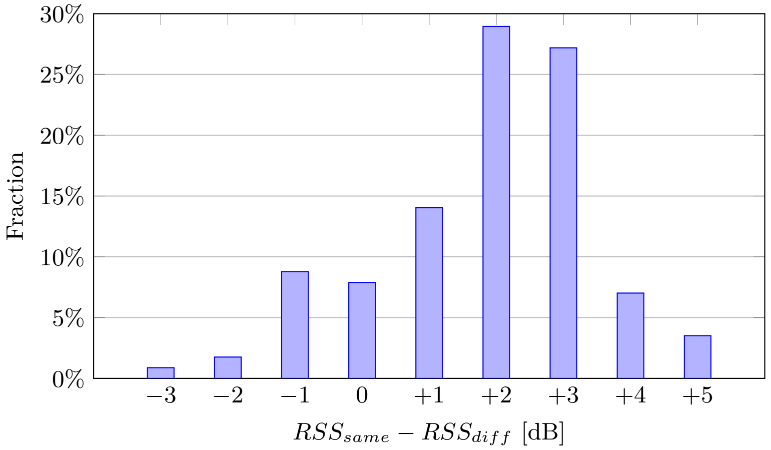


Figure 3.7: The distribution of of the measured RSS gain of simultaneous sending versus either of the two senders, given both senders sending at the same power.

Same Data	0	1	2	3	4	5
0						
1	32%	35%				
2	39%	35%	34%			
3	67%	36%	40%	41%		
4		43%	48%	41%		
5	58%	59%	51%	46%	42%	56%

Different Data	0	1	2	3	4	5
0						
1	0%	2%				
2	0%	0%	1%			
3	7%	3%	2%	0%		
4		14%	9%	8%		
5	83%	50%	21%	3%	3%	0%

Table 3.2: The PRRs measured in an experiment with 3 senders. The results are split by RSS difference between the senders: rows correspond to the difference between the weakest and second weakest sender, columns correspond to the difference between the weakest and strongest sender. (Column/row labels are in dB.)

Same Data	0	2	6	10
0	34%			
2	36%	38%	56%	
4	51%	48%	52%	50%
6	70%	91%	78%	62%
8				100%

Different Data	0	2	6	10
0	2%			
2	1%	2%	0%	
4	23%	10%	12%	0%
6	80%	78%	50%	24%
8				67%

Table 3.3: The same data as in Table 3.2, aggregated differently: the results are split by average RSS difference to the strongest sender (rows) and the variance amongst the powers of the non-strongest senders (columns). (Column/row labels are in dB.)

Same Data	0	2	4	6	8	10
0	10%					
2	23%	24%	36%	21%		17%
4	30%	32%	30%	18%	9%	56%
6	25%	34%	36%	33%	22%	
8	50%	50%	50%			

Different Data	0	2	4	6	8	10
0	0%					
2	1%	0%	0%	0%		0%
4	1%	1%	2%	0%	0%	0%
6	4%	6%	9%	6%	6%	
8	30%	28%	29%			

Table 3.4: The same aggregation as in Table 3.3, but for a different experiment with 4 senders.

Further, the capture effect, whose occurrence can easily be discerned from the different data case, occurs more often the further the strongest sender's signal strength is from the rest (bottom left in Table 3.2, bottom and to a lesser degree also left in Tables 3.3 and 3.4). Finally, particularly high values can be found in the bottom right of Table 3.2, which are likely a result of the possibility of the two much stronger signals invoking both CI as well as the capture effect to survive the occasional timing errors in the weakest signal.

As a further confirmation of our approach, we also conducted a few runs of the experiment in which we configured one of the senders to deliberately send 1 µs late. The expectation is to have this completely remove all possibility for CI. The result of one such run can be seen in Figure 3.6(c), exhibiting almost no CI at all. This underlines the importance of the synchronization precision we strove for. Similarly, when increasing the resynchronization interval we found the effect to diminish after varying periods of time: in some cases as soon as after 30 seconds, in others not for multiple minutes. This can be attributed to the clock synchronization error rising due to changes in clock drift which drift compensation can no longer account for (see Figure 3.5).

For the packets sent by the senders we used payload sizes ranging from 12 to 26 bytes. Together with the 2 final checksum bytes, 4 bytes of preamble and 1 SFD byte, transmissions were thus $19 - -33$ bytes or $608 - -1056$ µs in length. The different packet sizes did not exhibit a significant difference in performance.

3.8 Summary and Future Work

We determined a bound for the total allowable error for CI to occur, and considered three the error sources of clock synchronization error, transmission synchronization error and travel time error:

$$e_{total} = e_{clock} + e_{transmit} + e_{travel} \overset{!}{\ll} 0.5\ \mu\text{s}$$

We reduced e_{clock} by combining the TelosB's 32 kHz quartz crystal with the MSP430's DCO to obtain a stable 4 MHz clock, and we reduced $e_{transmit}$ by optimizing our transmission code path down to the CPU cycle. Both of these values were minimized into the range of the length of a single clock tick (2^{-22} s). In an attempt to reduce e_{travel}, we picked senders such that they had roughly the same distance to the receivers.

Using our example implementation we conducted several experiments with 2 to 4 senders trying to create CI by simultaneously sending the same packet to a recipient node. To ensure packets were not arriving merely due to the capture effect we also sent packets with differing data. Our results show an increase in signal strength of at least 2 dB in 60% of cases for 2 senders, and an increase in PRR of 20–35% for any amount of senders when signal strengths are equal. Considering we expected to only meet the necessary signal alignment requirements in the fraction of samples, in which both e_{clock} and $e_{transmit}$ happened to be small enough, these 20–35% appear quite satisfactory and are likely mainly held back by the precision of the underlying hardware.

While our implementation and results are based on the TelosB and the 802.15.4 standard, we believe the concepts to apply in a similar fashion to most sensor nodes and wireless standards. Further, there are two properties the TelosB was lacking, which were stunting our results: 1) a clock with a precision a magnitude higher than the used chip length and 2) a transceiver designed for a similar precision in transmission timing.

In conclusion, we showed that maintaining the option of CI on commodity sensor node hardware over longer periods of time is feasible, through extraordinary inter-node synchronization and without incurring the overhead of a global “wave” of synchronizing packets before every transmission requiring CI.

4

Capturing Attention Using the Capture Effect

4.1 Introduction

Wireless networks have been around for a long time, and almost always the participating nodes adhere to a simple setup: There is a sender which transmits data to a single receiver or a set of receivers. More recently, this setup increasingly includes multiple antennas on both sides (multiple-input and multiple-output, MIMO). However, conceptually, we still have a single sender trying to transmit a packet to a (set of) receiver(s).

In this chapter, we introduce a third party, a "third man" in the form of a second sender. This second sender tries to capture the attention of the receiver(s) using the capture effect. The second sender ignores the standard rules of wireless communication, and blatantly transmits *during* the transmission of the first sender. The second transmitter could transmit during the payload of the original packet, or during its header. We want the second sender to be stronger than the first sender, either by being closer to the receiver(s), or by transmitting with more power.

We present two scenarios benefiting from such unorthodox behavior. However, we believe that the general idea of having a second sender may have potential beyond our two examples.

Our first scenario addresses networks with partially low density, e.g., where a few nodes failed and the network was separated into two or more parts that can hardly communicate with each other. Consider the case of a spread out multi-hop wireless network containing a "chasm", i.e., a virtual or physical gap between two parts of the network across which links are very poor, as illustrated in Figures 4.1 and 4.2. Even if the network still forms a connected graph via routes around the chasm using only stable links, these detours incur a large latency and medium usage overhead due to the additional hops packets have to make.

By sending across the chasm we might collect several partially incorrectly received packets at multiple nodes on the receiving side. These erroneous copies may then be pieced together to obtain a completely correct copy of the sent packet. However, if already the synchronization header is not received correctly at a node, that node will not recognize the incoming packet and hence will not be able to record it. By employing a node broadcasting a fake synchronization header on the receiving side, we are able to ensure that every node which is a candidate for receiving at least parts of the packet is listening. This improves the chances of correctly receiving every part of the packet in some cases from less than 5% to about 30%.

The second scenario we discuss makes use of what we call the packet-in-packet communication primitive: Here, the second sender does not transmit during the header, but during the payload of the first sender's packet. As such, the second sender may inject a short packet into concurrently ongoing transmissions using the capture effect. Doing so, the time and control message overhead for avoiding collision between high-priority and low-priority traffic can be reduced to a minimum. This is particularly beneficial when the high-priority traffic consists of short spontaneous bursts.

Naively sending a high-priority packet into an ongoing transmission with a sufficient power differential for the capture effect to occur (about 4–5 dB on TelosB) results in the inserted packet being decoded correctly in only about 5% of cases. However, with proper improvements, we can bring this value up to 70%.

In particular, we apply two techniques to improve the success rates in both scenarios: 1) improving transmission synchronization and 2) repairing misinterpreted symbols where possible.

1) By improving transmission synchronization we increase the chance of the symbols of the strong/injected and the weak/base packet to overlay closely enough, such that the symbols have a high chance of being decoded properly. In the chasm scenario this results in a complete packet transmission in 25% of cases. For packet-in-packet this improves the correct decoding rate to about 40%.

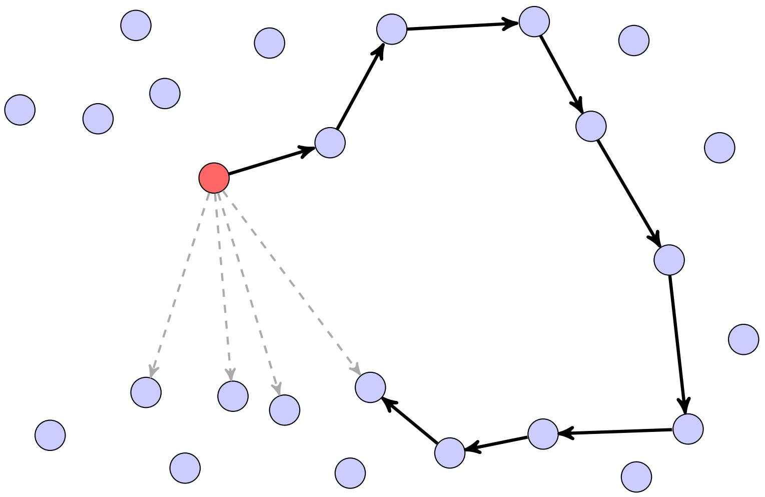


Figure 4.1: Chasm scenario: connection from the red node to bottom half of the graph via A) stable edges around the right side versus B) unstable crossing edges (dashed and in gray).

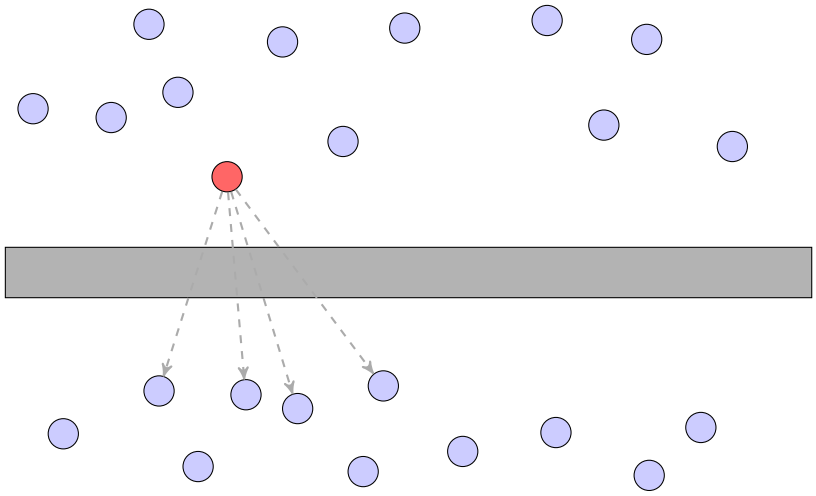


Figure 4.2: Alternative chasm scenario: the network is partitioned by a wall without any stable edges crossing it.

2) Among the remaining bad cases we observe many to experience a deterministic mapping of values on the injected symbols. By reversing this mapping we are able to repair some of the bad cases, raising the total correct decoding rate to 30% in the chasm scenario and to 70% in the packet-in-packet scenario.

We implement proofs of concept of these techniques based on Contiki for both scenarios on TelosB and test them on the FlockLab testbed.

4.2 Related Work

The inherent multi-hop topology of wireless sensor networks as well as the unique properties of the wireless medium have enabled the inception of several new communication primitives. In particular, most recently, ways to induce constructive interference through concurrent forwarding of a single transmission have been explored [18,19]. Constructive interference can boost signal quality and allows rapid network flooding without the overhead of a carefully planned flooding schedule. Another example is the deliberate use of the capture effect, which was previously thought of as a nuisance. Yet it is now counted on more and more to resolve collisions without experiencing destructive interference, which most classical models would predict [76] (see also Chapter 2).

Much like the previous two chapters, our work in this chapter also mainly falls into this field of medium access primitives. However, one of our scenarios also deals with the topic of unreliable links. To avoid repeating ourselves too much, we will omit some of the related work here and refer the gentle reader to Sections 2.2 and 3.2.

The popular Glossy protocol [19] provides a technique by which simple single antenna sensor nodes could send with unprecedented synchrony, leading to a fast single source flooding algorithm avoiding collisions through constructive interference and the capture effect. This technique is based on tying the departure time of each packet to the end of the incoming transmission of the previous packet, effectively reducing the timespan during which clock skew can erode synchrony. For the scenarios examined in this chapter, a Glossy-like mechanism is not applicable, as 1) the presence of a common reference packet for all senders is not guaranteed and in fact impossible in the case of the chasm and 2) these scenarios aim to have the two senders start sending at different times rather than simultaneously.

Unreliable links have been studied from various different angles. Zuniga et al. [81] proposed an analytical model predicting the behavior of links in the transitional region between stable and nonexistent communication. On the other hand, adapting behavior to deal with intermittently missing

4	1	1	Length - 2	2
Preamble	SFD	Length	Payload	Checksum

Figure 4.3: IEEE 802.15.4 PHY layer packet format plus checksum footer (technically part of the MAC layer). The values on top designate the length of the respective segment in bytes. The preamble consists of only zero symbols and the SFD (Start of Frame Delimiter) is a pair of constant symbols. The checksum is a simple 16-bit CRC value of the payload.

links has been proposed: Su et al. [74] developed a set of routing algorithms dealing efficiently with intermittent link outages, while Seth et al. [72] came up with a system to facilitate communication in rural areas piggybacking on vehicles.

Santhapuri et al. [69] proposed a hardware feature allowing a transceiver to disengage from the reception of an ongoing transmission and lock onto a newly started stronger one. To do so they proposed the hardware keep scanning for synchronization headers even during transmission reception, similar to the way we detect injected packets. However, the approaches we present in this chapter do not rely on such hardware support, but instead make do with the capabilities of run-of-the-mill transceivers. Further, in most cases of the packet-in-packet scenario we are able to recover after the injected packet and correctly receive the remaining tail end of the original packet.

To the best of our knowledge, cleanly injecting and decoding packets in packets as we study in this chapter had not been attempted yet.

4.3 Concepts

4.3.1 Preliminaries

In the following we will discuss the basics of wireless receiver operation relevant to our undertaking. The details of the parts of a transmission vary slightly between wireless standards, but the presented primitives are only loosely tied to the IEEE 802.15.4 standard [32] we used. See Figure 4.3 for an overview of the IEEE 802.15.4 packet format.

When a packet is transmitted, it is usually prepended a preamble and a synchronization word (sometimes called start of frame delimiter, SFD). Together, the preamble and the synchronization word form the synchronization header. The preamble is a predefined pattern of symbols used to let listening receivers synchronize to the correct phase of symbols and chips.

In IEEE 802.15.4 it consists of '`0`' symbols only. The synchronization word is a constant string of symbols to mark the start of the packet, which signals receivers to start recording the following symbols. If a receiver does not hear the synchronization word of a transmission, it will not decode the signal and perceive it as noise.

At the head of a packet there usually is a length field specifying the number of bytes in the payload, and thus also the duration of the transmission. Receivers use this field to know how many symbols to decode. A common feature of receivers is to "lock on" to a transmission once they received its header including the length: they commit to decoding that packet in its entirety no matter how bad the signal quality may get. This is intended to allow the application to still make use of the non-broken parts of a packet even if part of the packet reception was disrupted.

Finally, a packet usually contains a checksum over its length and payload to allow telling whether the whole packet was received correctly. However, if there are any errors, the checksum does not help in determining their location.

The capture effect occurs when multiple signals arrive at a receiver concurrently and the strongest signal is stronger than the noise plus the other signals by some threshold. In this case, the receiver will decode the strongest signal without error, completely ignoring the other signals. When a receiver is already receiving a packet, it does not scan for further synchronization headers. Thus, an overlapping packet may not be recognized as a packet, even if it is strong enough to induce the capture effect. Instead, a stronger but later packet essentially writes its data over the payload of the weaker packet as it is being received. This means that the overwritten data of the weaker packet is lost, but it also means that the stronger packet is not lost. In many cases, the receiver can find the stronger packet's data (and headers) either directly in the weaker packet's payload (if the two packets' symbol phases were well synchronized) or after descrambling the received symbols (see Section 4.6).

4.3.2 The Chasm

Consider the situation described in Figures 4.1 and 4.2: traditionally, to send a packet to the other side of the chasm, one would use the multi-hop route using the stable edges. This route, however, causes a high latency for the packets and incurs a noticeable overhead in medium usage. Worse, sometimes this route may not even exist, and without being able to send across the chasm the network graph would be disconnected.

In the chasm scenario, we isolate this situation and explore what is possible using the weak links only, hence avoiding the detour. We assume

```
1b1f21ffdededededededededededededededededede

1b1f229fd3dedededededeeed2deddddededededede8ede
1b1f21ffdeded4dededededededededededededededede
1b1f21ffde2ede1ed77eddd92ed2ded0deded7dede6e94
```

Figure 4.4: An example of a single short packet after being received by 3 different receivers, exhibiting seemingly independent decoding errors. The first line shows the packet as it was sent.

the weak links to have a near zero chance of transmitting a complete packet correctly, while still being capable of correctly decoding some symbols now and then. If after a cross-chasm transmission the nodes on the receiving side exchange what symbols they got, they may be able to piece together the complete packet and verify its correctness using the checksum field.

For this setup to be viable, errors in the transmission should not occur at or near the sender. Instead, they should occur in the decoding stage at each receiver, independently of the other receivers. We found this assumption to be correct in our setup by comparing the received symbols of multiple receivers for the same packet. An illustrative example can be seen in Figure 4.4.

As mentioned above, the receivers will in fact only record incoming data if a correct synchronization header is received. We cannot make the assumption that this will always be the case for our receivers in the chasm scenario, since the symbols of the synchronization header are just as likely to not be received correctly as any of the data symbols. In other words, we will not get any information about the packet from the portion of receivers which failed to recognize the synchronization header.

To solve this problem we propose having another node on the receiving side of the chasm send out a synchronization header as well as a large packet length field, but not transmit any packet payload, just before the cross-chasm packet arrives (see Figure 4.5). This header should be easily recognizable by all the receivers due to their proximity and cause them to start recording symbols.

Note that this does require knowledge of when the next cross-chasm packet will arrive. This may be determined from a pre-determined schedule or be agreed upon in a previous cross-chasm packet. It also implies maintaining a synchronization error low enough to ensure the cross-chasm packet is sent during the fake packet. In IEEE 802.15.4 the duration of a maximum length payload is about 4.1 ms. This dictates a maximum absolute synchronization error of around 2 ms for short cross-chasm packets. The longer the cross-chasm packets are, the more stringent the synchronization needs to be,

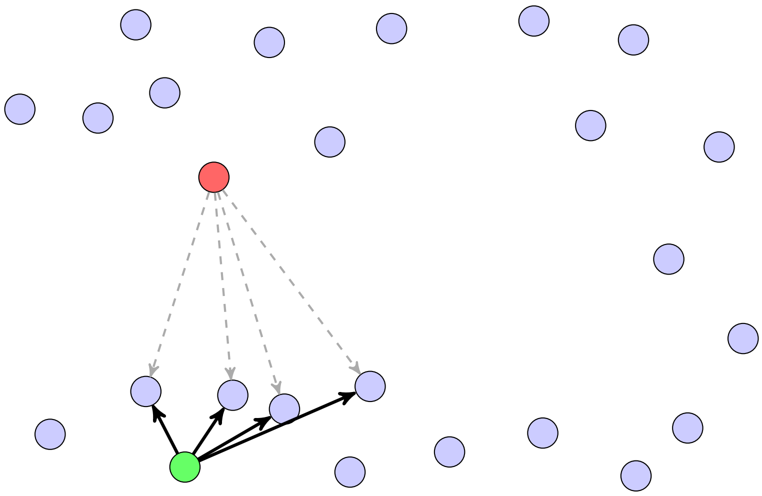


Figure 4.5: By having the green node send out a synchronization header first we can ensure that all the nodes with a chance of receiving parts of the cross-chasm packet are indeed listening and recording symbols.

up to a symbol level synchronization at maximum length. An alternative would be to try and detect packets by monitoring the energy levels on the channel. However, this approach is prone to false positives from noise and non-chasm transmissions. Due to the large overhead incurred, we propose to employ this primitive only when the detour alternatives are prohibitively long or nonexistent.

In our implementation we pre-assign all nodes their respective roles (cross-chasm sender, wakeup sender and receivers) and establish a proof of concept for the ideas above. We consider short cross-chasm packets only and no other traffic within the network. Integrating the transmission primitive into a more general and adaptive protocol is left to future work.

4.3.3 Packet-In-Packet

For this scenario imagine a network of nodes being used both for low-priority bulk data transmission and short, irregularly occurring high-priority messages. This is similar to the scenario considered in Chapter 2, but assumes the lower-priority packets to generally have large payloads. Further, in this chapter we do not require the adherence to time slots. Given such a similar scenario, traditional wisdom proposes the same approaches: One could use a schedule guaranteeing time slots for high-priority traffic, but this comes with a large overhead to the low-priority traffic even during times without high-priority traffic. To avoid the overhead of scheduling one could instead use an opportunistic channel access scheme: trying to send as soon as the channel appears free (using clear channel assessment, CCA). This approach has trouble guaranteeing traffic prioritization in busy networks as low-priority traffic may starve out high-priority traffic. Additionally, the hidden and exposed terminal problems become problematic [33]. To improve traffic prioritization, one could impose time slotting on the network. By then delaying low-priority messages by a constant time high-priority messages are allowed to access the channel first. This, however, still does not solve the hidden and exposed terminal problems.

Beyond the obvious downsides, the traditional approaches also have in common that they require high-priority messages to wait at least until all currently ongoing conflicting low-priority messages have finished. The packet-in-packet primitive we propose makes use of the capture effect with the aim to allow high-priority messages to be sent regardless of any ongoing low-priority messages, i.e., immediately upon traffic emergence. If a low-priority message is already being received when a high-priority message arrives, it will overwrite and be decoded as part of the low-priority message's payload. We say, the high-priority message is *injected* into the low-priority message. Again, we present a proof of concept implementation of this basic primitive. Integrating it into a proper protocol or a MAC layer is left to future work.

For this scheme to work we need to be able to reliably induce the capture effect, i.e., we need high-priority traffic to always be at least 4–5 dB stronger than any low-priority traffic at the intended receivers. This can be accomplished one of two ways: 1) placing the receiver significantly closer to the high-priority sender than to the low-priority sender, or 2) using transmission power control, which requires the used links to be capable of operating at different transmit power settings. If this condition is fulfilled, no additional overhead is imposed on low-priority traffic beyond the data lost as a direct result from high-priority data taking its place. Otherwise, the additional hops necessitated by the unavailability of certain links too weak or too strong to induce the capture effect may offset this primitive's usefulness.

In addition, this scheme relies on low-priority packets to be at least 2–3 times longer than high-priority packets to be able to operate efficiently. This is because a tail portion of an injected high-priority message will be lost if its transmission does not finish before the end of the low-priority base packet it was injected into. This is a result of the aforementioned behavior of receivers to record exactly as many symbols as specified in the packet header of the packet whose synchronization header was recognized.

The shorter the high-priority messages are in relation to the length of the low-priority packets, the less frequently such a loss occurs.

Detecting the potential presence of an injected packet is trivial using a checksum covering the whole payload as is commonly attached as a packet footer: if the checksum check fails, there may be an injected packet. To find the injected packet, one may simply search the payload for the symbol sequence of the synchronization header. Similarly, the length of the injected packet can be determined, and thus the exact portion of the payload that was overwritten is known. To verify the suspected injected packet, its own regular footer checksum can be used. Extracting further injected packets from the same base packet is possible following the same procedure. Note that we cannot distinguish the injection of a packet from the unlikely coincidence of a regular payload containing a synchronization header and valid packet length together with a matching packet checksum.

Depending on the link quality one may assume the remaining non-overwritten symbols of the packet to be correct in spite of the lack of a correct checksum. However, to ascertain the correctness in the case of an injected packet, additional checksums across parts of the payload or forward error correcting codes could be used. Naturally, the data that was overwritten will need to be transmitted again at a later time.

4.4 Experiment Setup

4.4.1 Hardware

We again employed the TelosB sensor node as testing hardware, deployed on the FlockLab testbed. The channels (frequencies) we used experienced low to medium outside interference from the office environment. We observed instances of the motivating scenarios described in earlier sections in this testbed while using each node's maximum possible transmit power. Unfortunately, the wireless environment conditions proved very prone to fluctuations caused by changing conditions such as the closing or opening of doors or changes in temperature or humidity. To obtain reproducible results, we had to rely on the transmission power control options of the CC2420. The available output powers on the CC2420 range from -30 dBm to 0 dBm with a significantly larger degree of granularity in the higher power options (see Figure 2.4).

On the physical layer, each byte is represented by two symbols: first, one representing the least-significant 4 bits of the byte, followed by a symbol representing the most-significant 4 bits. The IEEE 802.15.4 standard assigns a pseudorandom chip sequence of 32 chips to each of the 16 possible symbol values. Each chip directly corresponds to an interval of a certain waveform on the carrier medium. Figure 3.1 shows an example encoding of the zero symbol, 16 µs in length. When receiving a symbol, the receiver will determine the received symbol to be the one whose pseudorandom chip sequence correlates the most with the received signal. We would expect the pseudorandom chip sequence design to cause a desynchronized waveform (as would be caused by a poorly synchronized injected packet) to be decoded to an "arbitrary" (pseudorandom) wrong symbol; however, as shown in Section 4.6, this is not always the case.

4.4.2 The Chasm

We choose a sending node and a set of receiving nodes to represent the two sides of the chasm, such that there are stable links with roughly equal qualities between the sender and each receiver when the sender uses its maximum transmit power. Additionally, these links should falter at approximately the same transmit power value. We then designate a node amongst the receiving nodes to be responsible for sending the wakeup synchronization header. It should have a good link to all the other receiving nodes at its maximum transmit power. In our experiments, this node will not participate in receiving the packets from the sender. However, it might occasionally be able to, if it happens to receive the synchronization header from the sender correctly. For the results presented in this chapter we used 4 receiving nodes plus the wakeup header sending node, but in practice any number of receivers is imaginable. Even a single receiver may profit from the wakeup header, although it is unlikely to receive the whole packet correctly.

The test procedure consists of several rounds in which, first, the sender broadcasts a packet at maximum transmit power for synchronization purposes. This will allow the node responsible for later broadcasting the wakeup synchronization header to update its clock and to know approximately when the weak packets will arrive. In practice, this shortcut for announcing weak packet arrival times is not available or there would be no reason to use this scheme in the first place. Instead, to achieve initial synchronization, one would need to either use a route around the chasm or rely on getting a single packet through by normal transmission, which may happen with a small percentage chance (see Figure 4.13).

The main part of a round then consists of several packets of length 29 (58 symbols without headers) being sent by the sender at a weaker power $P < P_{max}$. 256 µs before every second of these packets, the wakeup node will send a synchronization header and a packet header specifying the maximum packet length (127) at maximum transmit power. If a receiving node receives a synchronization and packet header, it will store the entire received raw symbol string, no matter which of the two possible headers it received.

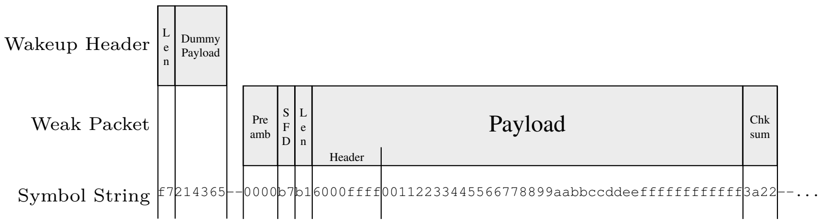


Figure 4.6: The symbol string the receiver would ideally receive in the chasm scenario, each letter corresponding to one symbol representing 4 bits. The symbols are given in order, i.e., each byte's least-significant 4 bits come first. '–' denotes a symbol slot where no node was sending anything; the values of these slots will be determined by the noise in the environment.

The symbol string is then evaluated manually, ignoring the CC2420's automated checksum test. By alternating between sending weak packets with and without the wakeup synchronization header, we can directly measure the advantage gained through our method, independent of fluctuations in the environment.

To send a synchronization and packet header only, the wakeup node starts transmitting a packet of maximum length. However, it then deliberately does not supply the CC2420 with enough data bytes to continue transmitting beyond the first 3 bytes. This causes a buffer underflow to occur, automatically causing the CC2420 to stop transmitting. The receivers of this packet will, however, not notice the ceasing of the transmission and continue recording symbols to the best of their ability as discussed in Section 4.3.1. The chosen advance time of 256 µs corresponds to 16 symbols. This allows the wakeup sender to finish sending its synchronization and packet headers (8 symbols) and the 3 bytes of dummy data (6 symbols) before the weak packet from the sender arrives. Figure 4.6 illustrates the symbol strings sent and what is received in the ideal case.

Each round ends with every receiver reporting to the sender how many packets it received when no wakeup synchronization header was sent. These packets are sent at maximum transmit power to be able to cross the virtual chasm. Using this feedback the sender can adapt its choice of P to be able to concentrate on the interesting transmit power values: values such that no single receiver is likely to receive the packet completely all by itself. In practice, the range of interesting values settles after only a few dozen rounds, i.e., the remaining rounds all repeatedly scan a window of the same few values for P.

We noticed that over 95% of the packets that were received without

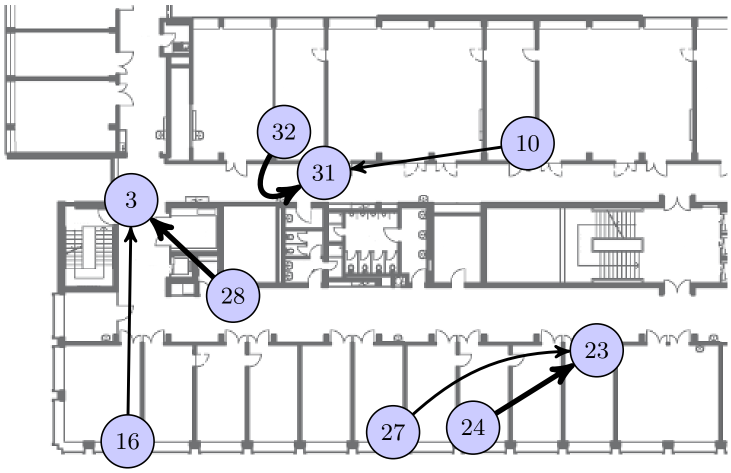


Figure 4.7: A subset of the wireless sensor node testbed we used for our practical tests with three particular node triplets used for packet-in-packet experiments highlighted. The second, stronger senders are indicated by thicker arrows.

a wakeup header were received completely without error. Effectively, if a link happened to be stable enough to receive the complete synchronization header, it also was stable enough to receive the packet completely without error 95% of the time. To reduce the filtering at the synchronization header, we decided to use a shorter preamble of only 2 instead of 4 bytes, shortening the synchronization header to 3 instead of 5 bytes. In the interesting transmit power range, this dramatically increased the number of packets received without a wakeup header, but lowered the ratio of packets received without any error to 0–25%.

4.4.3 Packet-In-Packet

We choose triplets of 2 sending nodes (B and I) and 1 receiving node (R), where node B will send the base packet and node I will send the injection packet. These triplets are selected such that at node R node I's signal is at least 5 dB stronger than node B's, to facilitate the capture effect. Further, we also ensure that node B's link still has a high packet reception ratio ($> 98\%$) in normal operation without packet injections. Finally, both senders should be able to hear the receiver for synchronization purposes. If necessary we use the transmission power control options of the CC2420 to achieve this constellation. See Figure 4.7 for a few example triplets.

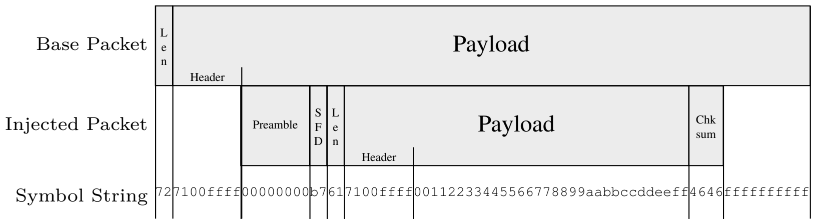


Figure 4.8: The symbol string the receiver would ideally receive in the packet-in-packet scenario. The individual packet fields of both the outer base packet and inner injected packet are shown. Note that the preamble, SFD and checksum of the outer packet are missing as they are already removed by the transceiver by the time software can read the symbol string.

The test procedure again consists of several rounds which start with a packet broadcasted for synchronization purposes. Then, node *B* begins transmitting a packet of length 40 bytes at a designated time after the synchronization packet. Node *I* aims to begin transmission 320 µs after the base packet. This delay corresponds to 20 symbols or 10 bytes, ideally placing the second signal at the fourth payload byte of the base packet.

Both packets contain a short header of 4 bytes which we use to store packet metadata such as sender, receiver and packet type. After, the inserted packet's payload contains 16 bytes repeating each of the 16 possible symbols twice, which will be useful for demonstrating symbol mapping, see Section 4.6. The base packet's payload is filled with the symbol of value 15 (or '`f`'). This choice of packet contents was made for presentation purposes in this chapter. In practice, any data can be used, as our tests show the packet contents not to influence the success in decoding the injected packet. Figure 4.8 shows the symbol string the receiver would receive in the ideal case, and details the locations and sizes of the packets' fields.

Without closer synchronization we would expect to correctly decode the injected packet's symbols in on average at most 1 of 16 cases. In the other cases, we would expect the chip string to be misaligned with the symbol boundary so much that the waveform is interpreted to be a different one of the 16 pseudorandom chip sequences.

4.4.4 Baseline Results

Using the packet-in-packet scenario as example, we present the results one would obtain without applying either of the two improvements discussed in the next two sections.

In practice, the success rate is only 4.8%, slightly less than the predicted

```
727100ffff22222222d103132299992233445566770011aabbccddeeff889960600ff3ffcfcf
727100ffff11111111c0720211888811223344556677009 9aabbccddeeff885757ffffffffff
727100fffff77777777a6506077eeee77001122334455 66ff8899aabbccddee3535ffffffff
727100fffff6666666595475766dddd6677001122334455eeff8899aabbccdd2424ffffffff
727100fffff77777777a6506077eeee7700112233445566ff8899aabbccddee3535ffffffff
727100fffff5555555b423f6557c0ce56617d0911223346d829c58799aab70f1313ffffffff
727100fffffd4d4444d03703ed4b7874d091f4b33447733f58199c3bbecff7b3b35ffffffff
727100fffff66668816209e5b664444162d314e001149002f35775b6329064e02bffffffffff
727100ffff11111111c07202118888112233445566770099aabbccddeeff885757ffffffffff
727100fffffd4d4414d03703ed4b2b74d09112b338452f3596179657bec3ab93b35ffffffff
727100ffff44444444ff2535444bbbb4556667700112333cddeeeff8899abb0202ffffffffff
727100ffff33333333e2142433aaaa3344556677000122bbccddeeff8899aa71711ffffffff
727100fffffeeeeeeee1f9cacee7552eecc88e9aa88f9dd6644006127334455acacffffffff
727100ffffdddddddd0fb9fedd8b88ddefff8d99a8bbffff8794a5ddecff889b9bffffffffff
727100ffff22252522b7637f22f33332a644556657667bb12e6664e01604f366d6ffffffffff
727100fffff66868666219e5b66b7b86550a1d800f1ee30eed5b7cb8279617eb209ffffffff
727100fffff66666666695475766dddd6677001122334455eeff8899aabbccdd2424ffffffff
```

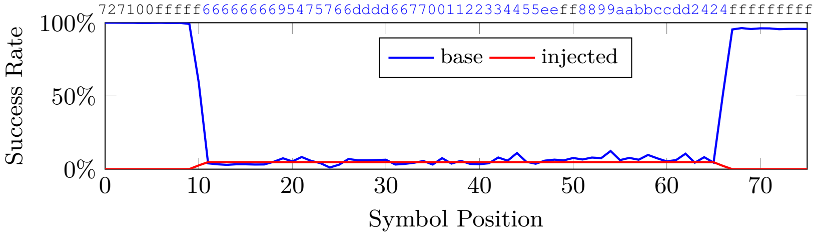
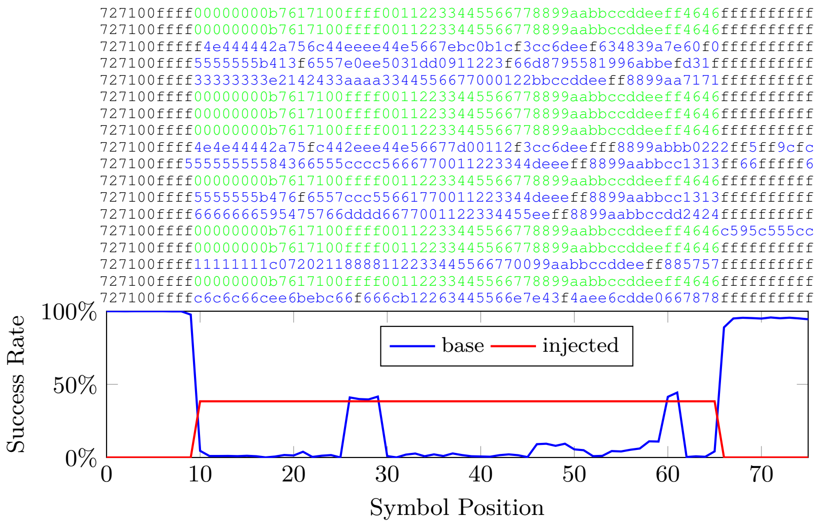


Figure 4.9: Loose Synchronization, No Symbol Remapping. Sample and correctness rate on a per symbol basis.

$\frac{1}{16} = 6.25\%$. For intuition, examine Figure 4.9, which shows a representative sample of 17 received symbol strings. Every letter corresponds to a symbol in symbol transmission order, i.e., for every byte the 4 least-significant bits come first. Unexpected symbol values are marked in blue. Since not a single preamble was detected, all non-‘f’ symbols are classified as errors in the body of the underlying packet.

It is easy to see the lack of tight synchronization in the variation of the first and last affected symbol of the underlying packet. However, a certain regularity in the erroneous symbols can be observed. In some cases, we can exploit this to salvage some of these scrambled lines, see Section 4.6.

At the bottom of Figure 4.9, the distribution of correctly received symbols can be found. The red line for the injected packet is almost completely constant at around 5% for the duration of the injection. This means, when an injected packet’s preamble was correctly decoded, the remainder of the packet was nearly always also completely without error. The uneven distribution of the base packet’s successes is caused by the varying densities of ‘f’ symbols caused by the injection. This is due to certain of the injected packet’s symbols appearing to be more likely to be mapped to an ‘f’ than others. Finally, note the “tail” of the base packet sometimes experiencing symbol errors even after the transmission injection has ceased.



```
727100ffff00000000b7617100ffff00112233445566778899aabbccddeeff4646ffffffffff
727100ffff00000000b7617100ffff00112233445566778899aabbccddeeff4646ffffffffff
727100fffff4e444442a756c44eeee44e5667ebc0b1cf3cc6deef634839a7e60f0ffffffffff
727100ffff5555555b413f6557e0ee5031dd0911223f66d8795581996abbefd31ffffffffff
727100ffff33333333e2142433aaaa3344556677000122bbccddeeff8899aa7171ffffffffff
727100ffff00000000b7617100ffff00112233445566778899aabbccddeeff4646ffffffffff
727100ffff00000000b7617100ffff00112233445566778899aabbccddeeff4646ffffffffff
727100ffff00000000b7617100ffff00112233445566778899aabbccddeeff4646ffffffffff
727100ffff4e4e44442a75fc442eee44e56677d00112f3cc6deefff8899abbb0222ff5ff9cfc
727100fff55555555584366555cccc5666770011223344deeeff8899aabbcc1313ff66fffff6
727100ffff00000000b7617100ffff00112233445566778899aabbccddeeff4646ffffffffff
727100ffff5555555b476f6557ccc55661770011223344deeeff8899aabbcc1313ffffffffff
727100ffff6666666595475766dddd66770011223344455eeff8899aabbccdd2424ffffffffff
727100ffff00000000b7617100ffff00112233445566778899aabbccddeeff4646c595c555cc
727100ffff00000000b7617100ffff00112233445566778899aabbccddeeff4646ffffffffff
727100ffff11111111c07202118888112233445566770099aabbccddeeff885757ffffffffff
727100ffff00000000b7617100ffff00112233445566778899aabbccddeeff4646ffffffffff
727100ffffc6c6c66cee6bebc66f666cb12263445566e7e43f4aee6cdde0667878ffffffffff
```

Figure 4.10: Tight Synchronization, No Symbol Remapping. Sample and correctness rate on a per symbol basis.

4.5 Transmission Synchronization

We apply the techniques presented in Sections 3.5 and 3.6 again to minimize both clock synchronization error and transmission timing in both our scenarios. We observe a clock error within about ±0.2 µs in 70% of cases and within ±0.5 µs in 95% of cases when synchronizing at least once every couple of seconds. As before, we can reduce the transmission timing error caused by our code paths to the minimum possible of at most 1 DCO cycle, and we expect the time between issuing the start of transmission and the transmission starting to be constant.

Applying this rigorous transmission synchronization, the results in the packet-in-packet scenario improve to 38.4% of cases allowing correct decoding of the injected packet. Figure 4.10 shows another representative sample of received symbol strings. Blue indicates errors in the base packet (as before), green indicates the recognized and correctly decoded injected packet, and red indicates errors part of the injected packet (although there are no such errors in this sample).

Again, the injected packet barely experiences any internal symbol errors, and its symbol decoding success rate remains constant, this time at

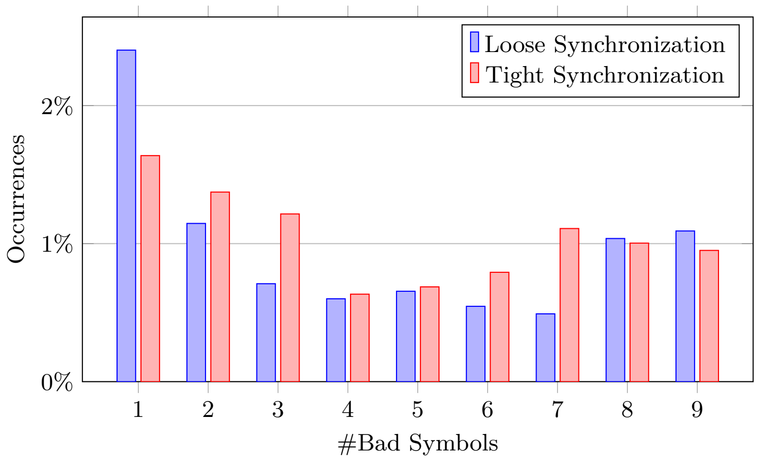


Figure 4.11: Distribution of the number of misinterpreted symbols after the injected packet. Not pictured: in 91% of cases none of the symbols are broken.

around 40%. The base packets' apparent peaks in decoding successes during the injection period is in fact misleading, as the injected packet's correctly decoded '`f`' symbols are interpreted as also matching the base packet and occur at static offsets.

Note that the tail of the base packet is still experiencing decoding errors. Figure 4.11 shows the distribution of the number of symbol errors in the tail. While both with loose and tight synchronization the tail is decoded completely correctly in about 91% of cases, loose synchronization appears to more often only incur a single symbol error. There is also a tendency visible for all of the remaining 8–9 tail symbols to be corrupted. This may be explained by the demodulation of correct but slightly misaligned symbols sometimes throwing the synchronization of the demodulator off permanently.

4.6 Mapping Symbols

4.6.1 Mapping Symbols

In the last sample (Figure 4.10), a large degree of determinism in the symbol decoding error can be observed in many of the broken packets. In fact, there appears to be a one-to-one mapping between actual and misinterpreted symbol values in most cases, albeit there appear to exist 7 different mappings.

Based on the symbols which were received at the location of the preamble of the inner packet we can identify which mapping took place. If we denote by a the value of the preamble symbols and by s the correct symbol value, the 7 one-to-one mappings $m(s,a)$ for $1 \leq a \leq 7$ are:

$$m(s,a) = s - (s \bmod 8) + ((s+a) \bmod 8)$$

We can exploit the bijectivity of these mappings to reverse them to reconstruct the original symbols in the majority of cases.

There are no simple one-to-one symbol mappings for the remaining cases of $8 \leq a \leq 15$. This is easy to see when examining the pairs of symbols in the payload of the injected packet not being recognizable as pairs anymore. However, it is likely that some information can still reliably be decoded, as certain pairs of subsequent symbols should also follow an exploitable deterministic mapping, judging by the interplay of the predefined chip sequences when time-shifted.

To detect injected packets after such a symbol mapping within the base packet, we not only search for the synchronization word (preamble + SFD) as before, but also for the synchronization word with each of the 7 mappings applied. If one is found, the respective mapping is then used to decode the length, payload and checksum of the injected packet.

4.7 Results

4.7.1 The Chasm

The wakeup synchronization header sent at maximum power by a close neighbor unsurprisingly increases the number of cases in which any symbols were recorded to essentially 100% (see Figure 4.12).

Figure 4.13 shows how often every symbol in the packet was received correctly at least once. For lower transmit power settings – emulating the case of weak links – in the range from 7 to 10, corresponding to transmit powers from -15 dBm to -10 dBm [75], our scheme raises the chance of receiving a complete copy of the packet from 5% to 30%. Without the symbol remapping only 25% are achieved.

As the transmit power increases, regular transmissions without wakeup reach $> 99\%$ reliability. This is because frequently one of the receivers receives a perfect copy of the packet by itself, while our approach still suffers from the penalties induced by using a mismatching synchronization header. Hence, we do not recommend using our approach for any and all links, but rather only for chasm-like scenarios in which only weak links are available.

Figure 4.14 shows the number of correct symbols received on average within the transmit power setting range of 6 to 10 at each of the individual

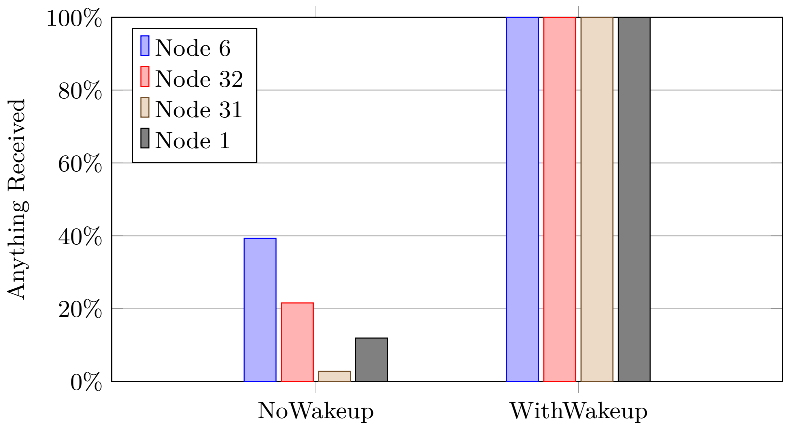


Figure 4.12: Percentage of received packets with and without the wakeup synchronization header at all 4 receivers of a test run.

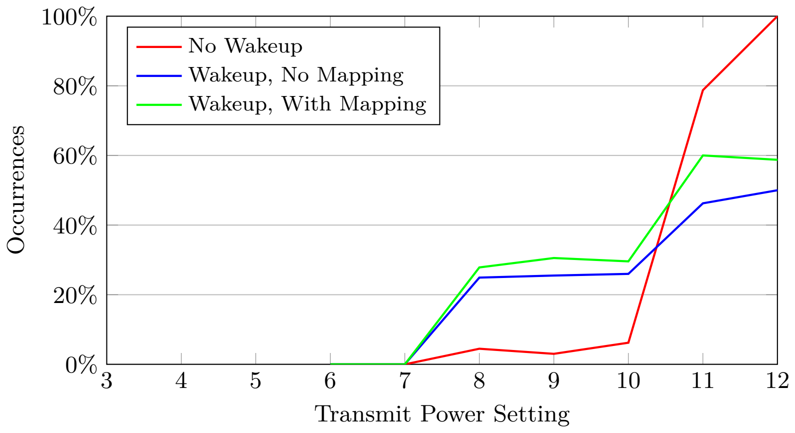


Figure 4.13: Percentage of cases in which at least 1 complete copy of the sent weak packet could be reassembled. Up to transmit power 10, our approach of adding a wakeup synchronization header significantly improves the chances.

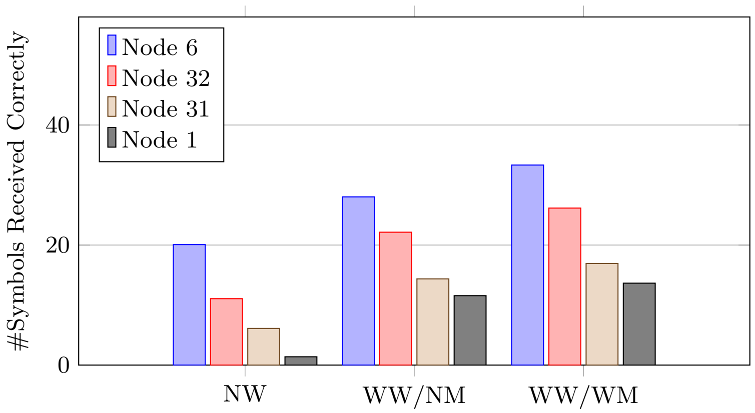


Figure 4.14: Average number of correct symbols received (out of 58) with and without the wakeup synchronization header at all 4 receivers of a test run. NW/WW = No/With Wakeup, NM/WM = No/With Symbol Mapping.

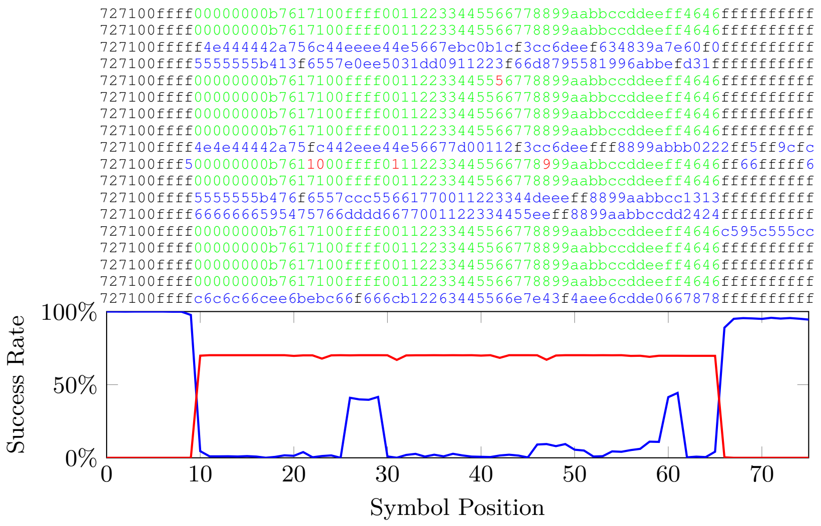
receivers. While the node with the best link (node 6) only gained about 65% of additional correct symbols, overall the number of correct symbols received more than doubled. As which symbols are correctly received for a receiver is independent of the other receivers (see Section 4.3.1), it stands to reason such an increase directly improves the chances of being able to assemble a complete correct copy of the packet.

The test run used for these plots contained 3800 samples, 3000 of which were taken within the transmit power setting range of 6 to 10.

4.7.2 Packet-In-Packet

In the packet-in-packet scenario, applying both techniques increases the success rate for decoding the injected packet completely without error to 64.3%. Tolerating up to 3 symbol errors, a success rate of 70% is reached. Figure 4.15 shows the same sample as in Figure 4.10, but with symbol remapping applied. It is easy to see that some of the erroneous lines from before have been corrected. However, in some instances a few symbols were corrupted (marked in red). Peculiarly, judging from the success rate graph at the bottom, some locations in the packet are particularly prone to symbol errors not following the mapping.

Figure 4.16 shows the distribution of the number of symbol errors in the decoded injected packet given that its synchronization word was intact and recognizable. While the number of errors appears to be higher



```
727100ffff00000000b7617100ffff00112233445566778899aabbccddeeff4646ffffffffff
727100ffff00000000b7617100ffff00112233445566778899aabbccddeeff4646ffffffffff
727100fffff4e444442a756c44eeee44e5667ebc0b1cf3cc6deef634839a7e60f0ffffffffff
727100ffff5555555b413f6557e0ee5031dd0911223f66d8795581996abbefd31ffffffffff
727100ffff00000000b7617100ffff00112233445556778899aabbccddeeff4646ffffffffff
727100ffff00000000b7617100ffff00112233445566778899aabbccddeeff4646ffffffffff
727100ffff00000000b7617100ffff00112233445566778899aabbccddeeff4646ffffffffff
727100ffff00000000b7617100ffff00112233445566778899aabbccddeeff4646ffffffffff
727100ffff4e4e44442a75fc442eee44e56677d00112f3cc6deefff8899abbb0222ff5ff9cfc
727100fff500000000b761 1000ffff01112233445566778999aabbccddeeff4646ff66fffff6
727100ffff00000000b7617100ffff00112233445566778899aabbccddeeff4646ffffffffff
727100ffff5555555b476f6557ccc556617700112233 44deeeff8899aabbcc1313ffffffffff
727100ffff6666666595475766dddd66770011223344 55eeff8899aabbccdd2424ffffffffff
727100ffff00000000b7617100ffff00112233445566778899aabbccddeeff4646c595c555cc
727100ffff00000000b7617100ffff00112233445566778899aabbccddeeff4646ffffffffff
727100ffff00000000b7617100ffff00112233445566778899aabbccddeeff4646ffffffffff
727100ffff00000000b7617100ffff00112233445566778899aabbccddeeff4646ffffffffff
727100ffffc6c6c66cee6bebc66f666cb12263445566e7e43f4aee6cdde0667878ffffffffff
```

Figure 4.15: Tight Synchronization, With Symbol Remapping. Sample and correctness rate on a per symbol basis.

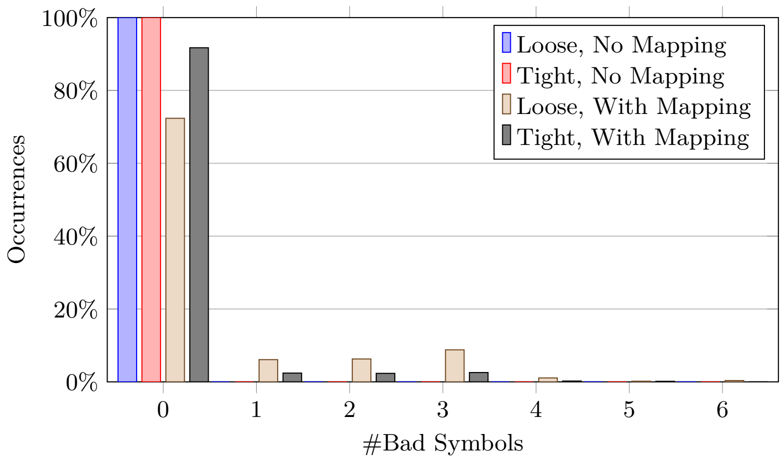


Figure 4.16: Distribution of the number of bad symbols within the injected packet once the injected packet's preamble has been read correctly.

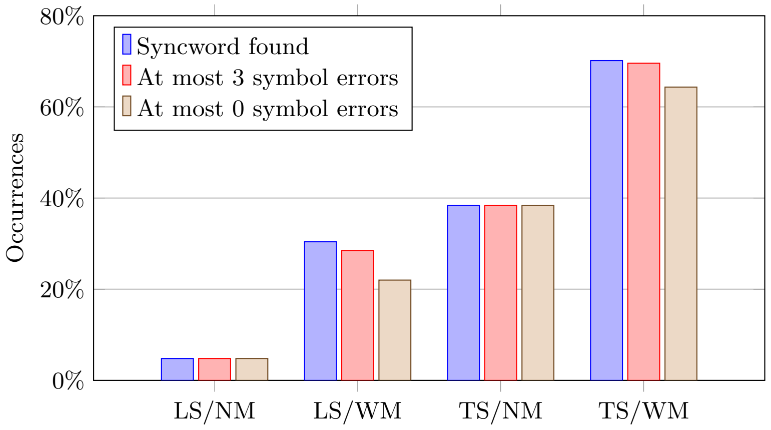


Figure 4.17: Success rates for decoding the injected packets using the different techniques. LS/TS = Loose/Tight Synchronization, NM/WM = No/With Symbol Mapping.

when applying symbol mapping, this is merely due to the fact that in these cases additional synchronization words could be salvaged. These additional instances appear to be particularly prone to symbol errors, although the extent is low enough to still preserve most of the packet data. Further, tighter synchronization appears to reduce the occurrences of these errors.

The total amount of recovered packets can be seen in Figure 4.17, subdivided by the combination of applied techniques. Enforcing a limit on the number of symbol errors hurts the performance of the cases employing symbol remapping, as is to be expected. However, allowing up to 3 symbol errors almost completely closes the gap to the number of cases in which the synchronization word of the injected packet was at all detected. Recalling our injected packet being 46 symbols in length (when excluding the synchronization word), an error of 3 symbols appears rather acceptable.

The test run used for these plots used the node triplet $R = 23, I = 24, B = 27$ and contained 1900 samples for loose and tight synchronization each.

4.8 Summary and Future Work

We explored the possibilities unfolding when mixing and matching synchronization headers and packet payloads as is made possible by inducing the capture effect. Correctly interpreting the resulting sometimes jumbled symbol string is not trivial and certainly not as efficient as with a matching synchronization header. However, we showed that some scenarios may nevertheless profit: A) bridging a chasm which is only crossed by weak links and may otherwise only be crossed taking a long detour – or not at all, if the network graph is not connected otherwise, and B) propelling high-priority packets across a network without any waiting or any control message overhead, barely disturbing low-priority traffic. The resulting gains we observed in our experiments are best summarized by Figures 4.13, 4.14 and 4.17.

Integrating such new transmission primitives into existing systems and MAC layers will no doubt carry an additional overhead we did not have to deal with in our proof of concept implementations. Exploring the tradeoffs of such integrations promises to be an interesting topic for future work.

Although we conducted all our experiments using the IEEE 802.15.4 standard, we believe these primitives to, in principle, be feasible in most wireless standards which use temporally separated symbols for modulation. We also expect this style of communication to be able to benefit many other scenarios beyond the two presented in this chapter.

The results of our solution to the chasm scenario undoubtedly leave room for improvements. One aspect worth further research is the use of multiple senders, ideally synchronized well enough to cause constructive interference in some subset of the receivers. Another promising direction is the use of forward error correcting codes to improve complete reception ratios at the cost of bandwidth. Such a scheme may also benefit the packet-in-packet primitive.

We see the main application of the packet-in-packet primitive in enabling high-priority messages to be sent and received at almost any time, in spite of ongoing lower-priority transmissions. This significantly reduces the latency these high-priority messages would otherwise incur, possibly accumulating further at every hop. Another imaginable application are data aggregation algorithms, using planned data insertions from many different nearby nodes to quickly form a single packet containing the aggregated data. To reduce the overhead of individual packets in this controlled scenario, one could shorten the synchronization word and omit or shorten the checksum footer.

In the remaining failure cases, often some of the preamble symbols are recognizable, but there is no reversible one-to-one symbol mapping. It may be interesting to explore the interplay of the chip sequences defined by IEEE 802.15.4 under varying degrees of desynchronization in future work. We believe that computing the original string of symbols no matter the time shift is generally not possible. However, by smartly choosing the symbol combinations used to carry the data of the injected packet, always retrieving all the data correctly may be possible. In the best case, this may even allow dropping the stringent synchronization requirement at the cost of perhaps doubling the injected packet's length.

5 Tempering Wireless Schedules

5.1 Introduction

Power control is the feature of some wireless transmitters to allow software to choose one of a selection of supported transmission powers for every transmission made. Even though this feature has become widespread, most wireless algorithms ignore its availability as it is often considered to introduce unnecessary complexity. We presented one example of a protocol benefiting from power control in Chapter 2. In this chapter, we will consider the problem of scheduling and examine how scheduling algorithms can be benefit from power control. As a baseline we use the RAND algorithm [63]. While simple in its greedy approach, RAND generally delivers good results. We chose RAND for both of those reasons, yet are able to improve upon many of the characteristics of the schedules it produces significantly.

More generally, we find ourselves in the scenario of harnessing a single frequency channel using the IEEE 802.15.4 standard, i.e., we focus on time and space division only. For medium access control (MAC) in this scenario, there are two general philosophies: opportunistic sending with backoffs and scheduling. With opportunistic sending, senders attempt to send as soon as a packet is ready to be sent. If the channel is busy, the attempt is repeated after a backoff period, until the transmission succeeds. Opportunistic sending incurs little to no overhead if traffic is low, but as traffic increases, it becomes increasingly inefficient and transmission delays become more and more unpredictable.

Scheduling on the other hand divides time into slots and assigns transmissions to slots such that all transmissions in each slot can be carried out simultaneously. Hence, collisions can be completely avoided and transmission latency becomes deterministic. This comes at the cost of flexibility as every update to a schedule may incur significant computation and dissemination overhead. Such updates are often necessitated by fluctuating traffic levels and changing environment conditions. Mechanisms to reclaim some of the flexibility have been proposed, such as slot stealing [42] and switching between opportunistic sending and schedules on-the-fly [65]. For ease of analysis, in this chapter we will assume the traffic pattern to be static and known in advance. Without loss of generality we can assume each traffic flow to have the same desired bandwidth and to flow from one node to an immediate neighbor. The quality of a schedule may then be defined by the number of slots required to service all traffic flows once: the shorter the schedule, the more often the schedule can be repeated in any given time frame, which directly correlates with the network's overall throughput.

By employing a model for the probability of a set of transmissions succeeding simultaneously, it is possible to efficiently predict viable slots offline without the need to try them first. This allows the creation of schedules optimized to make use of the medium to the fullest possible extent in each slot. The Signal-to-Interference-plus-Noise-Ratio (SINR) model is one of the most established models for this purpose, e.g., [26]. It posits that, if the ratio between a signal's power and the sum of the remaining signals' powers plus the noise floor clears a certain threshold at a receiver, then that signal may be correctly decoded at that receiver. There are different ways to obtain the variables required by this model, the gain matrix of the network and the noise vector: measuring the noise floor comes almost for free, but measuring link gains accurately usually requires actually sending packets. For the most accurate measurements, these packets should be sent without interfering traffic, which implies sacrificing regular transmissions slots. Less accurate but more subtle methods of measurement making use of knowledge about regular traffic failing or succeeding may be preferable in some circumstances.

The strength of each arriving signal – as well as the interference it causes – depends on two variables: (1) the gain of the link, and (2) its original transmission strength. Hence, there are in fact two fundamental questions that any scheduling mechanism needs to answer: who sends when, and who sends how strongly. In practice, the second question is often trivialized: having every sender use its maximum transmission power optimizes the

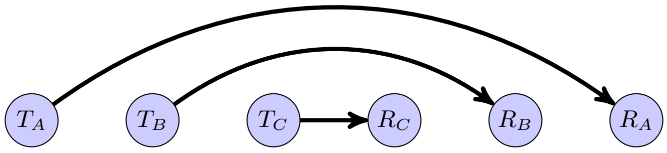


Figure 5.1: The "sandwich": an abstract geometric example of a scenario in which the use of individual transmission powers for each link reduces schedule length considerably. At equal transmission powers, all 3 links need to be scheduled in separate slots, while sensible power control allows scheduling all 3 links in the same slot.

signal quality at the receiver. This also reduces the task of computing a schedule to the combinatorics of finding a preferably small amount of slots able to serve all desired links.

However, power control is now widespread in hardware. Theory work in this area has long proven this option to provide significant benefits: in some situations significantly shorter schedules servicing the same traffic can be found [58]. Figure 5.1 shows a simple geometric example, in which power control allows shortening the schedule by a factor of 3. While such an example goes counter to intuition, log-distance path loss allows for such configurations when powers and distances are chosen carefully.

Unfortunately, realizing the algorithms and techniques proposed by theory is typically unrealistic. For example, in practice, the possible transmission power values each sender may choose from are usually bounded to a certain range and are not arbitrarily fine-grained. Yet, these concerns are not addressed by the theory community.

As a result, the real world usually ends up using simple heuristics to compute schedules while ignoring the opportunities offered by power control. For example, Z-MAC [65], as well as the RAND algorithm it is partially based on, only consider the maximum transmission power. We feel that, when choosing to take on the burden of the overhead associated with a scheduling approach, to not also incorporate power control is a waste.

In this chapter, we propose a practical algorithm to compute tempered (*hardened*) schedules harnessing power control: by using tempered (*lowered*) transmission powers, we can reduce the unneeded interference caused by transmissions. On average, this allows us to pack more links into a single slot than possible at maximum transmission power. Our results show that we are consistently able to schedule the same set of links in 20–25% fewer slots than possible without power control. That is equivalent to a 25–33% increase in overall throughput of the network.

Further, we can make use of our reduced interference levels to pack links into the schedule a second time at no additional cost. By scoring second schedulings of links a fraction of their first schedulings, we are able to quantify the additional value provided by a schedule without impacting its length. Schedules not employing power control also benefit from this ruling. However, our schedules profit more so, achieving a comparatively higher total score per slot value.

Additionally, we address the problem of the environment, i.e., the gain matrix, changing over time. This is a common problem in practice: doors may open or close, the temperature and humidity may change, people tend to move around. A schedule computed for one version of the gain matrix may turn obsolete within minutes as some links cannot be served anymore. By increasing the power of a link that was predicted to work but failed, we are in many cases able to salvage slots without having to move links to other slots or recomputing the schedule.

All our schedules are verified on a testbed of wireless sensor nodes deployed in an office building offering a typical noise background and dynamic environment. Experiments confirm that the improved schedules are just as reliable as traditional full power schedules. Both kinds of schedules suffer from up to 5% of the links failing to successfully transmit, depending on the chosen minimum SINR threshold for the admissibility of links. This is mainly due to these links being mispredicted on basis of a changed gain matrix.

5.2 Related Work

To avoid transmission collisions while also maximizing the possible throughput of a wireless network, numerous techniques and protocols have been proposed. The most important approaches can be grouped into time division (TDMA), frequency division (FDMA), space division (SDMA) and code division (CDMA). In this chapter, we focus on harnessing a single frequency channel and restrict ourselves to the IEEE 802.15.4 standard [32], i.e., we focus on time and space division. However, our solutions may be generalized to deal with multiple available frequency channels, and we believe the results to apply to a majority of today's wireless standards.

Various collision prediction models have been proposed and used in the past. The simplest models abstract the network into a communication graph in which every pair of nodes is connected by an edge if and only if they are able to communicate directly. A collision occurs if more than 2 neighbors are sending simultaneously. This model is used, for example, by scheduling algorithms such as RAND [63] and its derivatives. This model works best in homogeneous networks, i.e., networks in which all edge gains are very similar.

In practice, so called *physical models*, such as the SINR model, have been proven to be very accurate [2, 25, 26, 56]. These models typically require estimations of the signal strength fall-offs between two nodes, called *gains*. If the network geometry is known, these gains may be computed directly assuming, for instance, a log-distance path loss. This approach suffers greatly from irregularities in the environment as well as multi-path effects. Alternatively, each of the entries of the gain matrix may be determined through measurements. This carries with it a certain overhead – at the minimum, in several short slots, each prospective sender needs to send alone once. In this chapter, we valued the accuracy gained by the latter method more than the avoided overhead, as we were interested in estimating the maximum potential improvements possible through power control. However, based on the experiences we had with our experiments, we are able to suggest several ways to reduce the impact of this overhead in practice (see Section 5.8).

A framework for computing schedules avoiding collisions was proposed by Ramanathan [63]. In the same work, the RAND algorithm was introduced. Its basic idea is to iterate over the links to be scheduled in a random order and add them to the first slot able to accommodate them. If no slot is suitable, an additional slot is appended to the schedule. In spite of the algorithm's greedy nature, by taking the best outcome of multiple runs yields a consistently decent performance. One major advantage of RAND is its efficiency, which it gains by dropping all pretenses of attempting to guarantee an optimal solution or an approximation thereof. We will discuss this greedy bin packing algorithm in more detail in Section 5.5, Algorithm 2.

Rhee et al. [66] later provided a distributed version called DRAND with comparable performance to RAND. DRAND works by negotiating slot usage locally in each node's 2-hop neighborhood. This makes it especially well suited for larger networks as no global coordination is required. The popular Z-MAC protocol [65] uses DRAND as scheduling algorithm during phases of high load in the network.

Both RAND and DRAND assume transmission powers to be constant. In certain denser networks, choosing a different global transmission power (less than the maximum power) may already drastically improve these algorithms' performance. However, actually adding a transmission power variable for every link to these algorithms is a problematic proposition as "2-hop neighborhood" then turns into a rather fuzzy concept. ElBatt et al. [14] proposed enhancing schedules with power control similar to our work. However, they did not perform any practical evaluation of their heuristics.

In the last decade, power control has also received a lot of attention from the algorithmic networking community, starting with the work of [56].

The first algorithmic result to schedule an arbitrary link set was by Moscibroda et al. [58]. This result was soon superseded by the first approximation results [4, 24]. These early approaches involved (directly or indirectly) partitioning links into length groups, which results in performance guarantees that are at least logarithmic in Δ, the so-called link diversity, i.e., the ratio between the longest and the shortest link [5, 10, 24, 27]. However, link scheduling cannot be solved optimally, as NP-hardness was established in [24]. This is the reason why our algorithm is also heuristic in nature, and does not try to solve the problem "optimally". A breakthrough paper on the theory side was by Kesselheim [35], who discovered the first scheduling algorithm that uses power control with logarithmic approximation guarantees.

On the positive side, theoreticians established that – in theory – power control is in fact powerful. There are instances with n links that can be scheduled in a single slot with power control, while, without power control, all n links need to be scheduled sequentially [58]. This result can even be generalized to the case when we compare arbitrary power control to so-called oblivious power, where the power of a link only depends on the length of the link [17]. However, the networks which yield these bounds are extremely cooked up, they can only be witnessed in a "theory zoo" but certainly not in the wild. Sometimes the lengths of links must grow *doubly exponentially* [27, 28] in order to prove the theorem.

There has been some work that showed that power control can be effective in practice. For example, Lin et al. [47] proposed a feedback mechanism to balance out the quality fluctuations of individual links. In contrast to our work, these earlier papers again do not consider realistic networks. In an early effort, Moscibroda et al. [57] showed that power control can increase the throughput in a wireless sensor network with link diversity $\Delta = 3$ by a factor of 3. In fact, they proved the "sandwich" situation shown in Figure 5.1 to be feasible in practice.

5.3 Link Model

The link model is concerned with providing accurate and up-to-date link state information, such as a link's current gain and the noise floor at a receiver. It forms the basis for estimating link set success probabilities in the prediction model. While the physical locations of the nodes are readily available to us, we do not attempt to infer link gains from such data, as such approaches are fraught with inaccuracies, especially in complex indoor environments. Instead, we perform measurements using the same devices we will later use to verify the viability of schedule slots. Such a method is

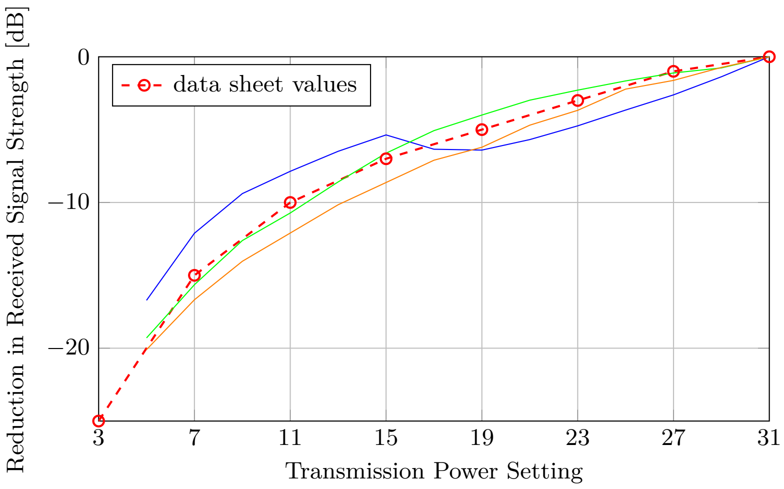


Figure 5.2: Relative received signal strengths plotted against the different transmission powers for a few different links.

not only convenient to integrate into the regular operation of just about any existing wireless network, it also yields measurement values which should in most cases match the performance in regular operation later on.

First, we attempted measuring only the achieved received signal strengths at the highest possible transmission power settings for every link. As a reduction of the transmission power by some factor x should also reduce the received power by the same factor x, we hoped to be able to infer the received signal strengths for other transmission power settings from this one measurement per link. Unfortunately, in many cases, the different transmission power values specified in the data sheet differed from what was measured in practice. Figure 5.2 shows the reduction in measured received transmission power compared against the case of maximum transmission power (on the vertical axis) plotted against the used transmission power settings (on the horizontal axis). The dashed line indicates the values as given by the CC2420's data sheet [75]. Each of the other lines represents the average values measured for a different single link repeatedly sampled over the span of 1 hour. The shown links were chosen to showcase some of the different kinds of discrepancies we encountered.

Figure 5.2 makes it clear that measuring the link gains only at the maximum transmission power and extrapolating the gains for other transmission powers is certain to incur up to 1–3 dB of error on average, i.e., up to a factor of 2 in absolute signal strength. As these curves are somewhat characteristic for individual links, often changing only marginally if sampled on successive days, one option is to measure them once and then only update the gain of the maximum transmission power frequently. As can be seen from the varied shapes of the curves, adding just one or two additional measurements is not sufficient to accurately determine a link's curve. Hence, in the interest of accuracy, we chose the most detailed option: measuring all points of the curves every power test. However, as that requires 15-times as many measurements as the extrapolating option, we expect that one would try to find a more reasonable tradeoff in practice.

This essentially adds a third dimension to our gain matrix, which is thus now parameterized by sender, receiver and transmission power. This link model turned out to work well at night, i.e., when the environment was almost static as the office building was empty. In contrast, during office hours, even a few fluctuating links caused a large portion of the generated schedule slots to be too unreliable for use. This is due to the fact that underestimating a link often ruins all slots it appears in: not only is the link afforded unneededly little interference at its receiver, but the other receivers also expect the link to cause far less interference than it actually does. Figure 5.3 shows the received signal strength over the course of 1 hour during office hours for a few select links. Signal strength jumps on the order of 5 dB in either direction between 2 or 3 consecutive measurements (2–4 minutes in our setup) are not uncommon.

To deal with these fluctuating links we decided to have the link model not only keep the most recently measured gain matrix, but also keep several older versions in memory. We now offer 2 values for each link to the prediction model: a signal gain and an interference gain. The signal gain of a link is the lowest of the gain values for that link amongst the kept gain matrices; analogously, the interference gain is the highest value. This pessimistically drops the peaks in Figure 5.3 for the signal value, and fills the dips for the interference values, effectively smoothing out the fluctuations. As retention time for old gain matrices we found 15 minutes to be reasonable in our case. This value should be chosen according to the power test frequency such that recent peaks and dips are likely to be found and remembered.

In addition to link gains, the link model also collects noise floor measurements for every node. These values serve two purposes: to deduct the noise component from measured received signal strengths, and for use in predictions in the SINR formula as additional interference source. In our testbed the noise is negligibly small (< -95 dBm) for most nodes, but not for all. Further, not subtracting the noise component from signal strengths can accumulate to a large error in predictions for slots with a large number

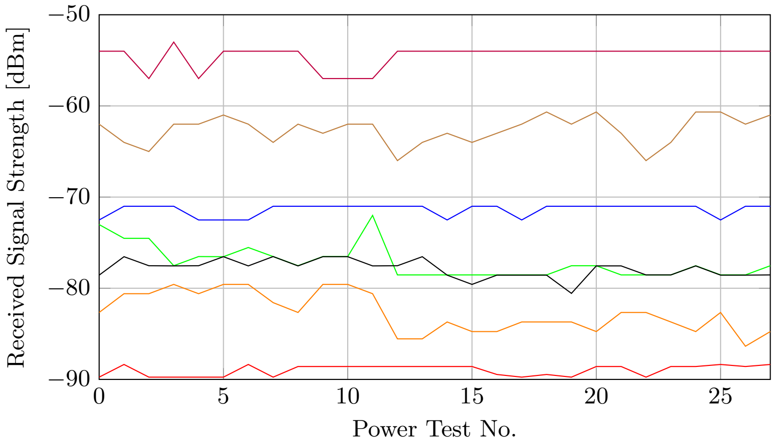


Figure 5.3: Received signal strength of 7 different links plotted over time during office hours. The links were chosen to showcase both stable and fluctuating behaviors.

of concurrent senders.

There are a few cases in which we further sanitize our data. For one, received signal powers may sometimes not increase monotonically with increasing transmission powers. This may occur either due to measurements fluctuations or in some rare cases due to strange link characteristics (see Figure 5.2). As our scheduling algorithms make the simplifying assumption that increasing transmission power also increases received power, we smooth out these bumps. Another data anomaly that may pose a problem are gaps of "neither signal nor interference on this link" in the data caused by power test packets occasionally getting lost through external interference. Such anomalies are easily detected and can be repaired by inserting one of the lower recently measured values for that link and transmission power.

The CC2420 does not in fact directly offer a signal strength measurement, but rather a received signal strength indicator (RSSI), which is required to be accurate to ± 6 dB by the IEEE 802.15.4 standard. Fortunately, it is far more accurate than what is prescribed: applying a simple constant offset as proposed by the CC2420 manual one can already obtain a value off by at most 3 dB. However, the RSSI values are in fact subject to a not completely injective piecewise linear function describing the relationship between RSSI and true received signal strength. The manual provides a plot of this function as a vector graphic. We wrote a script that extracts the exact data points from this graphic, and manually smoothed out the non-injective parts of the function to be able to create a reverse mapping. Chen et al. [7] explored this mapping in far greater detail, but our simple approach seemed to work well enough in practice.

The set of transmission power settings we use in all experiments in fact exceeds the set described in the manual: the manual lists 8 possible settings (3 through 31 in steps of 4). We use 7 additional setting values, each between two of the listed values, for a total of 15 power setting values (3 through 31 in steps of 2). Considering more power settings naturally improves the flexibility of algorithms employing power control. We tested the remaining possible power settings (the even values) but found them to almost always result in power outputs identical to adjacent power settings. These undocumented power settings were also examined by Maheshwari et al. [51]. However, we were unable to reproduce any successful transmissions using power settings below 3.

5.4 Prediction Model

A (schedule) slot can be defined as set of tuples of transmitting node, receiving node and transmission power. In our scenario we expect every node to appear at most once in a slot, either as sender or as receiver, but this assumption may be weakened: When considering broadcast transmissions, multiple tuples may have the same sender. Further, full-duplex radios allow a node to appear in a tuple as receiver even if it also appears in tuples as sender.

Given a slot, the goal of the prediction model is to make as accurate of a prediction of the outcome as possible – without trying the slot in practice. In particular, we are interested in whether each of the links in the slot is likely to work, i.e., have a success probability $p > p_{threshold}$. We call such slots *feasible*.

For our purposes we chose the SINR model, fed with measurements of every link's received signal power at every transmission power (the link model).

The slot is given by link set $L = \{l_1, \dots, l_k\} = \{(t_1, r_1, p_1), \dots, (t_k, r_k, p_k)\}$, the lowest and highest (see Section 5.3) recent received signal powers for a link l_a are denoted by $s(l_a)$ and $i(l_a)$ respectively, and the noise floor at node r by $n(r)$, then our SINR model states that a link $l_a \in L$ is able to transmit successfully if and only if the following equation holds at the receiver:

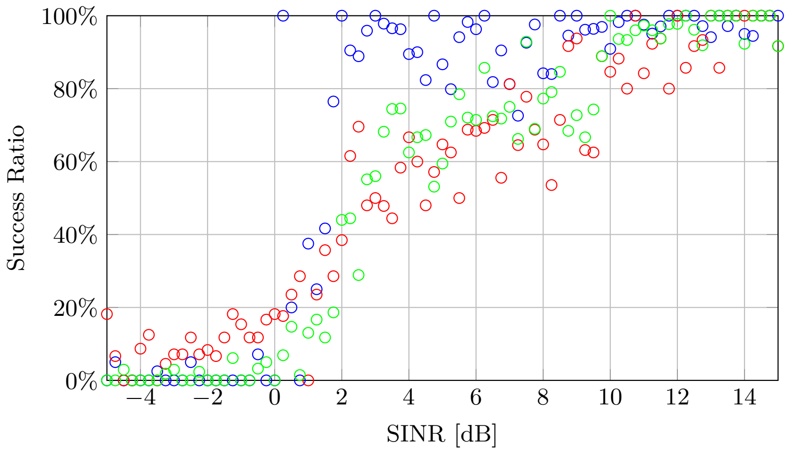


Figure 5.4: Scatterplot of link success ratios split into SINR buckets of 0.25 dB each. Colors indicate which of the 3 experiments the value came from.

$$SINR(l_a) = \frac{s(l_a)}{\sum\limits_{\substack{l_b \in L \\ l_b \neq l_a}} i(l_b) + n(l_a)} > \beta$$

Here β is the so called *SINR threshold*, which needs to be cleared by the *signal-to-interference-plus-noise-ratio* (SINR). Unfortunately, in reality link success is not as black and white as the formula suggests: between a "high enough" and a "too low" SINR there exists a transition region in which the packet reception ratio gradually tapers off.

To decide on a fitting β we conducted an experiment, in which we computed a schedule at the start and then tested that schedule's slots repeatedly for about an hour, ignoring changes in the environment. We then divided the results for all tested links into buckets based on the SINR we computed for the link beforehand. Figure 5.4 plots the ratio of successful links for each bucket, each bucket 0.25 dB in size. The results from 3 experiments, each with a different schedule, are shown, in a different color each. The experiment in blue used different parameters for the schedule computation, which should not be of interest for this discussion.

While in the blue test run the success ratio rather quickly increases from around 0% to 80% for SINR values from 0 dB to 2 dB, the other test runs experience a significantly wider transitional region. The red test run even consistently reaches a packet reception ratio of almost 10% at SINR values below 0 dB, i.e., when the quotient is below 1 as the interference and noise was measured to be stronger than the signal.

We explored computing a piecewise linear function mapping SINR to success probability. However, as this mapping appeared to vary depending on external parameters (such as the slots used and perhaps the environment), especially in the interesting area of 2 dB $< SINR <$ 10 dB, any mapping function procured from such experiments was doomed to be too inaccurate to be used for a subsequent experiment. Our main takeaway from these experiments is that the SINR model and/or our measurements are too inaccurate as to be able to guarantee a successful reception at low values of β.

For computing schedules, we decided that we were interested only in slots, in which all links had a success probability of at least 80%. Based on our findings above we forewent a SINR-to-probability mapping function and settled for using hand-picked constant values for β. As we are ultimately interested in computing schedules that are viable in practice, we tuned the model parameters to make the model decide pessimistically, i.e., focus on avoiding false positives (links predicted to work not working) at the cost of an increased amount of false negatives (links predicted to not work working). For exploiting links with a success ratio in the transition region we refer to other works specializing on this topic such as the work by Maheshwari et al. [51].

Reasonable but pessimistic values for β lie in the range of 6–12 dB: for a link to successfully invoke the capture effect to be heard over the interference from the other senders in the slot requires exceeding the their combined power by 2–8 dB, depending on various factors such as the number of interfering signals [51, 78] (see also Section 3.7.2). In addition, we have seen that the link qualities can easily spontaneously fluctuate by 2 dB in the most static phase of our environment, up to 5–6 dB in an active phase.

5.5 Tempering RAND

The input of a scheduling algorithm is a set L of links representing the traffic to be served. Without loss of generality we can assume all traffic demands to be between two neighboring nodes and of unit size: to solve these more general cases, we can split every traffic demand into multiple traffic demands of unit size (L now becomes a multiset) and split traffic between non-neighboring nodes into multiple single-hop traffic demands using a suitable routing algorithm. For our experiments we chose L to include all

Algorithm 2: RAND

```
Data   : Function Feasible(S)
Input  : Links L
Output: Schedule (set of slots (sets of links))
C ← {}
while L ≠ {} do
    l ← random element from L
    L ← L \ {l}
    foreach S ∈ C do
        S' ← S ∪ {l}
        if Feasible(S') then
            C ← C \ S ∪ S'
            break
    if l could not be added to an existing slot then
        C ← C ∪ {{l}}
return C
```

feasible links, i.e., all links that are able to successfully transmit alone at maximum transmission power.

A valid output of a scheduling algorithm is a set C of sets of links called slots. The union of all slots must be L, and all slots must be *feasible*, i.e., all links within the slot must be able to transmit successfully concurrently. In the case of a scheduling algorithm not using power control, it is implicit that all links are to be used at full transmission power. When using power control, the algorithm additionally needs to return the powers to be used for the individual links within each slot.

The classic RAND algorithm was a special case in the general scheduling framework proposed by Ramanathan et al. [63]. Pseudocode for RAND is given by Algorithm 2: the input links are greedily added to existing slots in a random order. If no existing slot is able to accommodate the next link, a new slot is added. The algorithm's convincing features are its simplicity and efficiency in practice, defying the combinatorial nightmare that NP-hard scheduling [24] poses. RAND's consistency can be slightly enhanced by running it multiple times and choosing the best output among the instances.

As our algorithm builds upon the basic structure of RAND, it can reap the same benefits: links are considered in a random order and greedily assigned a slot. All links assigned to a slot also carry a transmission power setting. Initially, when the first link is added to a slot, its power setting is set to the minimum for which the link is feasible. When another link is added to a slot, the existing links of the slot may need to raise their power settings

Algorithm 3: PowerRAND

```
Input   : Links L
Output: Schedule with transmission powers (set of slots (tuples of
        sets of links and power mapping))
C ← {}
while L ≠ {} do
    l ← random element from L
    L ← L \ {l}
    foreach (S, p) ∈ C do
        S' ← S ∪ {l}
        p' ← FindPowers(S')
        if p' ≠ ∅ then
            C ← C \ (S, p) ∪ (S', p')
            break
    if l could not be added to an existing slot then
        C ← C ∪ {({l}, FindPowers({l}))}
return C
```

Algorithm 4: FindPowers

```
Data    : Available transmission power settings P
Data    : Function Predict(L, p) returning working links
Input   : Links L
Output: Map: link → transmission power; or ∅
foreach l ∈ L do
    p[l] ← min P
repeat
    change ← False
    prediction ← Predict(L, p)
    foreach l ∈ L do
        if l ∉ prediction then
            if p[l] = max P then
                return ∅
            else
                p[l] ← min {x ∈ P | x > p[l]}
                change ← True
until change = False
return p
```

to account for the interference caused by the new link. If a link's power settings cannot be raised far enough to enable the new link, the new link cannot join the slot as it would otherwise become infeasible. We abstracted the process of attempting to find an agreeable set of power settings for a potential updated slot into a subroutine called `FindPowers`. The complete pseudocode listings are given by Algorithms 3 and 4.

After computing a schedule with our algorithm, it may be beneficial to optimize the individual slots: as each slot's power settings are as low as possible to allow for as much room as possible for potential further links, the SINR values of the individual links are likely not as high as they could be. Now that we know that no further links will be added to the slot, we may gain an overall "better" slot by raising the power settings. We define a set of power settings for a slot to be *better* than a different set of power settings, if the minimum SINR value it induces amongst the links of the slot is higher. In a stable environment, such increases in SINR values are not worth much (at least with the modulation prescribed by our chosen wireless standard), but as we are dealing with an environment in which links may suddenly and abruptly change, delivering more interference or a weaker signal, using any available additional buffer is prudent.

Exhaustively searching for the optimal power settings for a slot is not tenable in practice, especially for slots containing 5 or more links. Instead, we start at a random position in the space of reasonable power settings, i.e., we set every power to a value between the minimum suggested by the scheduling algorithm and the maximum possible, and then follow the gradient of the *quality* to its local maximum, where the *quality* of a set of power settings is defined as the resulting minimum SINR value among the links. We repeat this process for 50 different random starting values and use the local minimum with the best quality we found. Comparing the achieved quality to that of the exhaustive search for a several random samples, we find our gradient method to get sufficiently close to the optimum in most cases.

Without specifically giving scheduling algorithms an incentive to ensure all slots are utilizing the available medium well, the schedules they produce may contain slots containing only one or two links – not because these links make scheduling additional concurrent links impossible, but because all links that could be added are already being served in other slots. Such schedules are wasteful in practice.

Hence, to perform our experiments with realistic schedules, we decided to allow the scheduling algorithms to schedule any link a second time at a reduced reward. More concretely, we implemented the following scoring function for a schedule C that is supposed to cover L:

$$Score(C,L) = \begin{cases} 0 & \text{if } \exists l \in L : |\{S \in C \mid l \in S\}| = 0 \\ \frac{\sum\limits_{l \in L} LinkScore(C,l)}{|C|} & \text{otherwise} \end{cases}$$

$$LinkScore(C,l) = \begin{cases} 1+\varepsilon & \text{if } |\{S \in C \mid l \in S\}| \geq 2 \\ 1 & \text{if } |\{S \in C \mid l \in S\}| = 1 \\ 0 & \text{otherwise} \end{cases}$$

In practice, we chose $\varepsilon = 1/4$, i.e., 4 secondary schedulings are equivalent to 1 primary scheduling. This scoring approach can be seen as a generalization of the previous quality measure (the schedule's length) using an arbitrary *LinkScore* function: when setting $\varepsilon = 0$, the resulting schedule score would induce the same quality ordering on schedules as their length (or rather: shortness).

We extended both RAND and PowerRAND to create schedules targeting the new scoring function. The extension applies to both algorithms the same: Once the set of remaining links (L) is depleted, we replenish it once and continue the algorithm. Hence, at the end, every link is contained in the schedule twice. We then return a schedule consisting of the oldest k slots for the k that maximizes the schedule score for those slots (this requires the slot set C to preserve an ordering, which is not reflected by the pseudocode given above). If additional link score is awarded for a third scheduling of a link, one would need to replenish L a second time, and so on. We found the difference between rewarding only the second scheduling and additionally rewarding a third scheduling to be negligible in practice. However, the above algorithm and scoring achieve the desired effect of smoothing out the produced schedule such that all its slots appear to utilize the medium reasonably efficiently. Note that, even though we allow for our algorithms to add slots in this process, the choice of the scoring function makes the circumstances required for a slot addition virtually impossible.

Power control does not only afford us more efficient schedules, it also allows us to react swiftly to a slot ceasing to work, for example, due to changes in the environment. More specifically, if one link within a slot fails to transmit repeatedly, one may edit the schedule on-the-fly and increase that link's transmission power for that slot by a notch. This technique particularly benefits from larger minimum SINR values (larger β) as the additional interference caused by the attempt to repair one link is unlikely to disturb the other links. As we are dealing with a particularly active environment in our scenario, we found this technique to be helpful in offsetting the fluctuations and thus prolonging the applicability of a schedule.

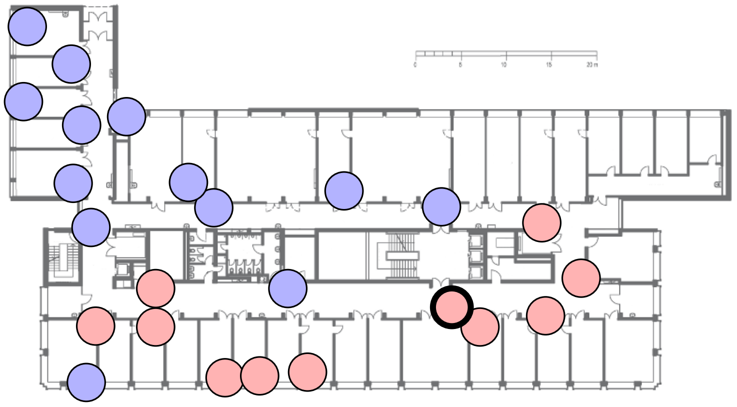


Figure 5.5: The floor plan and node layout of the wireless sensor node testbed we conducted our experiments on. We tested a smaller and a larger set of nodes. The smaller one only includes the nodes in red, the larger one also includes the nodes in blue. The node with the thick border was used as starting node for synchronization waves for both node sets.

Another attempt we made to fight the fluctuating link qualities was to add slots of a certain minimum size which worked well to a whitelist, to be automatically grandfathered in into schedules generated in the future. Similarly, we added slots that turned out to not work as expected to a blacklist, forbidding that combination of links to be used by a future schedule. Neither of these methods were particularly successful: whitelisting, while it may find and preserve particularly good link combinations by chance, is not immune against changes of links within the whitelisted slots, and blacklisting is a hopeless proposition in face of the myriad of possible combinations of links.

5.6 Experiment Setup

All our experiments were conducted on FlockLab with TelosB motes. As the testbed consists of installations on the walls of an office building, the activity of the people using this office building, such as office doors being opened or closed and people moving about, creates a dynamic environment. Such a real world scenario allows us to experience and address problems usually not present in lab setups. Figure 5.5 shows the floor plan and node layout of the testbed. All nodes were located on the same floor of the building.

To ensure that possible collisions in a schedule slot are not avoided due to a lack of synchronization, we periodically synchronize all nodes to a previously chosen central node of the network, which is at most 3 hops from every other node (see Figure 5.5). As synchronization period we found 11 seconds to be an adequate value for our experiments. This keeps our synchronization error below ± 2 µs at all times.

Further, we take special precautions that no node misses its scheduled transmission time: 30 ms before the transmission time is reached we start a series of busy waits on the separate parts of the target timestamp (see Section 3.6). We find that over 99.9% of our transmissions are started within ± 50 µs of the correct local time. For reference, the packets we transmit are about 2 ms in length. Hence, even if added up these two error sources and included the propagation delay of up to ± 0.34 µs (accounting for up to 100 meters difference in path lengths), any two packets sent within in the same slot are guaranteed to temporally overlap at the receiver for at least 2000µs$-$52.34 µs or over 97% of the packet. We deem this sufficient for obtaining meaningful results about the feasibility of schedule slots.

To coordinate the experiment, we make use of a feature offered by FlockLab which allows us to remotely receive and send serial input to nodes via a TCP connection each. We have a coordinator script running on a regular x86-64 desktop machine, connecting to each of the nodes to issue only very loosely synchronized commands. Such a central coordinator is an uncommon feature in real world wireless networks. However, it vastly simplifies result gathering, timely schedule computation and command dissemination. These tasks are not the focus of our work – we are more interested in a proof of concept and measurement of the physical feasibility of schedules employing power control. Our current setup essentially has the centralized control flair of popular software defined networks.

Our experiments contain 3 kinds of phases of operation:

- **Synchronization phases** to align node clocks to be able to enforce packet collisions as described above.
- **Power test phases** to obtain measurements for the values of the gain matrix.
- **Set test phases** to try out simultaneous use of a certain set of (sender, receiver, transmission power) triplets, i.e., test a schedule slot.

Each of these phases is subdivided into a number of *time slots*, each lasting 1/8 of a second (not to be confused with *schedule slots*). *Synchronization phases* start by the central node of the network being woken by the coordinator. This node's clock will serve as the global reference clock amongst the nodes. Woken nodes will broadcast their time at maximum transmission power at the start of each slot with a probability of 1/3, or with a probability of 1 if have not transmitted since being woken. Nodes implicitly gain a parent node – the node whose broadcast woke them. In this phase, they will continue accepting timestamps from their parent node, should it transmit again. To minimize the error incurred each hop, we apply the techniques detailed in Section 3.5. Each synchronization phase lasts for 10 time slots, i.e., 1.25 seconds.

In a *power test phase*, one node after the other will transmit once at each available transmission power level. The packets being sent contain both the sender's ID and the transmission power that was used. Whenever a node successfully receives a packet during this phase, it will note sender and transmission power as well as the received signal strength indicator (RSSI) and pass them on to the coordinator via serial output. I.e., once a power test phase is complete, the coordinator will have collected sufficient data to create a fresh complete snapshot of the gain matrix. Also in this phase, noise floor levels are measured several times by all nodes. To avoid outliers due to other transmissions in the environment, all but the lowest of the measured noise floor values are discarded. To construct the gain matrix, the coordinator first converts the RSSI values into more accurate received signal strength values (see Section 5.3) and then subtracts the measured noise floor values for the respective receivers. We run power test phases once every 2 minutes.

In a *set test phase*, the coordinator chooses a slot of a computed schedule and puts it to the test in practice: all transmitting nodes of the slot are informed of their duty as well as their transmission power for this schedule slot. Due to the loose synchronization between coordinator and nodes, the coordinator picks a time slot which lies 2 slots (0.25 seconds) in the future to ensure all transmitting nodes can prepare and agree on the same time slot. All non-transmitting nodes record which sender's packet they receive – if any at all – and report this to the coordinator via serial output. Set test phases are executed whenever no other phase is pending.

Experiments start out with 4 synchronization phases followed by 4 power test phases. The multiple synchronization phases serve to initialize the linear regression for clock drift compensation. The multiple power test phases aim to detect heavily fluctuating links early, see Section 5.3. Afterwards, the coordinator computes a schedule according to one of the algorithms listed in Section 5.5 and then tests this schedule. Each schedule is tested by performing set test phases for each the slots of the schedule in a round-robin fashion, repeated 4 times. If a link in a schedule slot is found to unexpectedly fail multiple times early, it may be upgraded in strength and its slots repetition count may be reset, see Section 5.5. Once a schedule has been evaluated, a new one is computed, based on the data from the newest power test phases.

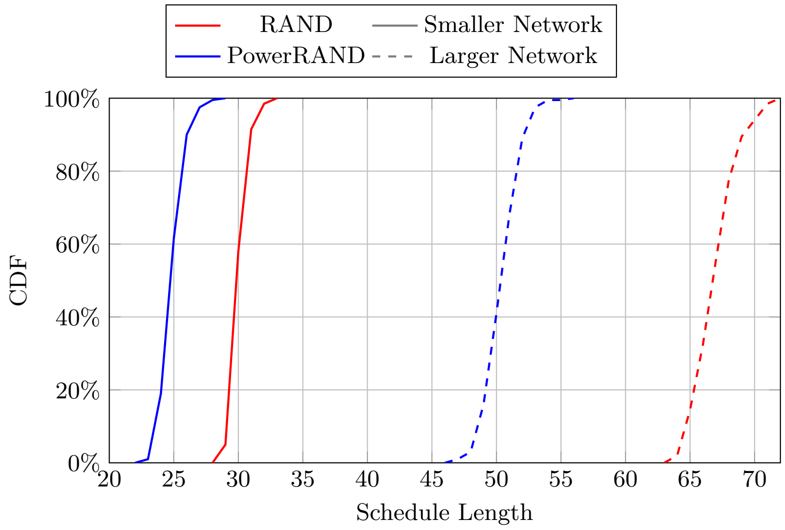


Figure 5.6: The distribution of schedule lengths.

Overall, our experiments were consciously not designed to measure the maximum achievable throughput in practice directly. Instead, we concentrated on ensuring (1) that we have the most up-to-date environment information available to be able to compute high-quality schedules, and (2) that we evoked as many packet collisions as possible during schedule evaluation through tight node synchronization. We believe that the results from this set of parameters to be the most meaningful in determining the improvements schedules with power control can achieve.

5.7 Results

We ran RAND and PowerRAND each 200 times on the same link and prediction models (snapshots taken from a real test run) to obtain 200 comparable schedules. When interpreting the plots of the distributions of the various properties of these schedules, keep in mind that in practice one would typically run the algorithm at least 5–10 times to obtain a consistently decent result. We present the results for 2 different sets of nodes: a small set of only 11 nodes (59 viable links) and a larger set of 24 nodes (188 viable links). All computed schedules were to include all viable links. In all cases, we used $\beta = 9$ dB as cutoff parameter for the prediction model. Figure 5.5 show the

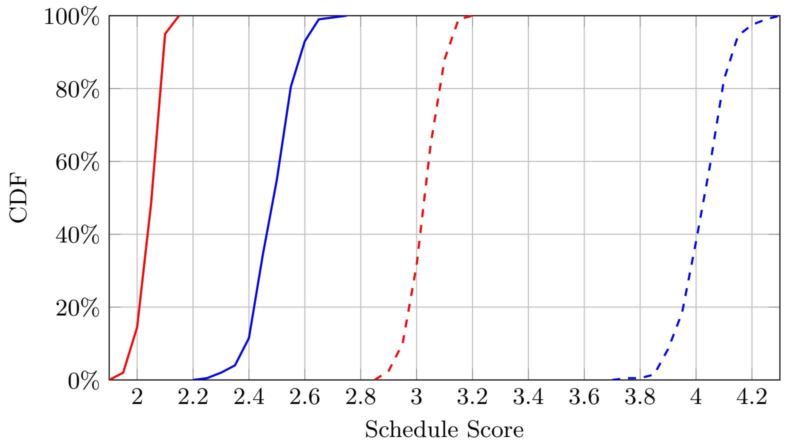


Figure 5.7: The distribution of schedules scores.

locations of the node sets within the office building.

Figure 5.6 shows the distribution of the schedule lengths (number of slots in each schedule), the classic measure of schedule quality. For the smaller node set, we observe PowerRAND achieving a reduction in schedule length by around 20%. For the larger node set, the reduction reaches just over 25%. In none of the 200 schedules generated for any of the traces the new scoring incentive for links scheduled a second time resulted in longer schedules. The distributions also appear to be rather stable, i.e., taking the best result of a small number of algorithm iterations is likely to produce a schedule from the "lower" 20–30% of the plot.

Figure 5.7 shows the distribution of the schedule score achieved as defined in the previous section. Intuitively, the scores represent the number of links which were served by a slot on average, with links being scheduled a second time only counting 1/4 as much as a link being scheduled for the first time. For both node sets, the relationship between the two algorithms' values reflect that of the schedule lengths: the increase by 25% corresponds to the schedule length reduction by 20%, and similarly for the larger node set. Save for the additional 1/4 scores, these values are exactly proportional to the reciprocal of the schedule length. Because even with the addition of the score from links being scheduled a second time, the total scores do not deviate from this relationship much, which implies that allowing links to be scheduled a second time appears to have a rather small overall impact.

However, the scoring mechanism still succeeds in incentivizing the algo-

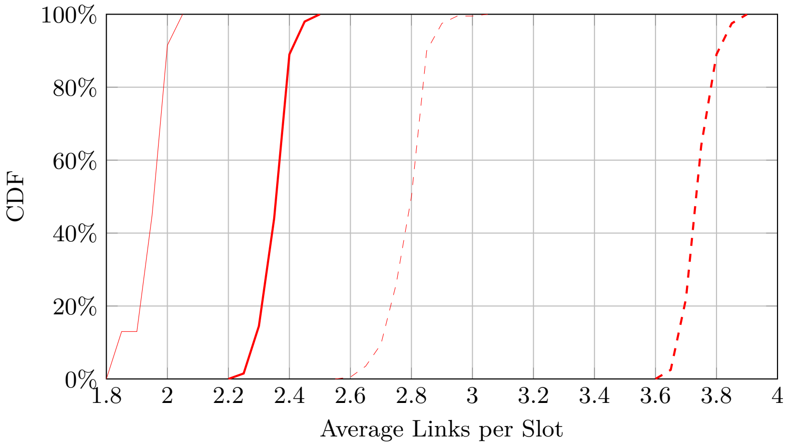


Figure 5.8: The distribution of the average links per slot for RAND without (thin traces) and with (thick traces) the new scoring.

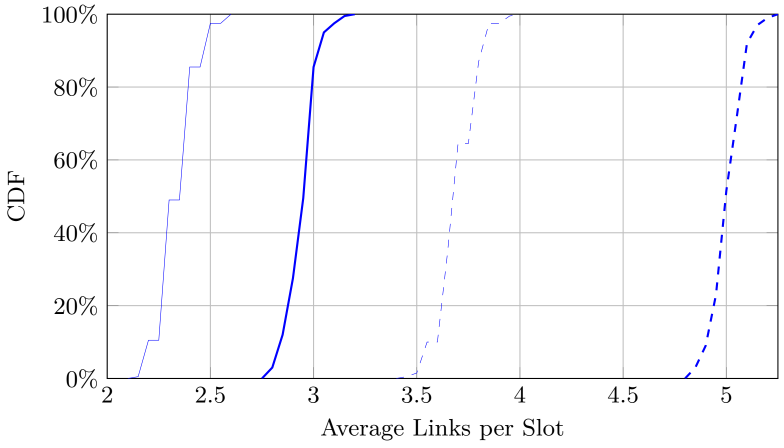


Figure 5.9: The distribution of the average links per slot for PowerRAND without (thin traces) and with (thick traces) the new scoring.

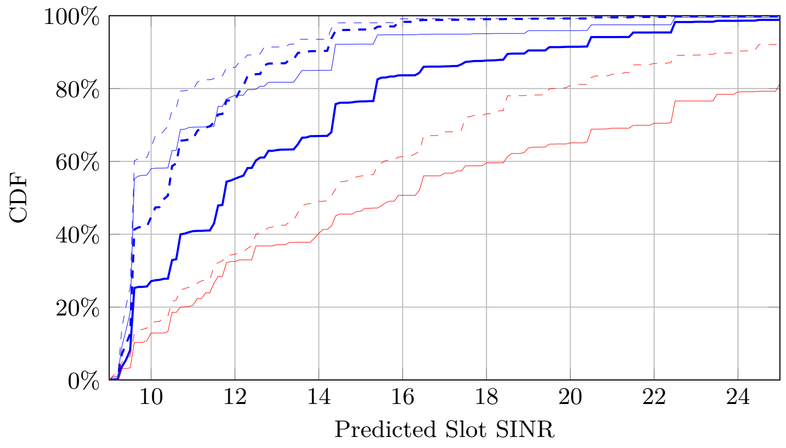


Figure 5.10: The distribution of the SINR values. The thinner traces show the values before slot optimization, the thicker traces the values after.

rithms to fill their slots up: Figures 5.8 and 5.9 show the distributions of the average over the number of links in a slot. For comparison, the thinner traces in this plot each show a separate set of 200 schedules (colors and dashes as before) that were computed without the new scoring incentive (but still use the same link and prediction models). Both algorithms appear to achieve a similar increase in links per slot, with the PowerRAND doing slightly better on the smaller node set. The larger node set profits more from this (35% versus 20–25% for the smaller set), which was to be expected due to the larger flexibility stemming from an overall larger number of links and nodes.

Figure 5.10 demonstrates the improvement in SINR values that is achieved through the optimization of individual slots after completion of the schedule computation. This optimization does not apply to regular RAND as it assumes the use of power control. The thinner traces in this plot correspond to the values before the optimization, the thicker traces conversely to after the optimization. No link is scheduled with a SINR value lower than 9 dB as this was precisely the cutoff we configured for the prediction model. For 20–40% of links the optimization appears to have little to no impact. These links are mostly part of already tightly packed slots without room for power adjustments. For the remaining links the improvement is on the order of 1–2 dB on average. While not a lot, in turbulent environments, every little additional stability translates into fewer mispredictions and longer lasting

Node Set	Time of Day	Power Control	β	Link Failures
Small	Night	Yes+Readj.	6 dB	9.0%
Small	Night	Yes+Readj.	9 dB	7.3%
Small	Night	Yes+Readj.	12 dB	0.4%

Table 5.1: Failure rate of links in live experiments using generated schedules with different values for β (the minimum SINR value parameter in the prediction model). For all schedules PowerRAND with on-the-fly slot readjustments was used. All data stems from experiments on the smaller node set.

schedules.

Also visible in Figure 5.10 is how well the each of the scheduling methods is able to utilize the given medium: as was preordained, the approach without power control tends to "waste" a lot of the medium by using an unneededly high transmission power, reflected in unneededly high SINR values here. Further, all scheduling methods appear to achieve lower SINR values in the larger network. This can be explained by the increased opportunities for heavier parallelism.

To verify that our algorithm can be used in a live scenario, we conducted the experiments described in Section 5.6. Each test run lasted for 1 hour, during which multiple schedules were generated from the collected link state data and then tested. Table 5.1 shows the different amounts of link failures encountered for a set of experiments ran with different values for the SINR cutoff β. All experiments used PowerRAND and applied on-the-fly slot readjustments when a link failure was first encountered, and were run on the small node set at night (low environment activity). We consider a link failed if it failed to transmit in more than 1 of the 4 trials we hold for each slot. Even with such a small number of trials, we deem it reasonable to consider links failing 2 out of 4 times to certainly be unable to meet our stated goal of only scheduling links with a minimum reliability of 80%. The results show that the choice of β significantly impacts the system's overall performance. In our case, even β =9 dB suffered from a very high failure rate.

To further analyze the cause of these seemingly bad schedules we aggregated the links that failed by the SINR values that our models predicted for the links at the time of the schedule's evaluation. Figure 5.11 shows the distribution of these links for the same experiments as used for Table 5.1: the 6 dB experiment in orange, 9 dB in blue and 12 dB in green. Note that the plots show a few scheduled links below the respective experiment's threshold due to rare cases in which the link model reports different values

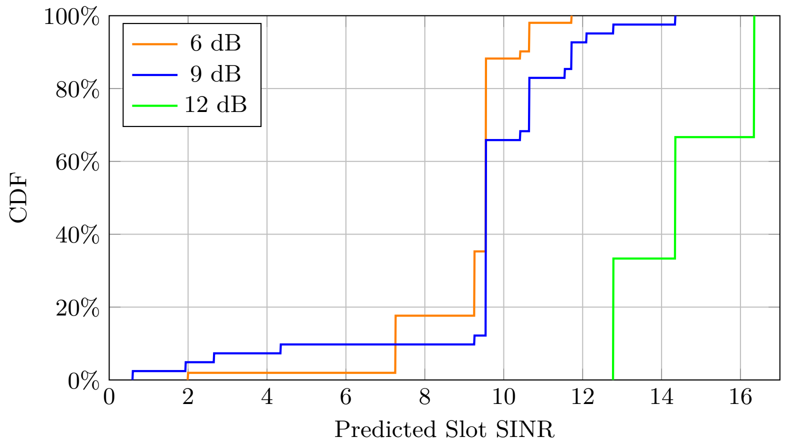


Figure 5.11: Distribution of the predicted SINR values of the links that failed in the experiments listed in Table 5.1.

at the time of schedule generation and at the time of schedule evaluation (which lie a few seconds apart). Save for these exceptions, most of the link failures occur in the range of 9–11 dB. For the case of $\beta = 9$ dB this is not very surprising, as we'd expect the weakest links in a schedule to fail the most frequently. However, we are unsure why most links failed around 9 dB in the case of $\beta = 6$ dB. In the 12 dB experiment only 3 links failed, all well above the 9 dB mark. In conclusion, we find that even at night, when the environment should fluctuate only very little, links with predicted SINR values of 9 dB, in some rare cases even as high as 16 dB, may fail.

In another set of experiments we compared the link failures incurred by RAND to those by PowerRAND. Additionally, we distinguish PowerRAND with and without the technique of on-the-fly slot readjustments (see Section 5.5). This technique does not apply to RAND as it requires power control. Again, all experiments were run at night and on the smaller node set. The results are shown in Table 5.2.

Interestingly, at night, RAND suffers fewer failures than vanilla PowerRAND. This is likely due to the inherent overprovisioning of transmission power in RAND, and, conversely, due to PowerRAND cutting more corners. Once we allow PowerRAND to adapt transmission powers within a slot to react to failing links, in the experiments ran at night, PowerRAND's link failure rate falls to only 0.2% (0.4% in the test run with identical parameters shown in Table 5.1). During daytime, both RAND and PowerRAND suf-

Node Set	**Time of Day**	**Power Control**	β	**Link Failures**
Small	Night	No	12 dB	1.2%
Small	Night	Yes	12 dB	4.5%
Small	Night	Yes+Readj.	12 dB	0.2%
Large	Night	No	12 dB	0.5%
Large	Night	Yes+Readj.	12 dB	0.2%
Large	Day	No	12 dB	2.8%
Large	Day	Yes+Readj.	12 dB	3.1%

Table 5.2: Failure rate of links in live experiments using generated schedules under different conditions: 'Power Control: No' refers to RAND, 'Power Control: Yes' refers to PowerRAND.

fer a noticeable amount of failed links, PowerRAND more so than RAND. These are the result of the computed schedules not being able to account for link quality changes induced by the daytime environment between the last power test before schedule generation and the testing of the schedule. In cases where multiple links are affected even the generous β of 12 dB cannot prevent the prediction for the slot to turn out wrong. For all practical purposes we find the failure rates we obtained to be acceptable.

A problem related to the scheduling problem is the so-called *one-shot* problem: given a set of links L, we want to find the largest possible subset $L' \subset L$ such that all links in L' can be scheduled concurrently. In other words, we are looking for the largest feasible slot. Due to the greedy "stuffing" nature of RAND-like algorithms, these algorithms tend to also find good or at least decent solutions for the one-shot problem. Figure 5.12 shows the maximum slot size in each of the schedules in each of our 200 schedule sets. While RAND finds slots of size 4 in 90% of cases in the smaller node set, PowerRAND finds slots of size 5 in 30% of cases. Note that 5 is the theoretical maximum for this node set as it contains only 11 nodes – and every link in a slot requires a unique sender and a unique receiver. In less than 10% of cases the algorithms produce an exceptionally large slot for the larger node set: RAND finds a slot containing 8 links, while PowerRAND reaches 10 links.

5.8 Summary and Future Work

In this chapter, we demonstrated that the often overlooked feature of power control can be a valuable tool even for existing wireless algorithms. Using

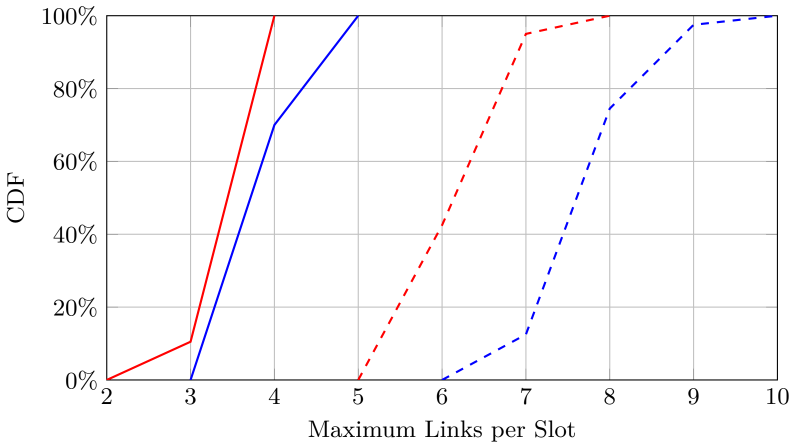


Figure 5.12: The distribution of the largest slot in the schedule.

the example of the simple yet powerful RAND scheduling algorithm, we show that by augmenting the algorithm to fine-tune transmission powers, we can improve the produced schedules as measured by various metrics. Our main result is the reduction in the length of the produced schedules by 20–25% as shown in Figure 5.6. Additionally, we propose other improvements possible with power control, such as optimization of SINR values within a slot and the on-the-fly adjustment to smaller changes in the environment. We verified that the shorter schedules our proposed PowerRAND algorithm produces are also feasible in practice.

One goal for future work is the integration of PowerRAND into a fully-fledged MAC layer implementation. The main issue to be addressed here is the overhead we incurred in our setup in the name of measurement accuracy. We perform power test phases every 2 minutes, which reserve slots for each node to transmit a packet alone, such that all other nodes are able to record the results without interference. While in practice lowering the test frequency to only once every 10 minutes, or testing only before computing a new schedule, is likely to suffice, such test phases still impose a significant overhead to regular operations.

The most ambitious solution we are envisioning is to adapt the link model not based on dedicated power tests, but purely on feedback from regular network operation: e.g., which links in which slots failed and succeeded, what accumulated power levels every node measures during every slot – even if the node does not participate in any transmissions in that slot. Further, changes to one link often also affect many others – perhaps all other links sharing a certain node or passing through the same hallway. Examples for such behavior are easy to find. One instance can be seen between power tests #11 and #12 in Figure 5.3. Harnessing the implicitly available information about such changes may be the key to avoiding regular sweeping power tests.

The other issue to address is the centralized nature of our setup. For a MAC layer to be practical, all its components should be distributed amongst the nodes of the network itself. As described earlier, it is not possible for PowerRAND to follow the same route from RAND to DRAND to achieve distributedness, mainly due to DRAND's simplified model of the network. However, we are confident that PowerRAND can be transferred to distributed operation by making some reasonable tradeoffs.

It may also be worthwhile to explore different avenues for dealing with unreliable links and slots. In this chapter, we focused on delivering a proof of concept for the possible benefits of power control, which is made clearer by avoiding, for instance, links that only successfully transmit in half of all attempts on average. If we are able to determine the rough probabilities for a successful transmission of a link, we may be able to simply schedule it multiple times to reach a desired average throughput. Alternatively, we could catch up on failed links every couple of executions of the schedule. Even more ambitiously, the schedule could constantly be adapted as links and traffic demands change. We already implemented a basic version of this with our slot readjustment technique responding to links underperforming.

6

Conclusion

In spite of the rapid spread of wireless technology, the space of both feasible wireless transmission primitives as well as algorithms making use of those primitives has yet to be fully explored. In this work, we made a contribution to this space by filling in a few of the many blanks. We investigated multiple applications that suffer when treated under traditional wired network based abstractions, and designed and evaluated solutions tailored to the wireless medium.

We showed that the combination of transmission power control and the capture effect may allow for traffic prioritization completely without overhead. Critical to the success of this method are the homogeneity of the network, or, more specifically, how small the biggest difference in "length" between any two links at a single node is, and the flexibility of the wireless nodes in terms of transmission power values available. Using TelosB motes on the FlockLab testbed we found the network to generally support at least 2 priority layers at every node. Our distributed fire alarm implementation indeed proved apt at quickly reporting high-priority alarms while causing little to no overhead to the low-priority protocol.

We showed the feasibility of synchronizing run-of-the-mill wireless sensor nodes' clocks well enough to achieve constructive interference. Unlike previous work, our solution does not require each sender to receive a common reference packet. Instead, it focuses on minimizing the error sources of clock synchronization and transmission timing.

Breaking with the abstraction that every packet is sent and received as a single packet, we considered possible applications for mixed sender schemes and explored the practical difficulties that needed to be overcome. Using signals of different strengths or by prematurely turning off a transmitter, receivers can be fooled into decoding a packet with mismatching synchronization headers and payloads. If we don't get lucky and the symbol string phases are not approximately the same, we are sometimes still able to puzzle the payload back together by exploiting certain relationships between the symbols' chip sequences.

Finally, we examined how power control can be applied to the RAND scheduling algorithm. We developed a power control aware version called PowerRAND, which attempts to use each link at the minimum possible operating power to minimize interference, and then in a second step optimizes the link powers within each slot. PowerRAND achieves schedules 20–25% shorter than RAND, and our proof of concept implementation showed that they are generally just as reliable in practice. Additionally, we investigated how power control allows schedules to react to changes in the environment.

While our protocol layering solution is already directly applicable to practice, the other results mostly remained as proofs of concept. Integrating these techniques into general purpose frameworks is left to future work. For many of these approaches, it would also be interesting to apply them to more powerful hardware: they are all limited in one way or another either by the available clock precision or by the granularity of the set of programmable transmission powers. Perhaps future hardware may also cater better to these needs.

Ultimately, the grand goal of our work is to inspire other applications to make more use of the wireless medium's unique properties. As virtually all sensor network MAC layers ignore power control, we envision that any MAC layer protocol X may be enriched to become PowerX, improving the protocol's characteristics such as medium utilization and throughput. Our treatment of RAND did not rely on any properties specific to RAND, yet PowerRAND certainly improved significantly over RAND. We do not see any reason why such an effect should not be repeatable across more wireless algorithms.

Bibliography

[1] Ash, D.L.: A comparison between OOK/ASK and FSK modulation techniques for radio links. Technical report, Technical report, RF Monolithics Inc (1992)

[2] Brar, G.S., Blough, D.M., Santi, P.: Computationally efficient scheduling with the physical interference model for throughput improvement in wireless mesh networks. In: Proceedings of the 12th Annual International Conference on Mobile Computing and Networking (MobiCom). (2006) 2–13

[3] Burri, N., von Rickenbach, P., Wattenhofer, R.: Dozer: ultra-low power data gathering in sensor networks. In: Proceedings of the 6th International Conference on Information Processing in Sensor Networks (IPSN). (2007) 450–459

[4] Chafekar, D., Kumar, V.S.A., Marathe, M.V., Parthasarathy, S., Srinivasan, A.: Cross-layer latency minimization in wireless networks with SINR constraints. In: Proceedings of the 8th ACM Interational Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc). (2007) 110–119

[5] Chafekar, D., Kumar, V.S.A., Marathe, M.V., Parthasarathy, S., Srinivasan, A.: Approximation algorithms for computing capacity of wireless networks with SINR constraints. In: 27th IEEE International Conference on Computer Communications (INFOCOM). (2008) 1166–1174

[6] Chang, H., Misra, V., Rubenstein, D.: A general model and analysis of physical layer capture in 802.11 networks. In: 25th IEEE International Conference on Computer Communications (INFOCOM). (2006)

[7] Chen, Y., Terzis, A.: On the mechanisms and effects of calibrating RSSI measurements for 802.15.4 radios. In: Wireless Sensor Networks, 7th European Conference (EWSN). Volume 5970 of Lecture Notes in Computer Science. (2010) 256–271

[8] Cidon, A., Nagaraj, K., Katti, S., Viswanath, P.: Flashback: decoupled lightweight wireless control. In: ACM SIGCOMM Computer Communication Review. Volume 42. (2012) 223–234

[9] Davis, D.H., Gronemeyer, S.: Performance of slotted ALOHA random access with delay capture and randomized time of arrival. IEEE Transactions on Communications **28**(5) (1980) 703–710

[10] Dinitz, M.: Distributed algorithms for approximating wireless network capacity. In: 29th IEEE International Conference on Computer Communications (INFOCOM). (2010) 1397–1405

[11] Doddavenkatappa, M., Chan, M.C.: P^3: a practical packet pipeline using synchronous transmissions for wireless sensor networks. In: Proceedings of the 13th International Symposium on Information Processing in Sensor Networks (IPSN). (2014) 203–214

[12] Dunkels, A., Grönvall, B., Voigt, T.: Contiki - A lightweight and flexible operating system for tiny networked sensors. In: 29th Annual IEEE Conference on Local Computer Networks (LCN). (2004) 455–462

[13] Dutta, P., Musaloiu-Elefteri, R., Stoica, I., Terzis, A.: Wireless ACK collisions not considered harmful. In: 7th ACM Workshop on Hot Topics in Networks (HotNets). (2008) 19–24

[14] ElBatt, T.A., Ephremides, A.: Joint scheduling and power control for wireless ad hoc networks. IEEE Transactions on Wireless Communications **3**(1) (2004) 74–85

[15] Elson, J., Girod, L., Estrin, D.: Fine-grained network time synchronization using reference broadcasts. In: 5th Symposium on Operating System Design and Implementation (OSDI). (2002)

[16] Elson, J., Römer, K.: Wireless sensor networks: a new regime for time synchronization. Computer Communication Review **33**(1) (2003) 149–154

[17] Fanghänel, A., Kesselheim, T., Räcke, H., Vöcking, B.: Oblivious interference scheduling. In: Proceedings of the 28th Annual ACM Symposium on Principles of Distributed Computing (PODC). (2009) 220–229

[18] Ferrari, F., Zimmerling, M., Mottola, L., Thiele, L.: Low-power wireless bus. In: Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems (SenSys). (2012) 1–14

[19] Ferrari, F., Zimmerling, M., Thiele, L., Saukh, O.: Efficient network flooding and time synchronization with Glossy. In: Proceedings of the 10th International Conference on Information Processing in Sensor Networks (IPSN). (2011) 73–84

[20] Flury, R., Wattenhofer, R.: Slotted programming for sensor networks. In: Proceedings of the 9th International Conference on Information Processing in Sensor Networks (IPSN). (2010) 24–34

[21] Ganeriwal, S., Kumar, R., Srivastava, M.B.: Timing-sync protocol for sensor networks. In: Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys). (2003) 138–149

[22] Ghosh, A., Ratasuk, R., Xiao, W., Classon, B.K., Nangia, V., Love, R., Schwent, D., Wilson, D.: Uplink control channel design for 3GPP LTE. In: Proceedings of the IEEE 18th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC). (2007) 1–5

[23] Gotzhein, R., Kuhn, T.: Black burst synchronization (BBS) - A protocol for deterministic tick and time synchronization in wireless networks. Computer Networks **55**(13) (2011) 3015–3031

[24] Goussevskaia, O., Oswald, Y.A., Wattenhofer, R.: Complexity in geometric SINR. In: Proceedings of the 8th ACM Interational Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc). (2007) 100–109

[25] Grönkvist, J., Hansson, A.: Comparison between graph-based and interference-based STDMA scheduling. In: Proceedings of the 2nd ACM Interational Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc). (2001) 255–258

[26] Gupta, P., Kumar, P.R.: The capacity of wireless networks. IEEE Transactions on Information Theory **46**(2) (2000) 388–404

[27] Halldórsson, M.M.: Wireless scheduling with power control. ACM Transactions on Algorithms **9**(1) (2012) 7:1–7:20

[28] Halldórsson, M.M., Tonoyan, T.: The price of local power control in wireless scheduling. In: 35th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS). Volume 45. (2015) 529–542

[29] Hänel, T., Krampe, F., Gericke, C., Aschenbruck, N.: On the potential of data-based time synchronization in wireless sensor networks for condition monitoring. In: International Conference on Distributed Computing in Sensor Systems (DCOSS). (2016) 216–224

[30] Hasler, A., Gruber, S., Beutel, J.: Kinematics of steep bedrock permafrost. Journal of Geophysical Research: Earth Surface **117**(F1) (2012)

[31] Huang, P., Desai, M., Qiu, X., Krishnamachari, B.: On the multihop performance of synchronization mechanisms in high propagation delay networks. IEEE Transactions on Computers **58**(5) (2009) 577–590

[32] Institute of Electrical and Electronics Engineers: IEEE Standard 802.15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks

[33] Jayasuriya, A., Perreau, S., Dadej, A., Gordon, S.: Hidden vs exposed terminal problem in ad hoc networks. PhD thesis, ATNAC (2004)

[34] Johansson, N., Körner, U., Johansson, P.: Performance evaluation of scheduling algorithms for Bluetooth. In: Broadband Communications: Convergence of Network Technologies, IFIP TC6 WG6.2 Fifth International Conference on Broadband Communications (BC). Volume 159. (1999) 139–150

[35] Kesselheim, T.: A constant-factor approximation for wireless capacity maximization with power control in the SINR model. In: Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). (2011) 1549–1559

[36] Kochut, A., Vasan, A., Shankar, A.U., Agrawala, A.K.: Sniffing out the correct physical layer capture model in 802.11b. In: 12th IEEE International Conference on Network Protocols (ICNP). (2004) 252–261

[37] Kusy, B., Dutta, P., Levis, P., Maróti, M., Lédeczi, Á., Culler, D.E.: Elapsed time on arrival: a simple and versatile primitive for canonical time synchronisation services. International Journal of Ad Hoc and Ubiquitous Computing (IJAHUC) **1**(4) (2006) 239–251

[38] Landsiedel, O., Ferrari, F., Zimmerling, M.: Chaos: versatile and efficient all-to-all data sharing and in-network processing at scale. In: Proceedings of the 11th ACM Conference on Embedded Network Sensor Systems (SenSys). (2013) 1:1–1:14

[39] Lee, J., Kim, W., Lee, S., Jo, D., Ryu, J., Kwon, T., Choi, Y.: An experimental study on the capture effect in 802.11a networks. In: Proceedings of the Second ACM Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization (WINTECH). (2007) 19–26

[40] Leentvaar, K., Flint, J.H.: The capture effect in FM receivers. IEEE Transactions on Communications **24**(5) (1976) 531–539

[41] Lenzen, C., Sommer, P., Wattenhofer, R.: PulseSync: An efficient and scalable clock synchronization protocol. IEEE/ACM Transactions on Networking (TON) **23**(3) (2015) 717–727

[42] Li, B., Nie, L., Wu, C., Gonzalez, H., Lu, C.: Incorporating emergency alarms in reliable wireless process control. In: Proceedings of the ACM/IEEE Sixth International Conference on Cyber-Physical Systems (ICCPS). (2015) 218–227

[43] Li, H., Feng, X., Shi, S., Zheng, F., Xie, X.: A high-accuracy clock synchronization method in distributed real-time system. In: Computer Engineering and Technology, 18th CCF Conference, NCCET 2014. Volume 491 of Communications in Computer and Information Science. (2015) 148–157

[44] Li, X., Zeng, Q.A.: Performance analysis of the IEEE 802.11 MAC protocol over a WLAN with capture effect. Information and Media Technologies (1) (2006) 679–685

[45] Liang, C.M., Chen, K., Priyantha, N.B., Liu, J., Zhao, F.: RushNet: practical traffic prioritization for saturated wireless sensor networks. In: Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems (SenSys). (2014) 105–118

[46] Lim, R., Ferrari, F., Zimmerling, M., Walser, C., Sommer, P., Beutel, J.: FlockLab: a testbed for distributed, synchronized tracing and profiling of wireless embedded systems. In: Proceedings of the 12th International Conference on Information Processing in Sensor Networks (IPSN). (2013) 153–166

[47] Lin, S., Zhang, J., Zhou, G., Gu, L., Stankovic, J.A., He, T.: ATPC: adaptive transmission power control for wireless sensor networks. In: Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (SenSys). (2006) 223–236

[48] Litjens, R., Roijers, F., Van den Berg, J., Boucherie, R.J., Fleuren, M.: Performance analysis of wireless LANs: an integrated packet/flow level approach. Teletraffic Science and Engineering **5** (2003) 931–940

[49] Lu, J., Whitehouse, K.: Flash flooding: Exploiting the capture effect for rapid flooding in wireless sensor networks. In: INFOCOM 2009. 28th IEEE International Conference on Computer Communications. (2009) 2491–2499

[50] Lynch, J.P., Loh, K.J.: A summary review of wireless sensors and sensor networks for structural health monitoring. Shock and Vibration Digest **38**(2) (2006) 91–130

[51] Maheshwari, R., Jain, S., Das, S.R.: A measurement study of interference modeling and scheduling in low-power wireless networks. In: Proceedings of the 6th International Conference on Embedded Networked Sensor Systems (SenSys). (2008) 141–154

[52] Manweiler, J., Santhapuri, N., Sen, S., Choudhury, R.R., Nelakuditi, S., Munagala, K.: Order matters: Transmission reordering in wireless networks. IEEE/ACM Transactions on Networking **20**(2) (2012) 353–366

[53] Maróti, M., Kusy, B., Simon, G., Lédeczi, Á.: The flooding time synchronization protocol. In: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys). (2004) 39–49

[54] Maróti, M., Kusy, B., Simon, G., Lédeczi, Á.: Robust multi-hop time synchronization in sensor networks. In: Proceedings of the International Conference on Wireless Networks (ICWN). (2004) 454–460

[55] Mills, D.L.: Internet time synchronization: the network time protocol. IEEE Transactions on Communications **39**(10) (1991) 1482–1493

[56] Moscibroda, T., Wattenhofer, R.: The complexity of connectivity in wireless networks. In: 25th IEEE International Conference on Computer Communications (INFOCOM). (2006)

[57] Moscibroda, T., Wattenhofer, R., Weber, Y.: Protocol design beyond graph-based models. In: 5th ACM Workshop on Hot Topics in Networks (HotNets). (2006) 25–30

[58] Moscibroda, T., Wattenhofer, R., Zollinger, A.: Topology control meets SINR: the scheduling complexity of arbitrary topologies. In: Proceedings of the 7th ACM Interational Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc). (2006) 310–321

[59] Newzoo: Newzoo's global mobile market report, April 2017. Technical report, Newzoo (2017)

[60] Nyandoro, A., Libman, L., Hassan, M.: Service differentiation using the capture effect in 802.11 wireless LANs. IEEE Transactions on Wireless Communications **6**(8) (2007) 2961–2971

[61] Patras, P., Qi, H., Malone, D.: Mitigating collisions through power-hopping to improve 802.11 performance. Pervasive and Mobile Computing **11** (2014) 41–55

[62] Polastre, J., Szewczyk, R., Culler, D.E.: Telos: enabling ultra-low power wireless research. In: Proceedings of the Fourth International Symposium on Information Processing in Sensor Networks (IPSN). (2005) 364–369

[63] Ramanathan, S.: A unified framework and algorithm for channel assignment in wireless networks. Wireless Networks **5**(2) (1999) 81–94

[64] Ray, S., Carruthers, J.B., Starobinski, D.: RTS/CTS-induced congestion in ad hoc wireless LANs. In: 2003 IEEE Wireless Communications and Networking (WCNC). Volume 3. (2003) 1516–1521

[65] Rhee, I., Warrier, A., Aia, M., Min, J., Sichitiu, M.L.: Z-MAC: a hybrid MAC for wireless sensor networks. IEEE/ACM Transactions on Networking (TON) **16**(3) (2008) 511–524

[66] Rhee, I., Warrier, A., Min, J., Xu, L.: DRAND: distributed randomized TDMA scheduling for wireless ad-hoc networks. In: Proceedings of the 7th ACM Interational Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc). (2006) 190–201

[67] Römer, K.: Time synchronization in ad hoc networks. In: Proceedings of the 2nd ACM Interational Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc). (2001) 173–182

[68] Sallai, J., Kusy, B., Lédeczi, Á., Dutta, P.: On the scalability of routing integrated time synchronization. In: Wireless Sensor Networks, Third European Workshop (EWSN). Volume 3868 of Lecture Notes in Computer Science. (2006) 115–131

[69] Santhapuri, N.K., Manweiler, J., Sen, S., Choudhury, R.R., Nelakuditi, S., Munagala, K.: Message in message (MIM): A case for shuffling transmissions in wireless networks. In: 7th ACM Workshop on Hot Topics in Networks (HotNets). (2008) 25–30

[70] Schmid, T., Dutta, P., Srivastava, M.B.: High-resolution, low-power time synchronization an oxymoron no more. In: Proceedings of the 9th International Conference on Information Processing in Sensor Networks (IPSN). (2010) 151–161

[71] Selavo, L., Wood, A.D., Cao, Q., Sookoor, T.I., Liu, H., Srinivasan, A., Wu, Y., Kang, W., Stankovic, J.A., Young, D., Porter, J.: LUSTER: wireless sensor network for environmental research. In: Proceedings of the 5th International Conference on Embedded Networked Sensor Systems (SenSys). (2007) 103–116

[72] Seth, A., Kroeker, D., Zaharia, M.A., Guo, S., Keshav, S.: Low-cost communication for rural internet kiosks using mechanical backhaul. In: Proceedings of the 12th Annual International Conference on Mobile Computing and Networking (MOBICOM). (2006) 334–345

[73] Son, D., Krishnamachari, B., Heidemann, J.S.: Experimental study of concurrent transmission in wireless sensor networks. In: Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (SenSys). (2006) 237–250

[74] Su, L., Liu, C., Song, H., Cao, G.: Routing in intermittently connected sensor networks. In: Proceedings of the 16th annual IEEE International Conference on Network Protocols (ICNP). (2008) 278–287

[75] Texas Instruments: 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver. CC2420 Data Sheet.

[76] Whitehouse, K., Woo, A., Jiang, F., Polastre, J., Culler, D.E.: Exploiting the capture effect for collision detection and recovery. In: Proceedings of the 2nd IEEE Workshop on Embedded Networked Sensors (EmNets). (2005) 45–52

[77] Xu, K., Gerla, M., Bae, S.: How effective is the IEEE 802.11 RTS/CTS handshake in ad hoc networks. In: Proceedings of the Global Telecommunications Conference (GLOBECOM). (2002) 72–76

[78] Yuan, D., Hollick, M.: Let's talk together: Understanding concurrent transmission in wireless sensor networks. In: 38th Annual IEEE Conference on Local Computer Networks (LCN). (2013) 219–227

[79] Yuan, D., Riecker, M., Hollick, M.: Making ‘Glossy’ networks sparkle: Exploiting concurrent transmissions for energy efficient, reliable, ultra-low latency communication in wireless control networks. In: Wireless Sensor Networks, 11th European Conference (EWSN). Volume 8354 of Lecture Notes in Computer Science. (2014) 133–149

[80] Zimmerling, M., Ferrari, F., Mottola, L., Thiele, L.: On modeling low-power wireless protocols based on synchronous packet transmissions. In: IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS). (2013) 546–555

[81] Zuniga, M., Krishnamachari, B.: Analyzing the transitional region in low power wireless links. In: Proceedings of the First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON). (2004) 517–526

Publications

During my PhD studies at ETH Zurich, I co-authored the following publications. The authors are listed in alphabetical order.

- On Local Fixing. Michael König and Roger Wattenhofer. *17th International Conference On Principles Of Distributed Systems (OPODIS), December 2013.*

- Sharing a Medium Between Concurrent Protocols Without Overhead Using the Capture Effect. Michael König and Roger Wattenhofer. *13th International Conference on Embedded Wireless Systems and Networks (EWSN), February 2016.*

- Maintaining Constructive Interference Using Well-Synchronized Sensor Nodes. Michael König and Roger Wattenhofer. *12th Annual International Conference on Distributed Computing in Sensor Systems (DCOSS), May 2016.*

- A Concept for an Introduction to Parallelization in Java: Multithreading with Programmable Robots in Minecraft. Klaus-Tycho Förster, Michael König and Roger Wattenhofer. *17th Annual Conference on Information Technology Education (SIGITE), September 2016.*

- Effectively Capturing Attention Using the Capture Effect. Michael König and Roger Wattenhofer. *14th ACM Conference on Embedded Networked Sensor Systems (SenSys), November 2016.*

- Multi-Agent Pathfinding with n Agents on Graphs with n Vertices: Combinatorial Classification and Tight Algorithmic Bounds. Klaus-Tycho Förster, Linus Groner, Torsten Hoefler, Michael König, Sascha

Schmid and Roger Wattenhofer. *10th International Conference on Algorithms and Complexity (CIAC), May 2017.*

- Tempering Wireless Schedules. Michael König and Roger Wattenhofer. *To appear.*