COM SYS Communication & Distributed Systems

RWTH AACHEN UNIVERSITY

Temperature Dependency of Bit Error Patterns in Wireless Sensor Networks

Bachelor Thesis

Niklas Hauser

RWTH Aachen University, Germany
Chair of Communication and Distributed Systems

Advisors:

Dr.	Matteo Ceriotti
Dipl.-Inform.	Florian Schmidt
Prof. Dr.-Ing.	Klaus Wehrle
Prof. Dr. rer. nat.	Bernhard Rumpe

Registration date: January 29, 2014
Submission date: May 29, 2014

I hereby affirm that I composed this work independently and used no other than the specified sources and tools and that I marked all quotes as such.

Hiermit versichere ich, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Aachen, den 29. Mai 2014

Kurzfassung

Mit der Verfügbarkeit von leistungsfähigen eingebetteten Geräten und kostengünstigen Funkmodulen, beginnen auch immer mehr batteriebetriebene Sensorgeräte miteinander zu reden und Probleme gemeinsam zu entscheiden. Es ist abzusehen, dass die Benutzung solcher intelligenten Geräte in drahtlosen Sensornetzwerken industrielle Prozesse vereinfachen und unsere Lebensqualität verbessern wird.

Allerdings ist die Laufzeit dieser mobilen Geräte durch ihre endliche Energiequelle, die meist nur mit viel Aufwand zu wechseln ist, beschränkt. Daher braucht es eine Programmierung, die energiebewusst kommuniziert, um die Lebensdauer der Batterie so weit wie möglich zu verlängern. Drahtlose Kommunikation ist stark durch Umweltveränderungen, allen voran die Lufttemperatur, beinflussbar, was zu Korruption oder Verlust von Paketen führen kann und ein erneutes Versenden des Pakets nach sich zieht.

In dieser Arbeit beschreiben wir eine kostengünstige Testplatform zum Testen der Funkverbindungen in einer temperaturkontrollierten Umgebung. Mit dieser Platform untersuchen wir die Empfangsraten und die Bitfehlerverteilungen in korrupten Paketen. Unsere Ergebnisse widerlegen frühere veröffentlichte Ergebnisse. Schließlich wenden wir diese Erkenntnisse auf die Programmierung eines Simulators an, mit dem wir untersuchen, wie sich Temperatur als Eingabegröße einer adaptiven Vorwärtsfehlerkorrektur verhält, um damit die Empfangsraten korrupter Pakete zu verringern.

Abstract

With the availability of powerful embedded devices and inexpensive radio communication, ever more battery powered sensor devices make use of the added connectivity to talk to each other and decide problems collectively. It is envisioned that usage of such smart devices in Wireless Sensor Networks will streamline industrial processes as well as improve our quality of life.

However, the runtime of these mobile devices is limited by their finite power source, which most often are impractical to exchange. Therefore energy-aware programming and communication is required to prolong battery life for as long as possible. Unfortunately, wireless communication is strongly influenced by environmental changes, most prominently air temperature, which can lead to packet corruption or loss and requires energy-consuming retransmissions.

In this work we describe a low-cost testbed for testing radio performance in a temperature-controlled environment, with which we examine packet reception rates and bit error distributions within packet corruption. Our results disprove previous published findings. Finally, we apply these findings on building a simulator, with which we investigate using temperature as an input for an adaptive Forward Error Correction scheme to improve packet reception rates and thereby reduce unnecessary retransmissions.

Acknowledgments

I want to thank my advisors Matteo and Florian for their time and support. Our discussions kept me focused on what was important and constantly pushed me towards improving the experiments and evaluations.

Many thanks go to my supervisor Prof. Dr.-Ing. Klaus Wehrle and to my second supervisor Prof. Dr. Rumpe.

Contents

1 Introduction **1**

2 Background **3**

2.1 Wireless Sensor Networks 3

2.2 Communications 4

2.2.1 IEEE 802.15.4 Standard 4

2.2.2 UART Communication 5

2.3 Error Control Schemes 6

3 Related Work **7**

3.1 Effects of Temperature 7

3.2 Bit Error Distributions 8

3.3 Forward Error Correction 9

4 Testbed Description **11**

4.1 Temperature Box 11

4.1.1 Hardware 11

4.1.2 Embedded Software 12

4.1.3 Performance 13

4.2 Mote Harness 13

4.3 Control Software 14

5 Experiment Results **17**

5.1 Locations 17

5.2 Desired Link Quality 18

5.3 Microcontroller Clock Drift 18

5.4 Patterns in Bit Error Distributions 20

5.4.1 Effects of Board Layout 20
5.4.2 Effects of Temperature 22
5.4.3 Pattern Anomalies 23
5.5 Packet Reception Rate 24
5.5.1 Effects of Temperature on PRR and BER 25
5.5.2 Effects of Temperature on LQI and RSSI 25
5.5.3 Discussion 26

6 Forward Error Correction **29**
6.1 Choice of FEC Scheme 29
6.2 RS Scheme Simulation 30
6.2.1 Design 31
6.2.2 Accuracy 32
6.3 Comparing RS Scheme Strengths 34
6.4 Discussion 36

7 Conclusion **39**

Bibliography **41**

A Appendix **45**
A.1 List of Abbreviations 45
A.2 Experiment Data 46

1

Introduction

With the introduction of more powerful low-power embedded devices, such as small microcontrollers, and the increased availability of low-cost radio transceivers especially in the last decade, more and more applications have been developed that enrich their functionality by receiving input from our environments. Such applications can help us to use natural resources, such as farm land or forests, more efficiently, or streamline industrial processes by retrofitting more data sampling points, or simply alarm us when we forget to water the plants at home.

Concepts of augmenting our environments with connected sensors and actuators have been talked about for many years, with the most prominent concept being the Internet of Things (IoT). Here everyday objects are able to communicate with each other over local networks or the Internet and come to smart decisions collectively. This infusion of information about the real into the virtual world will keep growing with many new and exciting ways to improve our quality of life.

However, these concepts are built on very new technology, which has to be understood to be improved for the next generation of such devices. The common culprit of today's mobile devices are their battery lives. Even though impressive advancements have been made, especially in the area of smart phones, charging or exchanging batteries is cumbersome at best, and, in regard to sensor devices, impractical when they are distributed over large areas. It is therefore imperative to extend battery life for as long as possible, which means not only using low-power hardware, but also merging it with energy-efficient software.

Fortunately, long and intensive computations do not have to be performed on the sensor devices locally, but can be delegated to a server connected to the power grid. For that however, the sensor nodes must communicate using radio communication, which is by the laws of physics and electrical engineering still inherently energy-inefficient, especially compared to wired communication.

What sounds simple actually is complicated by the fact that radio communication is at the mercy of the environment, which introduces noise into the signal and

strongly influences signal propagation. Further complications arise with different antenna designs, the limitations of the modulation scheme and non-linearities in analog circuitry. This leads to packets being corrupted or not being received at all, requiring the retransmissions of the original packet until it is received error-free. However, in conditions of poor link quality this might not even be possible. Hence, to reduce energy consumption, radio communication, especially retransmissions, must be kept to a minimum while not impairing functionality of the original application.

Additionally, a typical Wireless Sensor Network is deployed outside and its nodes are protected by air-tight containers from precipitation. On a sunny day, temperatures inside these containers can climb well above outside air temperature, which influence radio and microcontroller performance quite dramatically. In this thesis we focus on what impact temperature has on patterns in packet corruption and describe the experiments we created to study these patterns. We analyze the results and try to improve link quality using Forward Error Correction.

The main contributions of this thesis are:

1. the creation of a low-cost testbed for testing radio performance in a temperature-controlled environment (Chapter 4),
2. the investigation of microcontroller clock drift, and the limitations of its calibaration in TinyOS (Chapter 5),
3. the detailed analysis of the effects of temperature on bit error patterns and Packet Reception Rate, which disproves published findings by Boano et al. [6] (Chapter 5),
4. the modeling of the discovered patterns using a simulator (Chapter 6), and
5. the investigation of using an adaptive Forward Error Correction scheme to improve throughput in temperature-influenced link conditions (Chapter 6).

We provide the necessary background knowledge required for understanding our work in in Chapter 2, before briefly discussing previous, related work in Chapter 3.

2

Background

This work centers around investigating and improving the quality of corrupted communications in low power networks. Therefore this chapter will provide the necessary background information on the form of communication used in Wireless Sensor Networks and the radio and microcontroller hardware we employed in our study. Furthermore, we describe the theory of operation of Forward Error Correction and shortly describe what approaches to Error-Correction Codes exist and how they work.

2.1 Wireless Sensor Networks

A Wireless Sensor Network (WSN) comprises of two or more nodes, called *motes*, which can communicate using low power radio communication. This provides a means to easily deploy sensing, actuating and communication infrastructures in a given space. The nodes are mostly powered by small and simple microcontrollers connected to a radio and other peripherals such as sensors and actuators as shown in Figure 2.1.

Networked motes communicate with their neighboring motes and can address the whole network via multi-hop transmissions, therefore deprecating the use of expensive wired buses. However, due to the nature of wireless communication, the software must be built to tolerate unreliable communication links, which can change network topology and result in high latency in data exchanges or a complete loss of communication.

Integrated batteries are used to power the motes for a certain amount of time when deployed in areas where access to a power grid is too costly or impossible. Especially for these battery powered applications, conserving energy is a high priority and places tight restrictions on the type of CPU and radio transmission power and duty cycle, effectively limiting communication range and bandwidth.

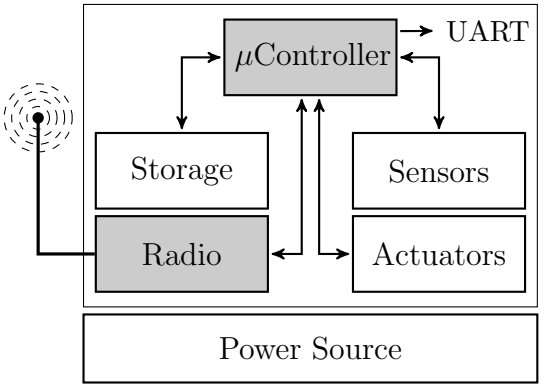


Figure 2.1 A schematic representation of a mobile WSN mote with a power source.

A typical application for a WSN would be the collection of meteorological information over large inaccessible areas, such as forests, or the monitoring of large industrial complexes to reduce the cost of wiring [4]. The high effort required to exchange empty batteries in these scenarios demands energy-aware programming, which keeps the overall energy consumption to a minimum.

2.2 Communications

Since this works focuses heavily on communications between motes, we use specific terms to help describe its properties.

A *packet* is the basic unit of communication between two motes, containing metadata, such as sender and addressee, and payload. A *message* is the interpretation of the payload of one or several packets. If a message is very long, it might be fragmented into several packets, however, in all our experiments, one message uses only one packet. We define a *link* as the transmission of messages from one mote addressed to another over a period of time, even when none of these messages where received by the addressee, due to channel effects or connection problems.

2.2.1 IEEE 802.15.4 Standard

In this work we use the CC2420 wireless transceiver [22], a very common communication radio in WSNs. The CC2420 implements the Institute of Electrical and Electronics Engineers (IEEE) 802.15.4 standard and uses Offset QPSK (OQPSK) modulation in the 2.4 GHz ISM band with a nominal data rate of 250 kbit/s.

The Physical Protocol Data Unit (PPDU) is the lowest accessible layer as shown in Figure 2.2 with a maximum size of 133 bytes. This leaves 127 bytes for the MAC Protocol Data Unit (MPDU) and between 102 and 122 bytes for the frame payload. To increase robustness of transmission, each byte is split into two 4-bit symbols, which are then mapped to a 32-bit chip sequence and modulated using OQPSK.

These 32-bit chip sequences are part of IEEE 802.15.4 and were chosen to produce high Hamming distances against each other, so that bit errors can already be corrected within the received chip sequence to some degree. The higher the Hamming

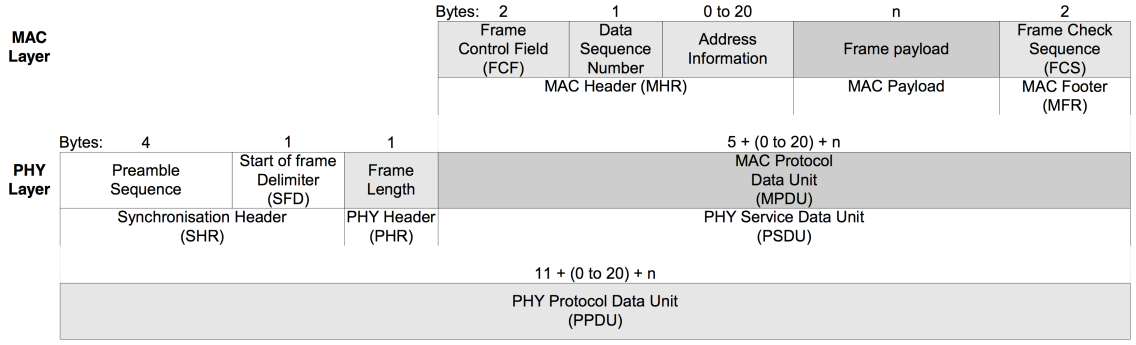


Figure 2.2 The packet format implemented defined by IEEE 802.15.4 taken from [22].

distance between two sequences, the less likely they are to be mistaken for one another. Therefore a corrupted 4-bit symbol is only received, when a corrupted chip sequence is mistaken for another sequence, which requires several bits of the 32-bit sequence to be flipped. The minimum Hamming distance between two sequences is 12 bit, the maximum is 20 bit [20].

At the beginning of every transmission, a Preamble Sequence is modulated, which triggers a listening mote to synchronize and receive this new packet. Typical reasons for not receiving packets are either the reception of a signal with too little power to be noticeable above the channel noise, or the reception of a corrupted preamble, so that the listening mote does not start receiving a packet, even though the power levels would permit it.

When a packet is received, its Frame Check Sequence (FCS) is automatically evaluated to check for bit errors introduced by channel noise. However, a failed FCS test can also be the results of a corrupted Frame Length field. This prompts the radio to read in too few or too many bytes, thereby comparing the FCS results to the wrong bytes at, what the receiver wrongly considers the “end” of the packet.

Even though the FCS is evaluated for all messages automatically, the CC2420 allows retrieval of corrupted messages, including all hardware link qualifiers such as Link Quality Indication (LQI) and Received Signal Strength Indication (RSSI). We make use of this ability to analyze the content of these corrupted messages in our work.

2.2.2 UART Communication

The microcontroller can also communicate over a wired bus using its Universal Asynchronous Receiver Transmitter (UART) module. UART does not provide a seperate clock line (hence *asynchronous*), which requires the receiver to synchronize itself at the beginning of each symbol transfer and sample the signal at multiples of the baudtime $t_{symbol} = f_{Clock}/f_{BaudRate}$, which derives from the *baudrate* of the transmitter. Here the receiver synchronizes on the falling edge of the start bit, which is always low, and has to sample the successive bits at the just right time, as shown in Figure 2.3. If the transmitter’s and/or receiver’s clock is running too slow or too fast, then t_{Symbol} is too small or too large, resulting in a corrupted sampling of the bits in the symbol as marked in red.

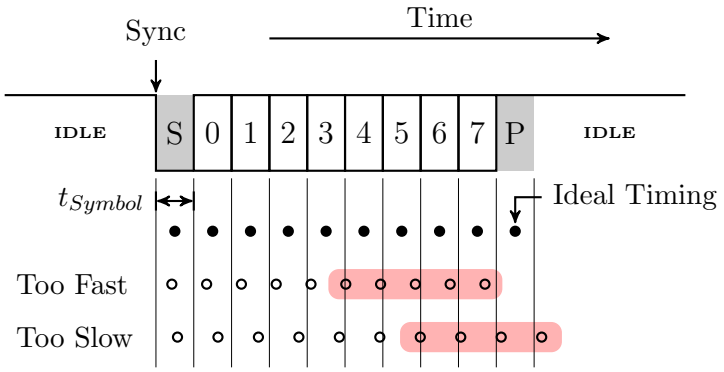


Figure 2.3 UART timing diagram of the sampling points of an 8-bit symbol.

The relative error tolerance for the 8N1 configuration shown in the Figure (8 databits, 1 startbit and 1 stopbit) is ±5%, since the sample point of the stop bit may only shift by at most $\pm t_{Symbol}/2$. With 10 bits to read, one t_{Symbol} equals one tenth of the symbol transmission time, hence a relative tolerance of ±5%. For example, the tolerance for 7-bit transfers (9 baudtimes) increases to ±5.56%.

However, since both transmitter and receiver may experience clock drift, the error must not exceed 5% in total, which in the worst case of opposing clock drift directions (one too fast, one too slow) imposes a tight allowed deviation of +2.5% and −2.5% on the modules respectively. In practice, UART can therefore be difficult to use with uncorrected or unknown clock drifts.

2.3 Error Control Schemes

There exist two approaches to improve the quality of an imperfect communication channel: using Forward Error Correction (FEC) to add redundancy before transmission, so that bit errors can be *detected* and *corrected* by the receiver, and using Automatic Repeat Query (ARQ) to only *detect* bit errors, but request a retransmission, instead of correcting them. While ARQ is simple and requires very little computational and transmission overhead, it does perform poorly in conditions with a high channel error rate. FEC allows constant throughput at the expense of coding and decoding time and transmission overhead, depending on the specific Error-Correction Code (ECC) used. To overcome their individual disadvantages, these two approaches can be merged into *hybrid* ARQ schemes, however, in our work we will be focusing on using FEC schemes alone.

The simplest ECCs belong to the class of linear cyclic coding schemes, where a bit stream is divided into non-overlapping blocks, which are then encoded separately. The most widely used linear cyclic block codes are binary Bose-Chaudhuri-Hocquenghem (BCH) and non-binary Reed-Solomon (RS) codes, both of which can be used to correct bit and burst errors. There are several other approaches to ECCs, however, they go above the focus of this work.

3

Related Work

The research field of investigating influences of the environment on wireless link quality is large and spans many technologies and protocols. However, we focus on technologies deployed in low power networks such as WSNs, where the advances in small and affordable radio technology pushed research forward. In a first step towards energy efficient, reliable communications, especially in battery powered applications, a deep understanding of why messages become corrupted is required.

Here we present research relating to the IEEE 802.15.4 protocol used in most WSNs, especially in the three areas this work focuses on, namely, the influence of temperature on link quality and bit error distributions within corrupted messages, as well as the use of FEC to counteract these effects.

3.1 Effects of Temperature

Bannister et al. [3] were one of the first to systematically investigate the effect of temperature on the performance of the CC2420 radio. They connected two radios together via coaxial cable with attenuators and placed one of them in a thermal chamber with a temperature range of 25 to 65 °C while measuring signal power using the RSSI. The results showed a reduction of output power by $4 - -5$ dBm when the transmitter was heated, but only 3 dBm reduction in measured input power when the receiver was heated. Boano et al. [5] also investigated this behavior in 2010 in a real world deployment, but found no difference in RSSI between transmitter and receiver. However, both works point to a loss of gain in the CC2420 Low Noise Amplifier (LNA) as the source of the reduction in signal powers at higher temperatures.

In a long-term outdoor study in 2013, Wennerström et al. [26] examined the correlation between RSSI and Packet Reception Rate (PRR) and environmental factors, such as temperature, relative and absolute humidity, precipitation and sunlight.

They found that of all environmental factors, temperature had the strongest influence, with an increase in temperature causing a significant decrease in both RSSI and PRR. This was further researched by Boano et al. [6], who focused on the impact of temperature on RSSI, Signal-to-Noise Ratio (SNR), Noise Floor, PRR and LQI. They determined a negative impact of temperature on all these values and expanded their work by looking at the asymmetry of PRR when heating the transmitter and receiver. They found that the loss in PRR was more pronounced when heating the transmitter than the receiver.

Finally, Zúñiga et al. [25] created a summarizing report on the effect of environment variables on current WSN motes. Apart from including all previous findings, they also describe a substantial negative impact of temperature on microcontroller clock drift. However, the CC2420 clock is compensated for temperature and experiences no such drift.

3.2 Bit Error Distributions

Liang et al. [14] investigated bit error distributions in the context of 802.11 (WiFi) interference. They found that packet collisions cause burst errors, due to 802.11 packets typically being much shorter than 802.15.4 packets, particularly in the beginning of a message. This is due to the 802.11 sender deferring sending its packets until the 802.15.4 transmission completed, however only if the 802.15.4 sender is close enough so that its transmission power can be sensed by the 802.11 collision avoidance algorithm. If the 802.11 sender cannot sense the 802.15.4 transmission, the authors show an even bit error distribution across the entire packet.

Schmidt et al. [20] examined the bit error distributions within corrupted messages in their outdoor testbed of 20 TelosB devices. Similar to Liang et al., they found that for random payload, Bit Error Rate (BER) remains stable throughout message, however, within all 4-bit symbols, the Most Significant Bit (MSB) is significantly less likely to break than the 3 Least Significant Bit (LSB). In addition, symbols with MSB set to 1 are more likely to break than symbols with MSB set to 0. This behavior creates very different Hamming distances between the broken symbols coded in 32-bit chip sequences than defined by the 802.15.4 standard. The authors also looked at burst error distribution and found these errors not independently distributed, but skewed towards longer bursts.

Their findings then where further completed by Hermans et al. [12] with an understanding of why the Hamming distances are different than expected. The authors discovered that the pattern fits, when their simulation of these patterns uses a Minimum Shift Keying (MSK) instead of the OQPSK demodulator suggested by the 802.15.4 standard. The MSK demodulator can correctly receive packets sent by an OQPSK modulator, but it outputs different code words than defined by the standard. The authors hypothesized that these code words then have to be translated using a table, which would create the distinct bit error patterns.

3.3 Forward Error Correction

The use of FEC schemes in communication has been investigated before, however, the available limited resources on WSN motes focus research in this area very much on coding and energy efficiency.

Jeong et al. [13] investigated the performance of single- and double-bit correcting ECC on very early WSN motes, not using the 802.15.4 standard. They came to the conclusion, that using BCH codes is simple to implement, but performs poorly when encountering burst errors. Busse et al. [8] then compared the performance of a Hamming code, an interleaved double-bit correction code and a RS code using their WSN and found the RS code to be the most efficient of them, both in the length of the added overhead, as well as the error correction capabilities.

Using mathematical energy consumption modeling, Tian et al. [23] showed that for small packet lengths (< 1023 bytes) BCH codes outperform ARQ. However, research by Ma et al. [16] points to RS being more energy efficient than BCH codes, especially when considering the energy consumption of transmitting the coding overhead.

Ahn et al. [1] examined changing FEC coding strength dynamically based on link quality properties. Their adaptive FEC algorithm outperformed several static RS codes in simulation, as well as in real traces. Liang et al. [14] created and evaluated an efficient, TinyOS compatible RS implementation, called TinyRS, during their research on bit error distributions mentioned before.

4

Testbed Description

In order to be able to investigate the effects of temperature on our WSN motes, we built a new low-cost testbed, which can accurately control mote temperature and allows the mote to maintain a certain antenna position. Furthermore, we wrote flexible control and evaluation software, which orchestrates the experiments and logs all relevant data to disk. In this chapter we will describe the design decisions we made and how they influence the testbed performance.

4.1 Temperature Box

For our experiments we needed a way to accurately control temperature of motes in our WSN. Even though previous setups used an infrared lamp [7, 11] to heat the motes directly, we chose to control mote temperature only using air temperature. The mote is placed in a closed Polystyrene (PS) hard-foam box and the confined air is heated to the requested temperature and circulated to transfer this heat to the mote. The box has the outside dimensions of 35 cm × 35 cm × 30 cm with a wall thickness of 5 cm which results in a holding capacity of 12.5 l.

4.1.1 Hardware

A microcontroller evaluates temperature sensors within the box and locally controls the duty cycles of a small fan and a 150 W ceramic heating element using a closed-loop PID controller to achieve the requested air temperature. The microcontroller can communicate over a serial connection to receive the desired temperature and send the current temperature.

Additionally, air temperature is displayed as a hue from blue (cold) via green (warm) to red (hot) on a RGB LED. The duty cycles of the heating element and fan are mapped to two white LEDs, to provide an immediate status overview and prevent burn injuries.

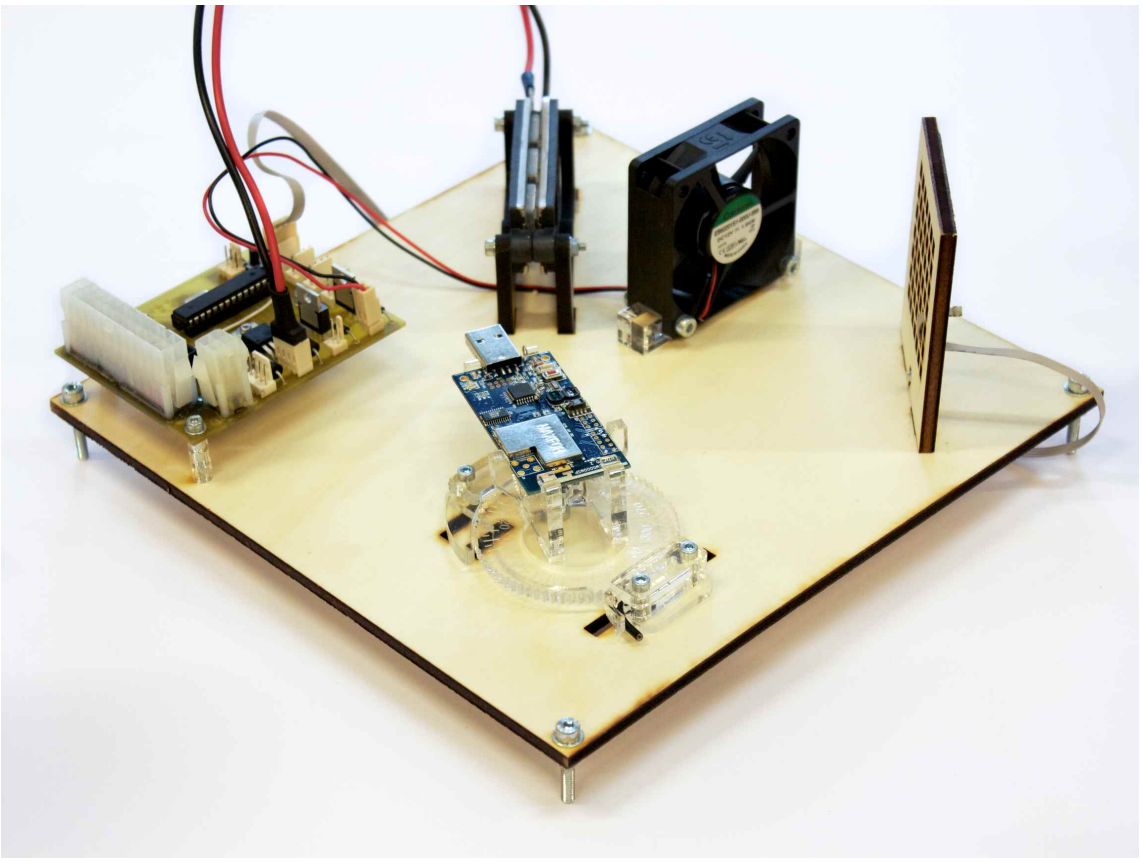


Figure 4.1 Baseplate with temperature controller, heating element, fan and sensor mote.

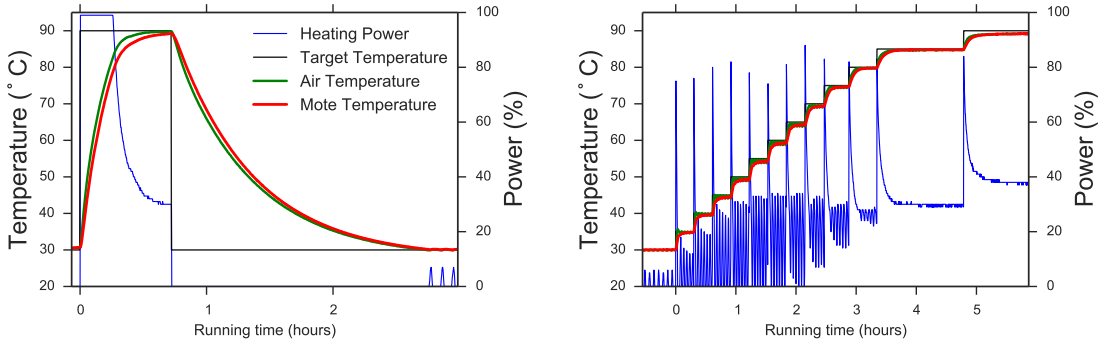
The box is powered by a 300 W PC power supply, which was chosen for its ability to provide 12 V, 5 V and 3.3 V at the required currents, which removes the need for additional (costly) voltage conversion. A custom designed PCB distributes power from the ATX connectors to two high and two medium power MOSFET switches and up to five temperature sensors, all controlled by an ATmega328.

The PCB, heating element and fan are fixed on a wooden baseplate shown in Figure 4.1 using custom fasteners and standard screws. All custom made parts were prototyped using the PCB mill and laser cutter of the RWTH FabLab [10].

4.1.2 Embedded Software

The embedded software is written in C++ using the xpcc microcontroller framework [27] and implemented as a set of asynchronous control tasks for input parsing and output formatting, temperature sensor evaluation, PID loop update and duty cycle generation. The choice of an ATmega328 as a microcontroller was deliberate to enable compatibility with the Arduino framework, which might be more familiar to developers.

Once the controller is programmed via In-System Programming (ISP), it will periodically send the values of all attached temperature sensors, as well as the current heating element power setting in human-readable ASCII format. Desired temperature can be sent as an ASCII formatted integer followed by the letter 'C'. All



(a) Heating to 90 °C, then cooling to 30 °C.

(b) Temperature increments of 5 °C.

Figure 4.2 Typical performance of the boxes over time. Notice the mote temperature (red line) lagging behind air temperature (green line).

switching frequencies were kept well below 1 kHz to avoid any kind of interference with the 2.4 GHz band.

4.1.3 Performance

The heating element has enough power to heat the air inside the box up to 120 °C. This can however damage both the PS material as well as the mote, so a hard limit of 90 °C is imposed during experiments.

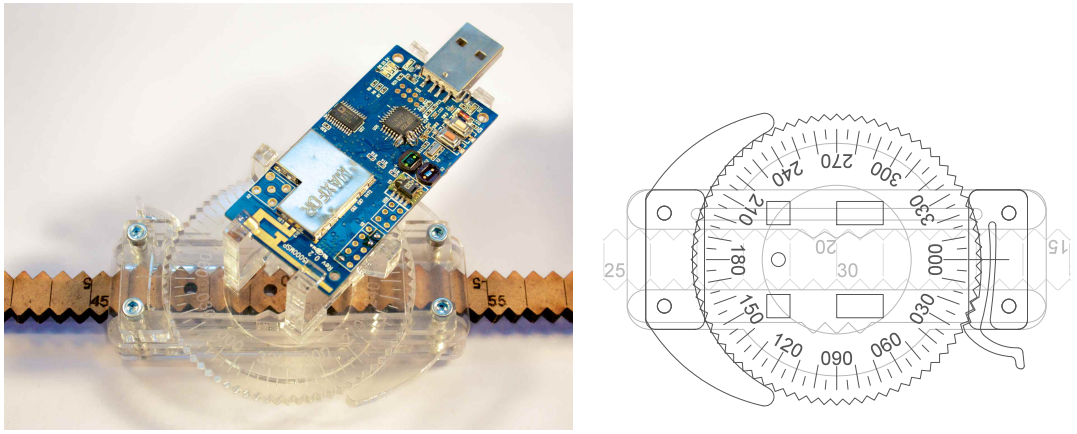
As seen in Figure 4.2(a) it takes about 40 minutes to heat up to 90 °C and about 2 hours to cool back down to 30 °C. The PID loop is deliberately dampened so that no overshoot in air temperature occurs, however, since the mote temperature naturally lags behind, a slight overshoot in air temperature might actually help achieve desired mote temperature quicker. However, during the experiments a more granular approach is used similar to Figure 4.2(b).

The boxes can be retrofitted with an active piezoelectric cooling element and a fan by connecting them to the two unused MOSFET switches on the controller and updating the software, which also allow reaching temperatures below room temperature. Since we did not use a cooling element, we set the minimum experiment temperature at 30 °C so that in a room temperature of 20−25 °C it can still be reached within reasonable time.

Compared to TempLab by Boano et al. [7], which uses infrared heat lamp controlled via the wireless Z-Wave home automation protocol, our boxes require more manual assembly, do not operate wirelessly and cannot change temperature as fast. However, our boxes also only cost one third as much (€ 91 versus € 293).

4.2 Mote Harness

In order to guarantee that both sender and receiver will not move during experiments a mote harness was developed and laser cut out of acrylic. The mote snaps into the harness so that the axis of rotation goes through the middle of the PCB antenna



(a) Harness with mote on a distance bar. (b) Vector drawing of the rotation disc.

Figure 4.3 The mote harness allows precise positioning of motes.

which then allows rotation in 5° steps as shown in Figure 4.3. Furthermore, the distance between two harnesses can be adjusted in 5 mm steps using laser-cut wooden distance bars.

In our experiments this greatly improved the handling of the motes. Where before we used to tape the mote down into position, which was an inaccurate matter at best, we now have a simple and elegant solution. However, we ended up not actually using the distance bars in most of our experiments, as it was simpler to use antenna orientation to control link quality than distance. Therefore, in Figure 4.1 the mote harness is fixed onto the baseplate.

4.3 Control Software

The motes and temperature boxes are connected via USB-to-Serial converters to a computer, and controlled with a Python program using the TinyOS Python SDK [24]. Since all motes are connected to one computer, packet loss due to colliding transmissions is avoided via central scheduling of transmissions. The software holds virtual representations of the mote, temperature controller and link, and logs experiment results to disk. A simple scripting language allows the definition of commands that are interpreted by the runtime and executed serially so that experiments can be described in independent and compact form. Multiple scripts can be added, so that experiments can run unsupervised around the clock.

During an experiment all transmissions and receptions are written to a log in ASCII format, which includes a timestamp, sending and receiving node ids, a sequence number, link quality metadata, mote temperature and the entire MPDU payload, including FCS.These logs describe the entire experiment in a unprocessed format, which is then read by the evaluation scripts that reassemble them into messages and links.

This modular software setup allows for a lot of freedom when designing these experiments, as all evalutation data is based upon these unprocessed logs. This enables

```
1 timestamp=2014-05-03 09:22:21,752   mode=tx id=0    seqnum=0
    temperature=29.9    length=93   data=[...]  power=3
2 timestamp=2014-05-03 09:22:21,753   mode=rx id=1    seqnum=0
    temperature=33.0    timeout=1
3
4 timestamp=2014-05-03 09:22:21,890   mode=tx id=1    seqnum=1
    temperature=33.0    length=93   data=[...]  power=3
5 timestamp=2014-05-03 09:22:21,890   mode=rx id=0    seqnum=1
    temperature=29.9    length=93   data=[...]  rssi=-90    lqi=104
    crc=1 timeout=0
```

Listing 4.1 A log excerpt showing two transmissions, of which the first reception timed out. The raw data field is ommited for clarity.

us to use one experiment for multiple purposes by focusing on different parameters, of which we will make use later in in Chapter 6.

5

Experiment Results

This chapter describes the experiment setups we designed to focus on the effect of temperature on the mote hardware and link quality. We present the evaluation of the experiment results and compare them to related work.

5.1 Locations

We executed these experiments in two locations: a temperature-controlled server room in a basement and a large empty storage space. We chose these locations for their low visitation frequency, since the presence of moving objects, especially humans, alters the link quality, which of course was undesirable.

In the server room, a space of about 3 m × 12 m was available, with a thick load-bearing concrete wall on the one side and metallic server racks on the other. This environment made for a very good link quality even at low power, which was the opposite of what we needed. We tried a power setting of 3 (−25 dBm), but even at the maximum attainable distance in the room the link was without bit errors, therefore we settled on power setting 2 (below −25 dBm), which required the motes to be very close to each other at about 30 cm. Due to the difficulty of manipulating link quality with fine granularity, we only ran one experiment in this room and then relocated the setup.

The storage room was larger with about 10 m × 15 m and had a window front and many shelves on the walls. Here we could use power setting 3 at about 3 m distance and very accurately manipulate link quality. Therefore we used this location for the remaining experiments.

5.2 Desired Link Quality

To create a link with the desired quality, we used a simple, live updated graphical display of LQI, RSSI and bit error values. While links of intermediate quality are relatively easy to find, we found it difficult to find the right intermediate link quality with a high enough BER for statistically significant results, but with also enough average message receptions, especially for a longer period of time. These temporal characteristics have also been described by Baccour et al. [2], specifically, that links of moderate average PRR are less stable than with very low or very high PRR. Furthermore, the mere presence of a person in the same room as the motes caused a dramatic change in link quality, which has also been described by Baccour et al.

We briefly considered using a controllable source of radio interference as proposed by Boano et al. [4] to be able to make it easier to create the desired link qualities, however, its effect on the BER patterns is unknown and we did not want to risk a corruption of our test results. Therefore, finding and verifying the desired link quality with the acceptable amount of bit errors regularly took hours, especially when the motes where to be heated to different temperatures. Even then, we had to repeat several experiments when link quality inexplicably changed. An automated link quality "creator" using a motorized mote harness would certainly have helped.

All experiments were done using the on-board PCB antenna sending on channel 26, which is outside the allotted spectrum of 802.11, to minimize WiFi interference [14].

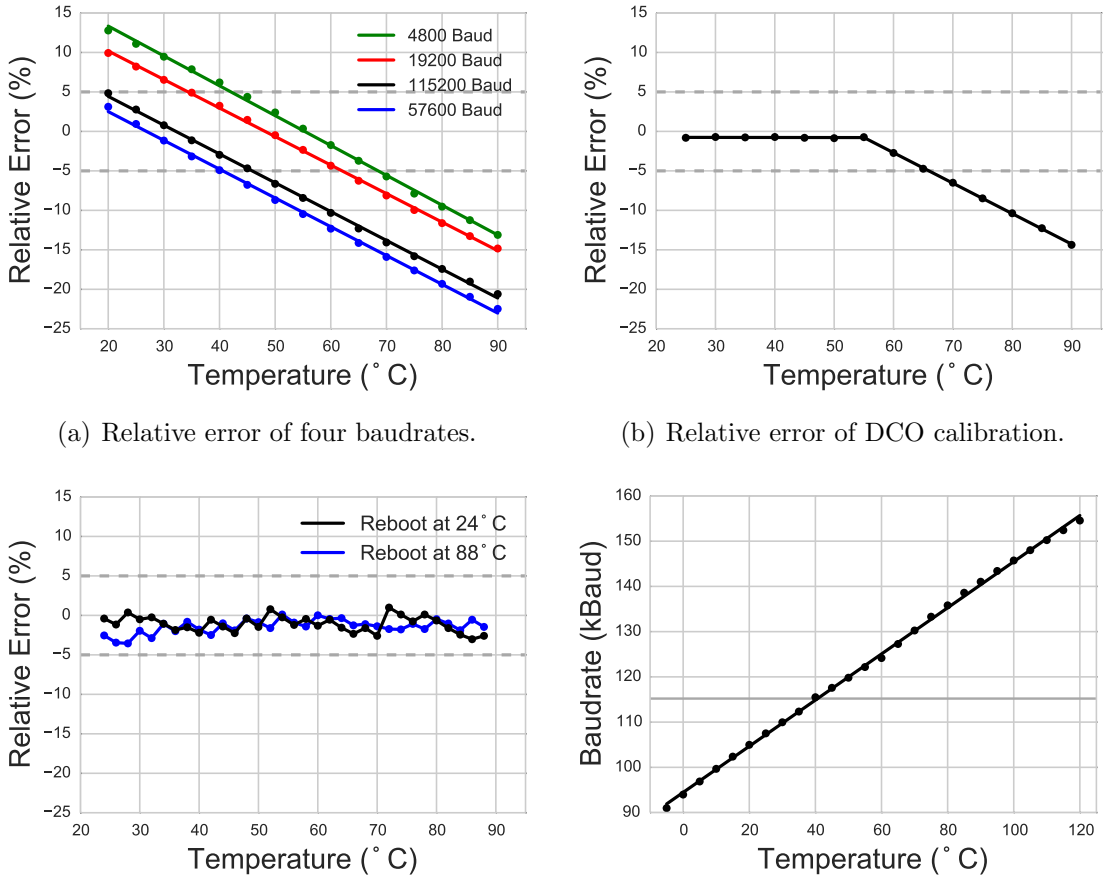
5.3 Microcontroller Clock Drift

The Tmote Sky uses a MSP430 microcontroller clocked by an integrated ring oscillator called the Digitally Controlled Oscillator (DCO). Since the generated clock frequency varies from chip to chip with temperature and operating voltage, the DCO can be fine-tuned in software using a modulation functionality. During booting, TinyOS calibrates the DCO using the external low-current 32.756 Hz watch crystal to generate a more accurate 1 MHz clock. It should be noted, that by default calibration only occurs after a reset, and not periodically during program execution.

We noticed that serial communication stopped working after heating the nodes above 55−60 °C. Below this temperature the problem could be mitigated by resetting the mote to trigger a recalibration of the DCO. Boano et al. reported similar synchronization trouble in their TempLab setup [7]. We therefore looked at the output of the UART module with a logic analyzer and measured how the selected baudrate changed over temperature. The UART receiver can tolerate up to ±5% relative error, however, with the relative errors of several baudrates reaching well over −20% as shown in Figure 5.1(a), resynchronization is impossible.

Since the baudrate is generated by scaling the DCO, the *relative* baudrate error is equivalent to *relative* DCO error over temperature. We calculated an average temperature clock drift coefficient of −0.367 %/°C, which is within the typical range according to the datasheet. Similar coefficients were found by Zúñiga et al. [25].

We then measured the relative baudrate error after rebooting over temperature. As shown in Figure 5.1(b), the TinyOS implementation of the DCO calibration



(a) Relative error of four baudrates.

(b) Relative error of DCO calibration.

(c) Relative error of corrected baudrate.

(d) Values of baudrate correction look-up table.

Figure 5.1 UART and DCO calibration errors over temperature. The dotted gray line indicates ±5% UART tolerance.

only works until 55 °C, after which it has no corrective effect on CPU frequency, making periodic DCO calibration during program execution ineffective. It is possible that Contiki OS [9] does not experience these problems as their DCO calibration algorithm is different from TinyOS. We did not test this, however.

To not influence the operation of the rest of the microcontroller and since understanding the DCO calibration algorithm is not the focus of this thesis, we chose not to correct DCO drift directly, but counteract only its effect on baudrate. To stay within the required tolerance of ±5% relative error, we created a fixed lookup-table of "inverse" correction baudrates for 115.2kBaud as shown in Figure 5.1(d).

Since calculation of prescaler values at runtime is very costly (requiring floating point arithmetic in a recursive algorithm) the look-up table only contains precalculated values, which are then copied into the registers at runtime. The on-board sensor provides temperature to the UART module, which then selects new prescaler values from the look-up table for every 5 °C temperature difference, which shows up as a sawtooth pattern in Figure 5.1(c) of the resulting relative error of the corrected baudrate.

We used this correction table without change on all of our motes and did not encounter this problem again. This shows that even with slight differences in clock

drift coefficient between different motes, this is approach is enough to solve this issue for our study.

Clock drift is not an issue for the communication between the MSP430 and the CC2420, as it is connected via the Serial Peripheral Interface (SPI), which is a synchronous master-slave bus that provides the communication clock for its slaves, requiring no synchronisation on data symbols.

5.4 Patterns in Bit Error Distributions

We designed two experiments to investigate the results of Schmidt et al. [20]. The first experiment was used to confirm their findings and also investigate the influence of hardware layout on BER patterns, while the second experiment investigated the influence of temperature on these patterns.

5.4.1 Effects of Board Layout

In all our experiments we used Tmote Sky motes, which are commercial drop-in replacements for the original Telos mote design by Polastre et al. [17]. We had hardware revisions available from two manufacturers: the original design from (now defunct) MoteIV Corporation and the Maxfor MTM-CM5000MSP, which uses a slightly different schematic and board layout. Notable differences include the 3 V regulator and the physical layout of the radio circuitry.

In the experiment a pair of CM5000 and original motes transmitted to two CM5000 and two original motes. All motes were powered by the on-board voltage regulator via the USB connector, which was also used to establish serial communications with the PC. This redundant placement shown in Figure 5.2 was chosen so that the same transmission was received by both types. Over the course of six days the four transmitters sent 563,500 messages each, totaling 2,254,000 transmitted messages at power setting 2 (below −25 dBm). Since each transmission was sent to four receivers, a total of 9,016,000 receptions should have been possible. Of those 5,280,369 were received and 2,497,744 had at least one bit error. The experiment was located in the large climate-controlled server room in the basement, where the temperature remained within 20–25 °C with no other changes in the environment.

The CM5000 motes required to be physically closer to the receivers at the same power setting to have similar link quality as the original motes. This might be hinting at a difference in range between the two hardware layouts, however, our experiment was not set up to systematically investigate range.

In the initial evaluation we noted some significant differences in the quality of some links, where the co-located transmitters are sending to the same receiver. For example, the link 3–5 is of very good quality with over 99% PRR, however link 2–5 shows quite the opposite with less than 1% PRR, even though both transmitters are located at the same distance and angle from the receiver. This confirms the findings of Baccour et al. [2], specifically that link quality is anisotropic, i.e., the communication range exhibits a non-spherical pattern. More exhibitions of this behavior can be found in the complete table of link qualifiers in the Appendix as Table A.1.

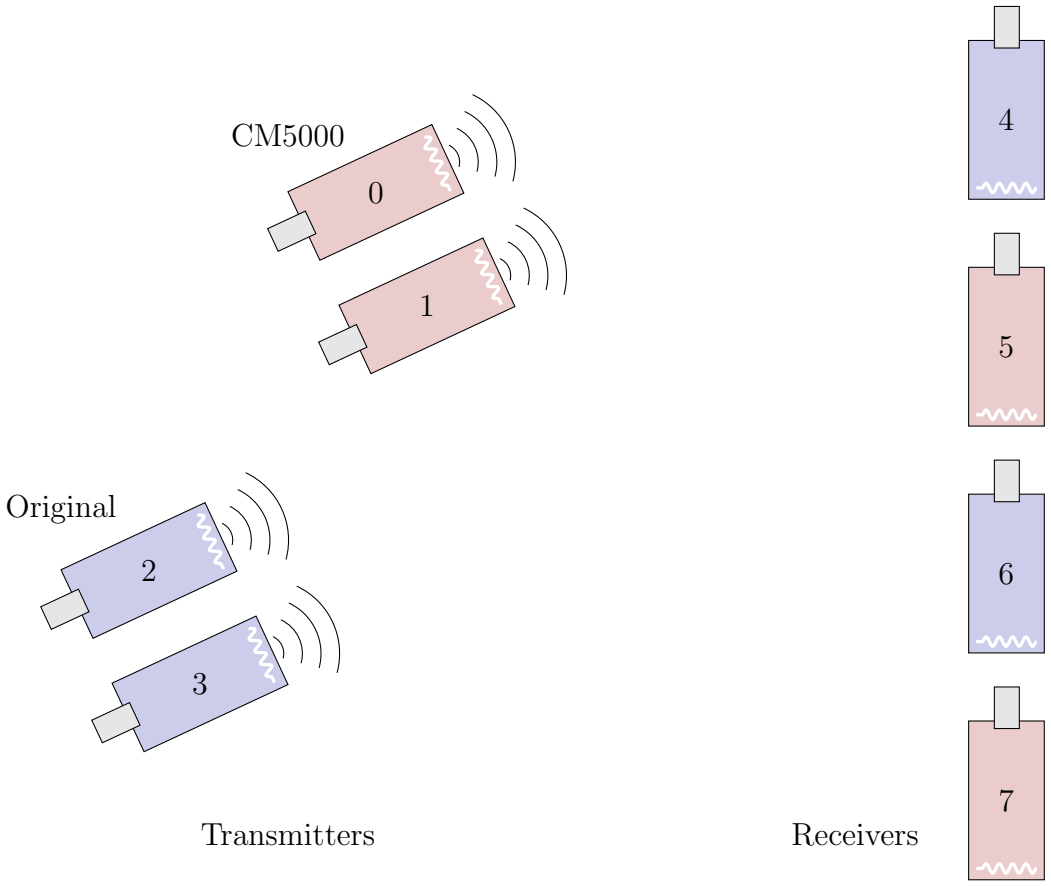
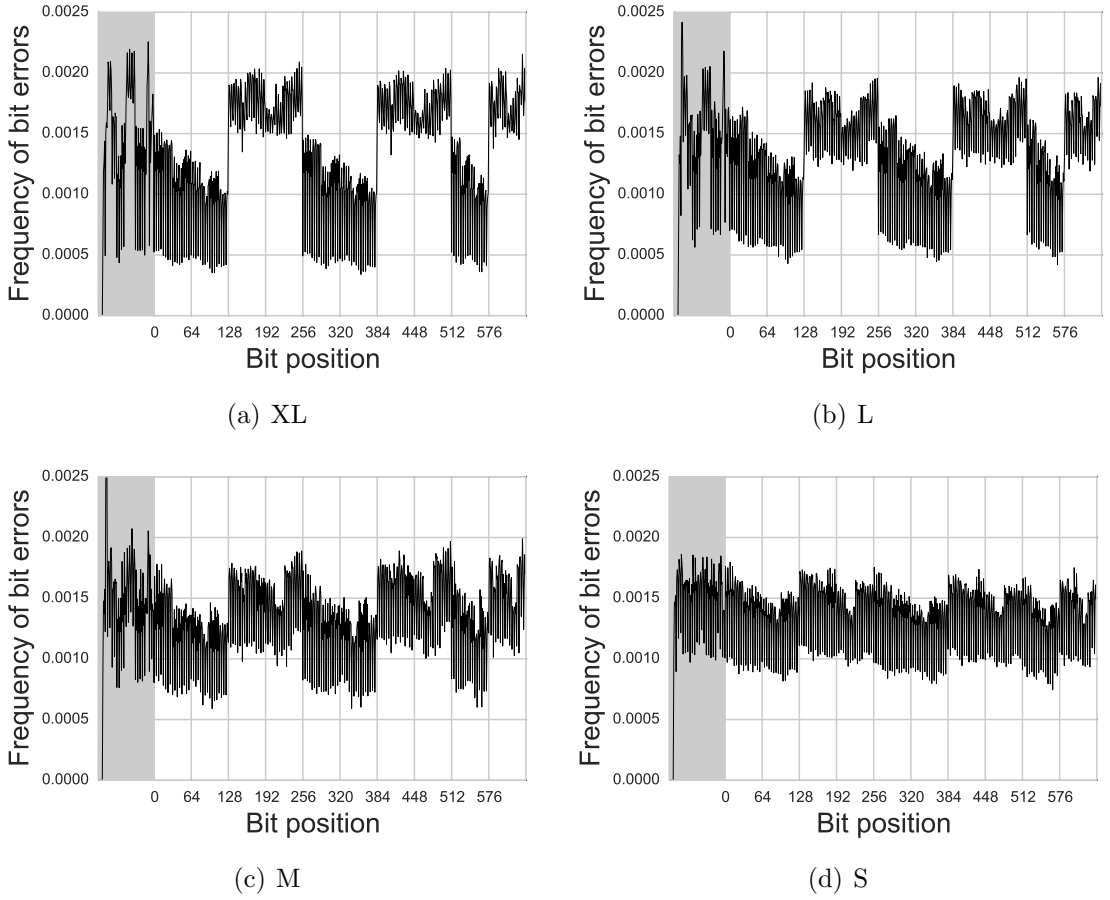


Figure 5.2 Experiment setup with four transmitters and four receivers.

Further analysis revealed the same bit and symbol error patterns as first discovered by Schmidt et al. [20], which state that within any transmitted symbol, the first three MSBs are more likely to break than the LSB and that symbols 8–15 with the MSB set to 1 are more likely to break than 0–7. The normalized occurance of bit errors are plotted in Figure 5.3, with the first 12 bytes (96 bits) consisting of the message header with partially fixed content and the remaining 80 bytes are the constant payload, made up of two 32 byte patterns of 0x0000, 0x1111, ... 0xFFFF, and one 16 byte pattern of 0x00, 0x11, ... 0xFF.

We were able to confirm the findings of Schmidt et al. and extend them with a classification of the influence of symbols on BER. The Subfigures in 5.3 show four magnitudes of this phenomenon, ranging from an extreme to almost no difference between symbols, named XL, L, M and S. The remaining links can be classified into these four categories as done in Table 5.1. XL and L patterns seems to be more common than M and S, with 8 vs. 3 links respectively, however, the table is incomplete, since only 11 out of 16 links had enough bit errors to recognize and categorize the pattern.

Schmidt et al. also looked into the burstiness of errors and showed in their research that burst errors are not independently distributed. The authors explained the drop from 4-bit bursts to 5-bit bursts with the added difficulty of corrupting a minimum of 7 bits to overcome the minimum Hamming distance of 12 between two symbol's chip sequences. They also hypothesized that a similar drop would occur between 8-bit and 9-bit bursts, but were unable to confirm this, due to their limited sample



(a) XL (b) L

(c) M (d) S

Figure 5.3 Four magnitudes of BER patterns with fixed payload of twice 0x0000, 0x1111, ... 0xFFFF and once 0x00, 0x11, ... 0xFF. The message header is indicated in gray.

size. With our larger sample size we can confirm their hypothesis, with the drop quite noticeable in Figure 5.4(a).

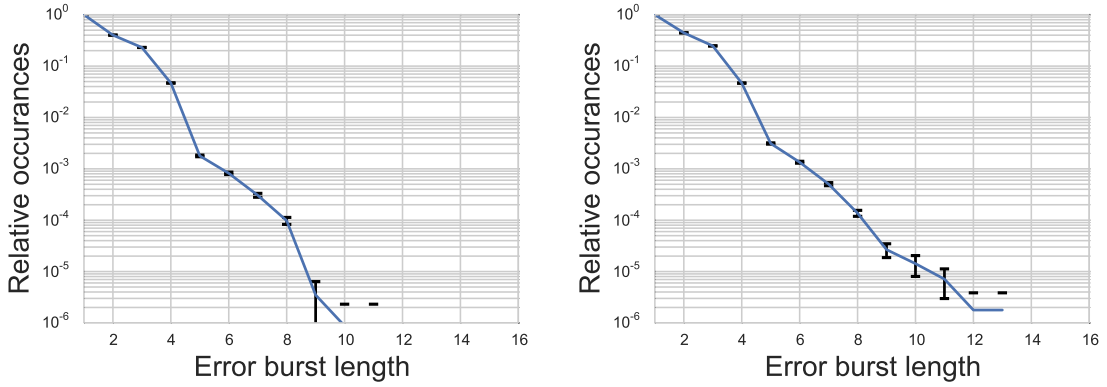
Receiver	Sender 0	Sender 1	Sender 2	Sender 3
4	S			
5	XL	XL	L	XL
6	XL	L	M	
7	L		S	L

Table 5.1 Classification of all links with enough absolute bit errors (otherwise gray).

We cannot see any correlation between the four BER pattern classifications and burst error distribution or any other variable. We therefore believe the magnitude of these patterns to be a result of the analog circuits of the radio, something that cannot be changed or improved by software.

5.4.2 Effects of Temperature

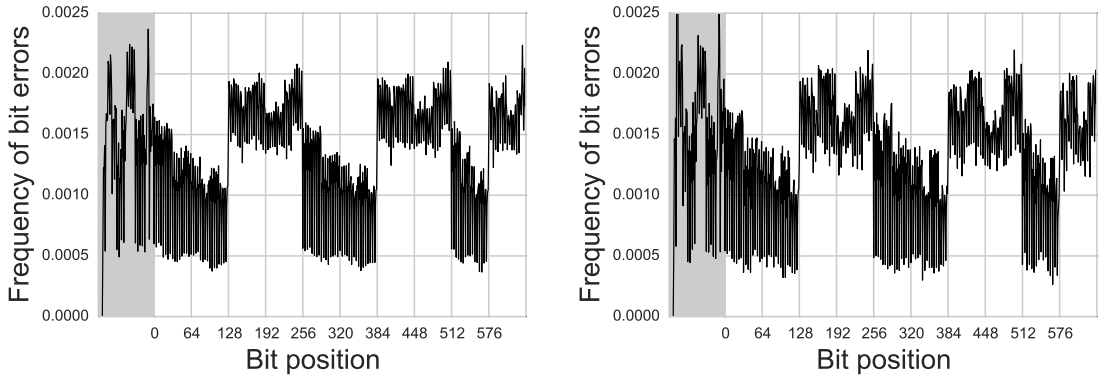
To investigate the influence of temperature on BER patterns, we used the CM5000 motes in the temperature boxes as described in Section 4.1. To minimize signal



(a) XL burst error distribution.

(b) S burst error distribution.

Figure 5.4 The burst error distributions of XL and S magnitudes show more longer burst errors in S magnitude. The error bars denote 99% confidence intervals.



(a) BER pattern at 50 °C.

(b) BER pattern at 70 °C.

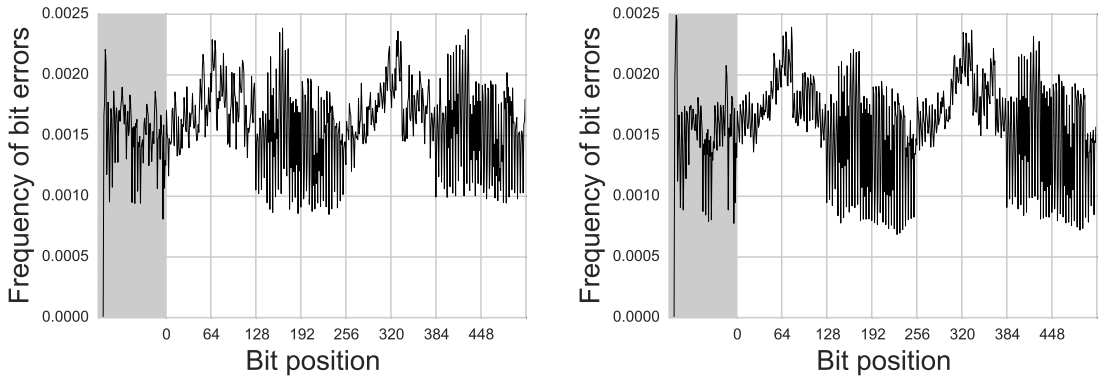
Figure 5.5 Bit Error Rate patterns show no difference between 30 °C, 50 °C and 70 °C. The pattern at 30 °C is omitted, since it is redundant.

disturbance we mounted the boxes in the storage room on wooden posts, so that they were floating in free space. We placed the boxes at a fixed distance of 280 cm to each other and controlled link quality sorely by rotating the antenna.

We ran three experiments with the same constant payload as described in the Subsection 5.4.1 but with power setting 3 (−25 dBm) at 30 °C, 50 °C and 70 °C. Due to the decrease of link quality at higher temperatures, discussed in detail in Section 5.5, we had to adjust mote rotation to regulate BER at these temperatures, therefore small differences in absolute bit errors are present. However, Figure 5.5 shows no significant difference between BER patterns at all three temperatures and therefore we did not pursuit this further.

5.4.3 Pattern Anomalies

Even though the vast majority of experiment evaluations confirmed these patterns, there were two runs with drastically different outcome. We were very surprised to find a second, completely opposite BER pattern as shown in Figure 5.6. Here



(a) First occurance.

(b) Second occurance.

Figure 5.6 "Inverted" BER pattern anomaly of constant data. Only the first two 32 byte patterns are shown.

the symbols 0-7 show a higher average BER and a less pronounced 4-bit saw-tooth pattern than symbols 8-15. The pattern almost looks like an "inverse" of those described earlier.

The first time it occurred, we attributed this to influences of the environment and discarded it. Two months later, this pattern occurred again with different hardware in a different setup in a different environment. Both times, the motes were exposed to high temperature (>70 °C) over several hours, however, we were unable to confirm if this indeed triggered the pattern. This pattern remained for days, regardless of temperature, link quality or message payload.

We cannot envision a source of interference capable of fundamentally changing this BER distribution and neither does a permanent change of the radio chip's hardware at high temperature seem possible. The CC2420 radio is rated to be operated at temperatures up to 85 °C and to be stored at up to 150 °C, while our experiments where limited to at most 90 °C for a few hours.

Since we could not find any other reference to this pattern in literature and due to the rarity of this occurrence in our experiments and the difficulty of its repeatability, we decided to focus on investigating the "normal" patterns.

5.5 Packet Reception Rate

The effect of temperature and other environmental factors on link quality has most thoroughly been investigated with the conclusion that all measurements of link quality deteriorate with higher temperature, as one would expect [26, 6, 7, 25]. However, Boano et al. showed in their experiments that this behavior is not symmetrical, since heating the transmitter caused a more significant drop in PRR than heating the receiver [6].

In this experiment we used the same physical setup as described in Section 5.4.2, but with random payload as message content. We fixed mote rotation to create a link that is just starting to deteriorate at 50 °C, so that at low temperatures the

link is of good quality, while at very high temperatures the link will experience high BERs.

We increased the temperature of the mote in box 0 in steps of 5 °C and 10 °C up to 80 °C and kept the mote on box 1 at constant 30 °C, while sending 180.000 messages back and forth. Then we repeated this, but heated box 1 and kept box 0 temperature constant. To cancel out environmental factors we ran this experiment four times over two days, so we could then pick the two results with the least interference.

For the evaluation we plotted the normalized PRR of messages sent from one mote and received by the other mote over temperature and time and included BER and LQI and RSSI values to illustrate link quality. Figure 5.7 shows messages sent from mote 0 addressed to mote 1 for both temperature cycles, while Figure 5.8 shows the opposite direction. It becomes immediately clear that a higher temperature of either transmitter or receiver makes the link quality worse, validating the findings of Boano et al. and Wennerström et al. [6, 26]. However, this relationship is neither linear nor symmetrical.

5.5.1 Effects of Temperature on PRR and BER

In Figure 5.7 the decrease in PRR is small and relatively symmetrical up to about 65 °C, however, past that point we see a dramatic change, with the increments in temperature translating very strongly into significant drops in PRR. The drop in PRR is not symmetrical and becomes much more pronounced, when the receiver is heated than when the transmitter is heated, especially visible in the last increment from 70 °C to 80 °C.

This asymmetry is even more extreme in link 1-0, shown in Figure 5.8, where heating the receiver beyond 70 °C will cause an almost complete loss of message reception. Interesting is the little dip in PRR around 50 °C in Figure 5.8(a), which was present in this link in all four experiments with varying intensity. We suspect that this is a non-linearity in the radio, cases of which have also been reported by Boano et al.

The BER acts like an inverse function of PRR, since higher BER yields more messages with at least one bit error. Noise on PRR is visible in the standard deviation of BER as exemplified by Figure 5.8(b).

5.5.2 Effects of Temperature on LQI and RSSI

The LQI is a very good mirror of the PRR of messages without error. When the receiver is kept at a constant temperature, the values decrease almost linearly with temperature of the transmitter. This is not the case when the receiver is heated, where a linear correlation to temperature , but is still very similar to the behavior of PRR. We therefore can confirm that LQI is a good source for an estimate on PRR.

This is very much not the case with RSSI, which has much lower resolution that LQI and exhibits hysteresis and non-linearities [6]. While in Figure 5.7, RSSI ends up being lower when the receiver is heated than when it is constant, Figure 5.8 shows very similar values, even though PRR is radically different.

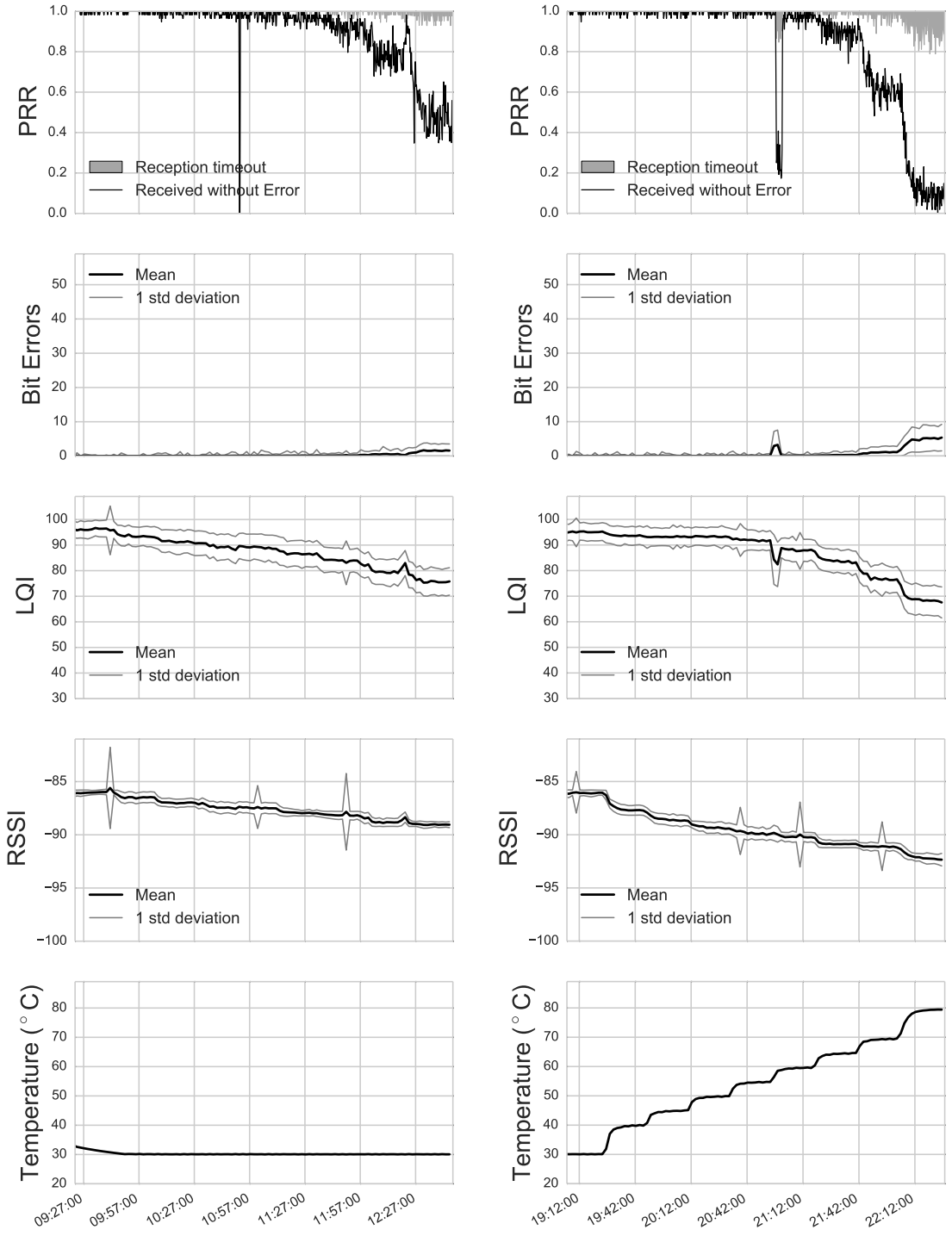
It is also noteworthy that contrary to LQI, RSSI attempts to describe signal strength (i.e., the power level being received by the antenna), which of course does not change, when the transmitter is at constant temperature. Therefore, without temperature information, the RSSI value is misleading, since it represents the power level not at the antenna, but at the signal amplifying stage of the radio. Therefore RSSI is usable for a very inaccurate estimate of link quality at best, with little to no difference between receiver and transmitter temperature.

5.5.3 Discussion

Our data strongly implicates the receiver as being more vulnerable than the transmitter to an increase in temperature, especially above 65 °C. These results are very much incompatible with the findings of Boano et al. [6], which is surprising since the only two differences between our experiment and theirs is the temperature range and the additional usage of the CC2520 [21] radio, which is very similar to the CC2420 [22]. However, these differences should not account for completely opposing results.

Research by Bannister et al. [3] saw an asymmetry in output and received input power when heating transmitter and receiver separately. At the same time, they measured Packet Error Rate (PER) and found that above −90 dBm RSSI, temperature had little to no influence, while below that RSSI value, PER increased. The authors concluded that the CC2420 Low Noise Amplifier stage is less efficient at high temperature, which would manifest itself in a significant increase in PER (or a *decrease* in error-free PRR) on the receiver. However, the RSSI behavior in Figure 5.8 does not reflect a general truth of the results by Bannister et al. and furthermore suggests no direct correlation between RSSI and PRR.

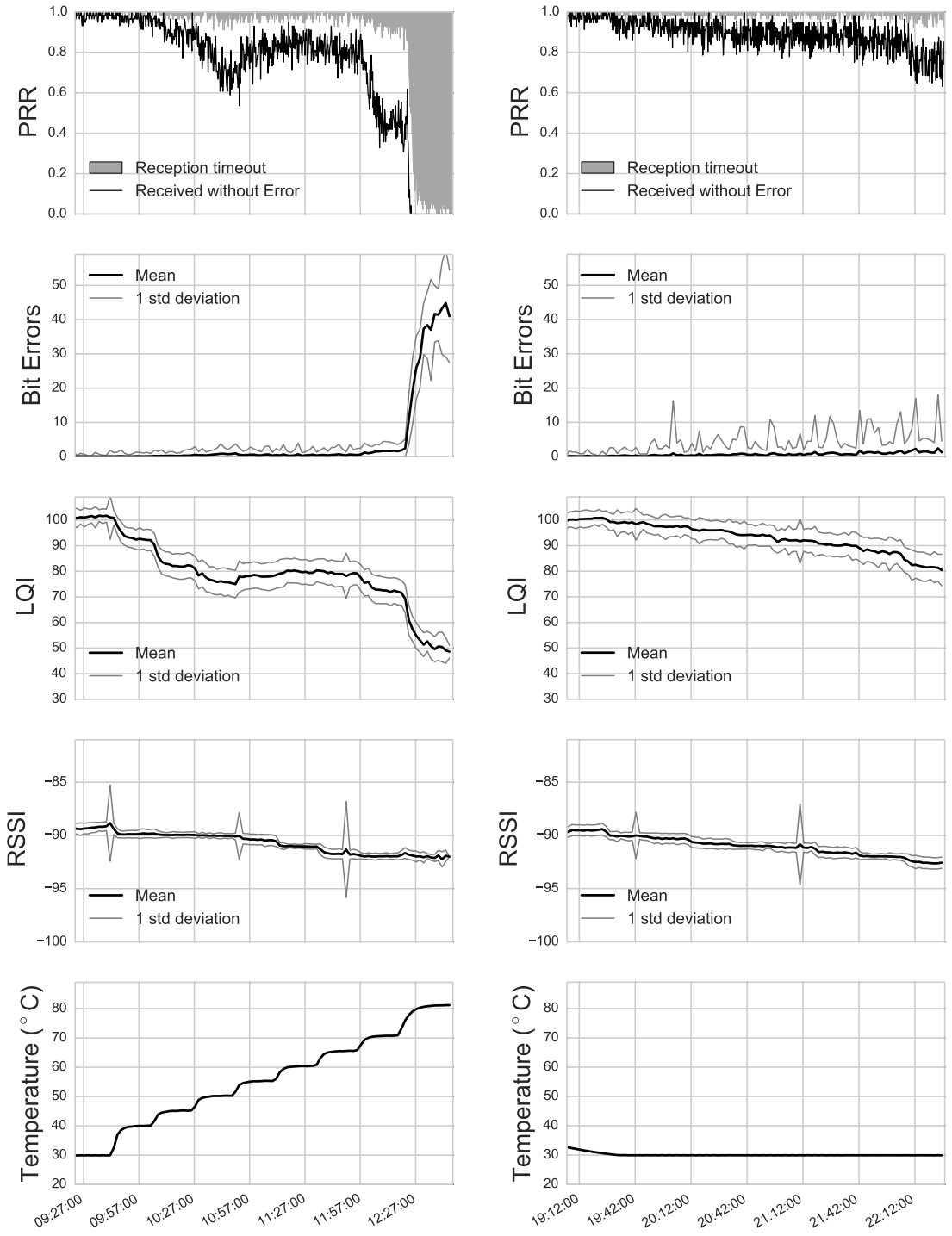
We noticed a strong focus in such research on understanding the impact of temperature on output and input signal power, however, it seems that RSSI does not actually correspond to link quality in general. From our data, a combination of LQI and temperature seems to have the best correlation to PRR, therefore we want to examine how we can use temperature to improve PRR in these conditions.



(a) Constant receiver, increasing transmitter temperature.

(b) Increasing receiver, constant transmitter temperature.

Figure 5.7 PRR and link quality of messages received by mote **1** vs. temperature. For transmitter temperature see Figure 5.8.



(a) Increasing receiver, constant transmitter temperature.

(b) Constant receiver, increasing transmitter temperature.

Figure 5.8 PRR and link quality of messages received by mote **0** vs. temperature. For transmitter temperature see Figure 5.7. The asymmetry in PRR shows much more clearly here.

6

Forward Error Correction

In the last chapters, we described the effect of temperature on BER patterns and PRR, without drafting any specific recommendations how to improve this loss of link quality. FEC is a well known technique to improve throughput in links of poor quality, however, once a ECC and its parameters have been chosen, usually they remain static and do not change over time. This means that for links where the quality is not stable over time, as in WSNs, the chosen ECC incurs unneeded overhead in links with overal good link quality, and underperforms in links with poor quality.

We therefore wanted to investigate how using temperature as an indicator for adapting ECC parameter can benefit PRR and throughput and what improvements and limitations to expect. In doing so, we are moving away from a macro-view of all influences on link quality to focus on the impact of temperature on FEC on *one* link at a time.

We extended the control software described in Section 4.3 to encode messages to be sent and to decode received messages with an FEC scheme. The messages are logged in the same format, with the additional information of the FEC scheme and coding strength. This allows us to reuse our existing evaluation software to easily map error-free PRR to *decoded* error-free PRR.

6.1 Choice of FEC Scheme

Choosing a coding scheme is not only a matter of its error correction capabilities, but also about the complexity of its coder and decoder, considering the tight computational resources of a WSN mote. Practical implementations are therefore limited to cyclic linear block coding, with the most widely used codes being binary BCH and non-binary RS codes, which can be combined with interleaving and code shortening [15]. Furthermore, since data payload size is limited to 125 bytes by the MPDU,

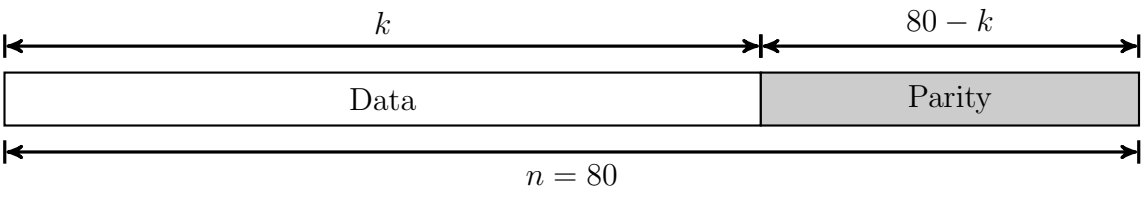


Figure 6.1 Structure of an $RS(n, k)$ encoded codeword of 80 bytes length.

of which we use 93 in our messages (with 80 bytes data), a low coding overhead among ECCs with the same error correction capability is preferred.

A RS code works over m-bit symbols, and is denoted as $RS(n, k)$, where n is the number of m-bit symbols in a codeword, and k the number of original m-bit data symbols. This leaves $n-k$ parity m-bit symbols, as shown in Figure 6.1. The symbol size m can be set to bit-level, byte-level or packet-level size. We will use byte-level ($m = 8$) symbol sizes, since they are efficient to work with on a microcontroller. A RS decoder can correct up to $t = (n - k)/2$ symbol errors, and up to $2t$ erasures, if the positions of the symbol errors are known [15].

Similarly, for symbol size $m \geq 3$ and symbol error occurance $t < 2^{m-1}$, a BCH code encodes block lengths of $n = 2^m - 1$ bit with $n - k \leq mt$ parity check bits by multiplication with a generator matrix, that contains a $n \times n$ identity and a $n \times (n - k)$ binary matrix. The decoder can construct a parity matrix using this information, which allows correction of t-bit errors, depending on the parameters chosen [15].

Since the RS scheme works by correcting entire m-bit symbols ($m = 8$ for our case), burst errors up to m-bit in the same symbol require only correcting this specific symbol, while one-bit errors in many different symbols requires correcting all of these symbols. RS codes therefore perform better than BCH codes in conditions with burst errors as described in Subsection 5.4.1, but worse if same amount of bit errors are spread around more independently. The performance of BCH codes can be improved however, by interleaving symbols after encoding before transmission, which spreads out burst errors into many single bit errors. However, BCH codes generally require more transmission overhead compared with RS codes of the same error correction capabilities.

RS codes are well understood coding schemes, which have been compared to many others. By using RS codes as a benchmark we enable application of our results onto other, more complex schemes, such as a modified Turbo Code [19] and Low-Density Parity-Check (LDPC) [18] codes, which have been shown to outperform cyclic linear block codes. Furthermore, a RS implementation optimized for TinyOS, called TinyRS [14], exists and therefore does not need to be implemented manually.

6.2 RS Scheme Simulation

To be able to investigate the effect that different RS coding strengths have on decoded PRR, we needed a way to compare experiment results. Due to the difficulty of recreating the *exact* same conditions for several series of real world experiments,

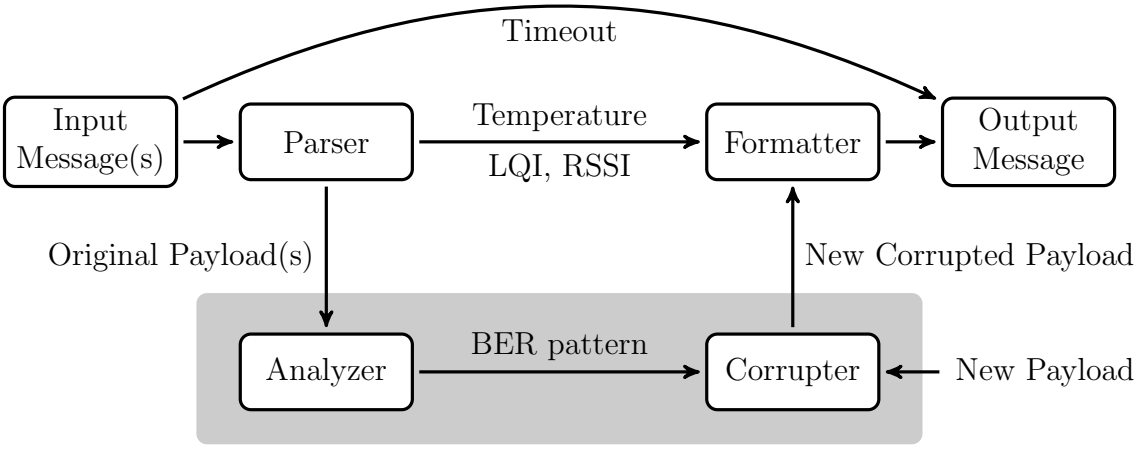


Figure 6.2 Schematic design of our trace-based simulator: the payloads of multiple original messages can be analyzed to extract the BER pattern, which is then applied to new payload.

we chose to create and use a trace-based simulator to apply the bit error patterns of an original experiment onto several new $RS(n, k)$ encoded random payloads.

By interpreting the $RS(80, 70)$ encoded random payload as pseudo-random in the context of Section 5.5, we made dual-use of that experiment. This allows us an in-depth description of the link conditions onto which we built our analysis of RS performance.

We will next describe how we use the raw data of this experiment as an input for our simulation, by extracting its BER patterns and applying them onto new payload, and discuss accuracy and limitations.

6.2.1 Design

The simulator re-runs an original experiment message-by-message over its log, and outputs a new one. We only simulate BER patterns, all other properties of the link, such as link qualifiers, temperature, transmission power, are copied from the original link, as visualized in Figure 6.2. In addition, if a message was not received by a receiver, we simply preserve it as a timeout, to make later comparison of the simulated results with the real experiments much easier.

Our simulator also works with experiments where a transmitted message is received by multiple receivers as in Section 5.4.1. In such a case, the simulator will apply its algorithm to every received message.

Pattern Extraction

Since we know that the BER pattern is different for every of the 16 4-bit symbols, we need to generate a corruption probability for each bit within every symbol. For that we sum up the bit errors per bit *per symbol* for every original received message in the experiment in a corruption table. The corruption table is then normalized for the occurrence of the respective symbol in the original message to yield a relative bit error occurrences *per symbol* for this specific link.

This table can be averaged over several links using a sliding window, which reduces noise and increases the likelihood of seeing all symbols with their respective bit error patterns at least once. This will create a corruption table of *average* bit error probabilities per symbol over one or more messages. We also average the distribution of bit error burst lengths over these messages in a burst error table. Both the corruption and the burst error table will then be handed over to the corruption generator.

Since these tables are generated on-the-fly entirely from an experiment log, even the anomalous patterns described in Section 5.4.3 can be extracted.

Pattern Application

A corruption pattern is generated by randomly corrupting each new symbol with the probability defined in the bit error table. This corruption pattern needs to be adapted for the burst error distribution of the original message by lengthening single bit errors accordingly.

The corruption sequence is applied to the new payload defined by us and written out as a new received message of the original link.

Limitations

Since we map and apply bit error distribution *per symbol*, every symbol must be available in the original payload at least once for this table to be complete. Experiment logs with constant payload in which not every symbol is available should be avoided. For random payload our message size of 93 bytes yields 186 symbols, for which it is unlikely to not see every 16 symbols at least once.

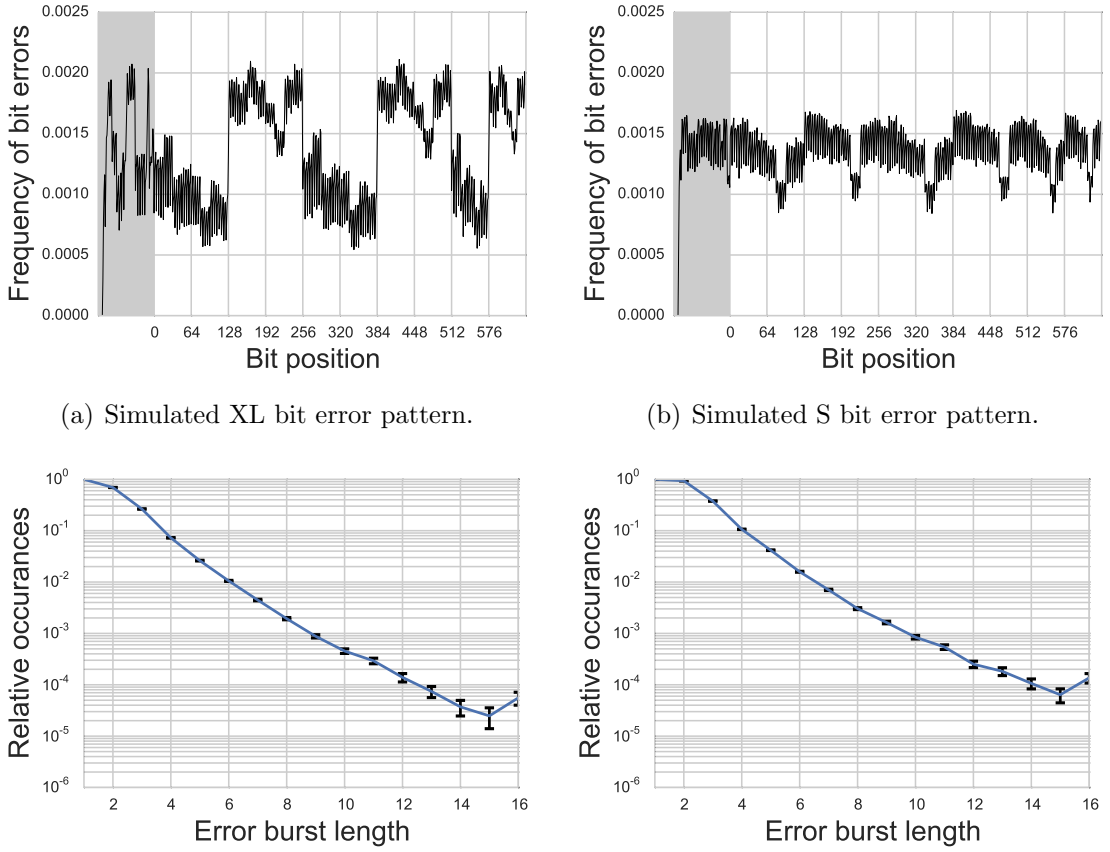
Another limitation is our modeling of the burst error distribution. By "simply" applying the collected bit error table by symbol onto new payload, we are effectively interleaving the original burst errors and therefore obscuring this property of the original link. With our simple simulation, we can only add burst errors to correct for relative occurrence compared to the original frame, but not willfully "place" these burst errors at the correct position to achieve the distinct burst error properties of the original link.

6.2.2 Accuracy

To evaluate the performance of the simulator, we ran it on experiment traces containing constant payloads as well as $RS(80, 70)$ encoded payloads. By tuning the link input window size and the burst error generator, we were able to replicate similar enough bit error patterns for our purposes. We found best accuracy with a window size of two messages.

Bit Error Distribution Patterns

Figure 6.3(a) and 6.3(b) show the bit error pattern resulting of simulating XL and S magnitudes of the experiment described in Subsection 5.4.1. Compared to the original patterns in Figure 5.3, the simulation generates a less noisy footprint with less



(a) Simulated XL bit error pattern.

(b) Simulated S bit error pattern.

(c) Simulated XL burst error length.

(d) Simulated S burst error length.

Figure 6.3 Error patterns of simulated constant payload using traces with constant payload. The error bars denote 99% confidence intervals.

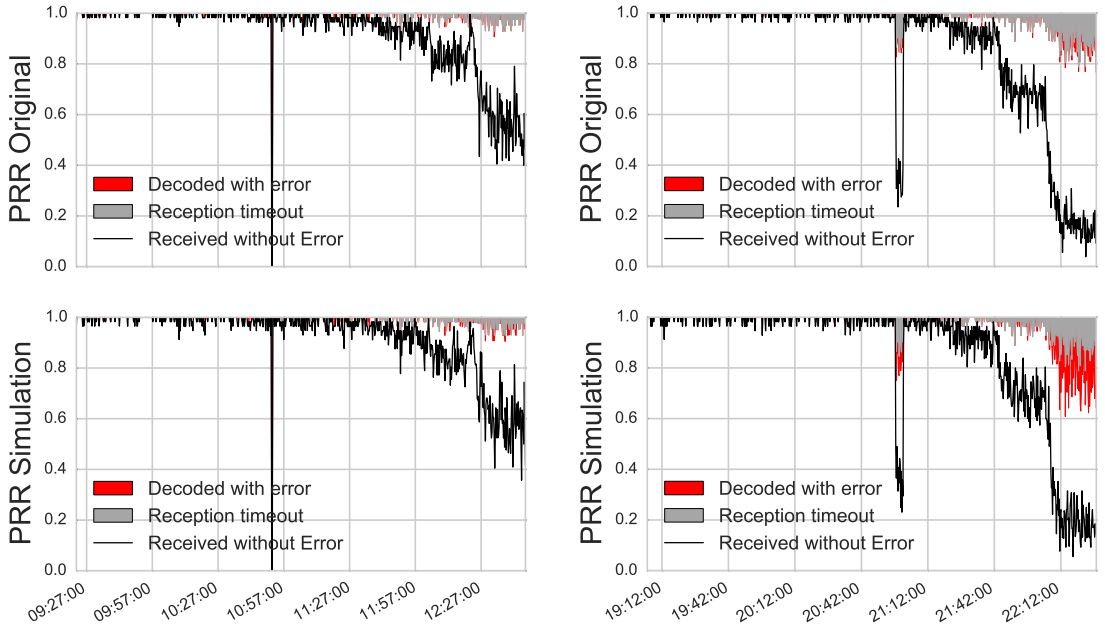
amplitude, which makes the differences in symbols clearly visible. This is, of course, due to the averaging during the symbol error construction that is accompanied with the smoothing of these values.

Burst Error Distribution

The limitations of our burst error modeling clearly show in the burst error graph in Figure 6.3(c) of the simulation. In comparison to Figure 5.4, the simulated corruption does not exhibit the drop between 4/5-bit and 8/9-bit bursts and generates longer bursts more frequently.

Packet Reception Rate

To compare the overall number of corrupted messages of the original with the simulated link, we plotted the normalized PRR of our original experiment with $RS(80, 70)$ encoded payload and the simulation of that experiment with the same payload, as shown in Figures 6.4 and 6.5 respectively. To be able to compare the PRR of encoded and non-encoded payload, we only evaluate the first $k = 70$ bytes in the payload. The timeouts shown in gray are the same for the simulation, since they are copied from the original, as mentioned before.



(a) Comparison of link 5.7(a).

(b) Comparison of link 5.7(b).

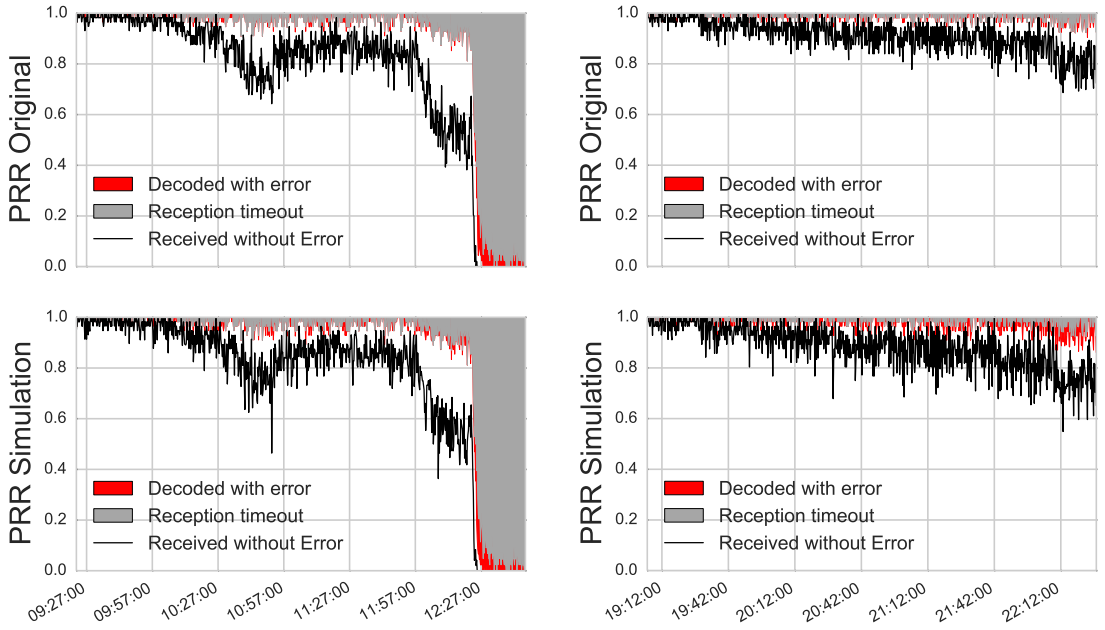
Figure 6.4 Original and simulated PRR of the two $RS(80, 70)$ encoded links of Figure 5.7. Note the drop in simulated RS decoded PRR in (b).

While the total amount of corrupted messages in the simulated link is the same as in the original link, as exemplified by the very similar error-free reception curves in the plots, error-free RS decoded receptions yields slightly worse results, especially in areas with high byte error count, as visible in Figure 6.4(b) and 6.5(b). This shows that in these areas, the simulator overshoots the target bit error distribution of the original messages, and therefore undershoots the RS decoded PRR of the original link. In the worst case, the simulated RS performance is underestimated compared to reality, which is sufficient for our assessments.

These results show that this simple simulator is a very good time- and cost-saving alternative to testing the effects of different payloads over a real link, if good original data is available. Using our trace-based simulator eliminates the effect of uncontrollable environmental factors or hardware issues on repeatability and allows us to focus purely on the message content, without assuming idealized and monolithic link properties.

6.3 Comparing RS Scheme Strengths

Our first idea was to keep a constant data size of $k = 80$ and add parity bytes, making the entire message longer. However, longer packets require more energy to transmit, a metric which has to be included somehow when comparing different k. Furthermore, because of the size limit of the MPDU of 127 bytes, of which 26 are already used, this only leaves $n = 101$ bytes with a maximum of $n - k = 19$ parity bytes, which would not allow to encode with more than 18% coding overhead. We could have split the message into two packets, however, this would have made it



(a) Comparison of link 5.8(a).

(b) Comparison of link 5.8(b).

Figure 6.5 Original and simulated PRR of the two $RS(80, 70)$ encoded links of Figure 5.8.

even harder to compare them, since now two packets must be received in the right sequence.

Therefore, we abandoned this idea and fit the entire encoded message into 80 bytes, including parity bytes as shown in Figure 6.1. This means we can only transfer k bytes of encoded data, effectively reducing data rate while gaining robustness, which we want to use for a metric for comparing RS performance. We combine this reduction of data rate and the corruption of decoded packets as throughput, normalized over the number of *received* messages. We calculated the normalized throughput $T(80, k)$ using the following formula:

$$T(80, k) = \frac{k}{80} \frac{PRR_{decoded}}{PRR_{received}}$$

This metric allows us to compare RS performance not only between different coding strengths, but augment that comparison with context of the link's quality.

For each of the four links discussed in Section 5.5 we simulated new $RS(80, k)$ encoded payload for k in 10 byte increments up to $k = 60$, then in 2 byte increments up to $k = 78$. We plotted the normalized throughputs $T(80, k)$ in Figure 6.6, with the dashed line showing throughput without RS encoding. The graphs illustrate that even using only 2 parity bytes at $k = 78$ can already significantly improve PRR, since most messages only have a few burst errors, and all those constrained to one byte can be corrected.

Adding two more parity bytes for $k = 76$ improves PRR even more, but only in a few areas. From $k = 74$ to $k = 60$ no significant improvement shows, especially at low temperatures. Using $k = 60$ shows stability up to high temperature of 80 °C in all Figures except 6.6(c). At and below $k = 50$ no improvement of throughput

is visible except in Figure 6.6(c), where $k = 30$ and $k = 20$ hold throughput the longest.

By comparing Figure 6.4(b) and 6.6(b), we can deduct that in that link the simulated throughput above 70 °C is actually worse than in reality. Therefore we can safely assume $k = 60$ as stable at that temperature.

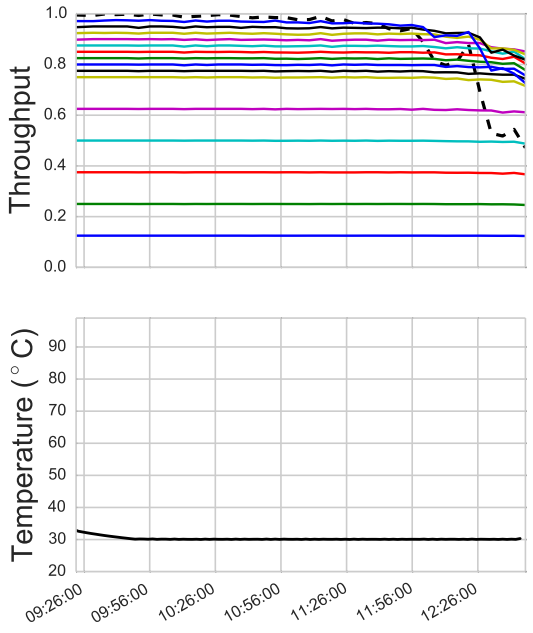
6.4 Discussion

Considering that the goal of using an FEC in the context of low power networks such as WSNs is to maximize energy efficiency of communications, we are not only looking for the k with maximum throughput, but also with the least retransmissions. We also have to consider that the link quality can already be poor at low temperature, therefore using no parity bytes at low temperatures is not a good idea either.

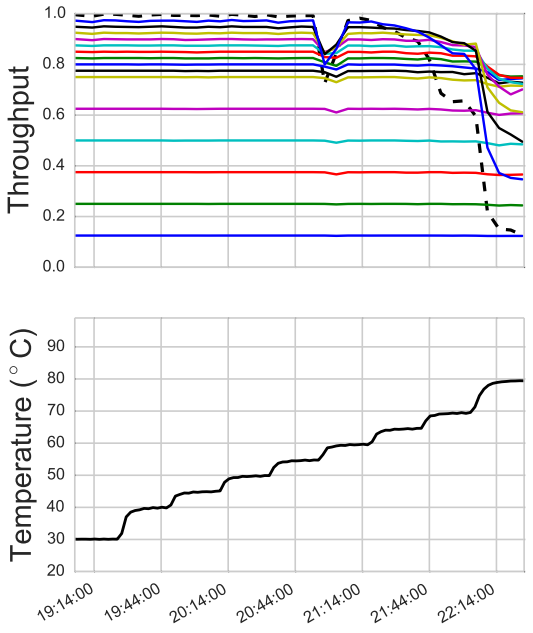
As per our findings, we propose one possible RS scheme starting with $k = 70$, which is 12.5% overhead (the same as the popular $RS(255, 223)$ [16]), at low temperatures and linearly increase this to $k = 60$ (or 25% overhead) as receiver temperature increases to 70 °C and above. Starting with $k = 70$ will give some protection against a random decrease in link quality as seen in Figure 6.6(b). In cases of extremely high bit error, such as in Figure 6.6(c), we can still regain some throughput by using $60 - 80\%$ coding overhead with $k = 30$ or $k = 20$. However, considering the extremely low PRR in this area (compare with Figure 6.5(a)), most of these messages will likely never be received anyway, which makes this option only viable when communication at these temperatures is absolutely necessary.

The results of Boano et al. [6] strongly suggested that the loss in PRR is more pronounced when heating the transmitter than the receiver. This would have allowed us to use local temperature measurements to adapt the coding strength of our FEC scheme to counteract the loss in PRR. Unfortunately, our results firmly oppose the findings of Boano et al.: heating the receiver creates a higher loss in PRR than heating the transmitter, therefore making such an approach unusable. However, using temperature as another source for assessing link quality has the distinct advantage of being always available on the receiver locally, compared to LQI and RSSI, which requires a message to be received first.

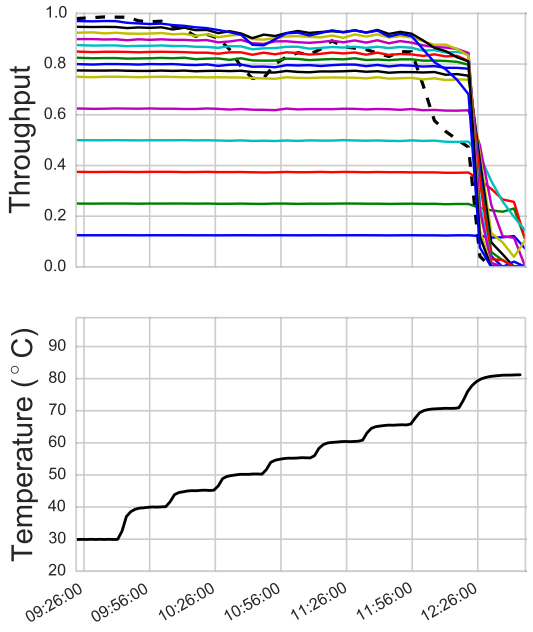
We therefore propose another solution to adapt FEC strength: the receiver should monitor its local temperature and send out a warning broadcast to all potential transmitters, before its temperature becomes too high. The transmitters can then address this mote with the appropriate FEC strength. The advantage of this active, preemptive approach over backchanneling link quality information is that in setups where transmissions only occur sparsely, a “test” transmission to judge link quality is not required.



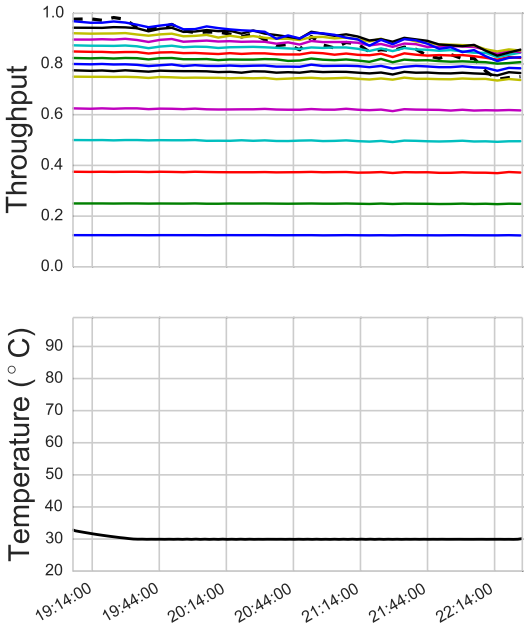
(a) Simulated link 5.7(a).



(b) Simulated link 5.7(b).



(c) Simulated link 5.8(a).



(d) Simulated link 5.8(b).

Figure 6.6 Comparison of throughputs $T(80, k)$ of all four simulated links of Section 5.5 for $k \in \{10, 20, 30, 40, 50, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78\}$ over temperature. The dashed line shows throughput at $k = 80$, which is equivalent to using no parity bytes.

7

Conclusion

This work investigated the effects of temperature on bit error patterns and PRR in the context of low power communications as used in WSNs.

We began with a description of the low-cost design of our testbed, which allows fine control of mote temperature, mote orientation and experiment execution. This testbed was the foundation, which allowed us to create and control links with high BER between CC2420 radios at different temperatures. Due to problems with the serial communication between mote and PC, we measured microcontroller clock drift and found the DCOcalibration algorithm in TinyOS not to be working above 55 °C, and provided and tested a correction method.

Then we showed that bit errors follow a distinct distribution, which is influenced by payload content, confirming findings by Schmidt et al. [20] and extending them with a classification of pattern magnitudes. We found two bit error patterns, which we did not know before, due to their rare occurrence. Furthermore, we saw no influence of temperature on these patterns.

The experiments were completed with a detailed investigation into PRR asymmetry, when transmitter and receiver were heated to different temperatures. Our results showed that the receiver is more vulnerable to an increase in temperature than the transmitter, thereby disproving findings by Boano et al. [6].

We used these findings to create a trace-based simulator to study the effects of temperature on FEC usage, and evaluated the simulators performance, which we found accurate. Using traces of our real experiments, we simulated several RS encoding strengths and compared them based on normalized throughput. On this basis we proposed a minimum encoding strength of 12.5% overhead, which is increased with temperature to a maximum of 25% overhead.

Future Work

The results of our experiment warrant further investigations into several topics.

In Section 5.2 we briefly described the difficulty we encountered in creating a link with the right BER even when using our mote harness from Section 4.2. An interesting addition to this harness could be a motor controlled z-axis to further automate link quality selection via antenna orientation in a closed-loop controller setup. In that respect, it might be more practical to rebuild this harness partially out of LEGO Technic parts, which would make the mechanic aspects of this function much easier.

Concerning the limitations of the DCO calibration algorithm of TinyOS described in Section 5.3, we filed a bug report on their open-source project page. We hope that this raises awareness of the algorithm's performance and prompts a fix.

In Section 5.4.3 we described two bit error patterns, which were very different to what we recorded in all other experiments we ran. Further investigation is required to understand what creates these patterns and whether they exhibit different Hamming distances than described by Schmidt et al. [20] and Hermans et al. [12].

Further work is required to validate our proposals regarding temperature dependent RS coding overhead using simulation and real experiments. Even though our simulation was accurate enough for our conclusions, they were based on four real traces, which certainly do not cover all possible link qualities that can be expected in real world deployments.

Bibliography

[1] Ahn, J.-S., Hong, S.-W., and Heidemann, J. An adaptive FEC code control algorithm for mobile wireless sensor networks. *Communications and Networks, Journal of 7*, 4 (2005), 489–498.

[2] Baccour, N., Koubâa, A., Mottola, L., Zúñiga, M. A., Youssef, H., Boano, C. A., and Alves, M. Radio Link Quality Estimation in Wireless Sensor Networks: A Survey. *ACM Transactions on Sensor Networks 8*, 4 (Sept. 2012), 34:1–34:33.

[3] Bannister, K., Giorgetti, G., and Gupta, S. K. Wireless sensor networking for hot applications: Effects of temperature on signal strength, data collection and localization. In *Proceedings of the 5th Workshop on Embedded Networked Sensors* (New York, NY, USA, 2008), HotEmNets '08, ACM.

[4] Boano, C., He, Z., Li, Y., Voigt, T., Zúñiga, M., and Willig, A. Controllable radio interference for experimental and testing purposes in Wireless Sensor Networks. In *IEEE 34th Conference on Local Computer Networks.* (2009), pp. 865–872.

[5] Boano, C., Tsiftes, N., Voigt, T., Brown, J., and Roedig, U. The Impact of Temperature on Outdoor Industrial Sensornet Applications. *IEEE Transactions on Industrial Informatics 6*, 3 (Aug 2010), 451–459.

[6] Boano, C. A., Wennerström, H., Zúñiga, M. A., Brown, J., Keppitiyagama, C., Oppermann, F. J., Roedig, U., Nordén, L.-Å., Voigt, T., and Römer, K. Hot Packets: A Systematic Evaluation of the Effect of Temperature on Low Power Wireless Transceivers. In *Proceedings of the* 5^{th} *Extreme Conference on Communication (ExtremeCom)* (Aug 2013), pp. 7–12.

[7] Boano, C. A., Zúñiga, M., Brown, J., Roedig, U., Keppitiyagama, C., and Römer, K. TempLab: A Testbed Infrastructure to Study the Impact of Temperature on Wireless Sensor Networks. In *Proceedings of the 13th International Symposium on Information Processing in Sensor Networks* (Piscataway, NJ, USA, 2014), IPSN '14, IEEE Press, pp. 95–106.

[8] Busse, M., Haenselmann, T., and Effelsberg, W. *The Impact of Forward Error Correction on Wireless Sensor Network Performance.* Universität Mannheim/Institut für Informatik, 2006.

[9] Contiki OS Homepage. `http://www.contiki-os.org`.

[10] RWTH FabLab. `http://www.hci.rwth-aachen.de/fablab`.

[11] Hermans, F., Rensfelt, O., Voigt, T., Ngai, E., Norden, L.-A., and Gunningberg, P. SoNIC: Classifying Interference in 802.15.4 Sensor Networks. In *Proceedings of the 12th International Conference on Information Processing in Sensor Networks* (New York, NY, USA, 2013), IPSN '13, ACM, pp. 55–66.

[12] Hermans, F., Wennerström, H., McNamara, L., Rohner, C., and Gunningberg, P. All Is Not Lost: Understanding and Exploiting Packet Corruption in Outdoor Sensor Networks. In *Wireless Sensor Networks*. Springer, 2014, pp. 116–132.

[13] Jeong, J., and Ee, C. T. Forward error correction in sensor networks. *University of California at Berkeley* (2003).

[14] Liang, C.-J. M., Priyantha, N. B., Liu, J., and Terzis, A. Surviving Wi-fi Interference in Low Power ZigBee Networks. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems* (New York, NY, USA, 2010), SenSys '10, ACM, pp. 309–322.

[15] Liu, H., Ma, H., El Zarki, M., and Gupta, S. Error control schemes for networks: An overview. *Mobile Networks and Applications 2*, 2 (1997), 167–182.

[16] Ma, R., Xing, L., Jin, T., and Song, T. A Data Transmission Mechanism for Survivable Sensor Networks. In *IEEE International Conference on Networking, Architecture, and Storage.* (2009), pp. 9–15.

[17] Polastre, J., Szewczyk, R., and Culler, D. Telos: Enabling Ultra-low Power Wireless Research. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks* (Piscataway, NJ, USA, 2005), IPSN '05, IEEE Press.

[18] Sartipi, M., and Fekri, F. Source and channel coding in wireless sensor networks using LDPC codes. In *First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks* (2004), IEEE, pp. 309–316.

[19] Schmidt, D., Berning, M., and Wehn, N. Error correction in single-hop wireless sensor networks: a case study. In *Proceedings of the Conference on Design, Automation and Test in Europe* (2009), European Design and Automation Association, pp. 1296–1301.

[20] Schmidt, F., Ceriotti, M., and Wehrle, K. Bit Error Distribution and Mutation Patterns of Corrupted Packets in Low-power Wireless Networks. In *Proceedings of the 8th ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization* (New York, NY, USA, 2013), WiNTECH '13, ACM, pp. 49–56.

[21] Texas Instruments. CC2520 datasheet, revision SWRS068. `http://www.ti.com.cn/cn/lit/ds/symlink/cc2520.pdf`, 2007.

[22] Texas Instruments. CC2420 datasheet, revision SWRS041c. `http://www.ti.com.cn/cn/lit/ds/symlink/cc2420.pdf`, 2013.

[23] Tian, Z., Yuan, D., and Liang, Q. Energy efficiency analysis of error control schemes in wireless sensor networks. In *International Wireless Communications and Mobile Computing Conference.* (2008), IWCMC'08, IEEE, pp. 401–405.

[24] TinyOS Official Source Tree. `http://www.tinyos.net`.

[25] TUD, M. A. Z., UzL, C. A. B., Brown, J., Keppitiyagama, C., UzL, F. J. O., Alcock, P., Tsiftes, N., Roedig, U., UzL, K. R., Voigt, T., et al. D-1.1 Report on Environmental and Platform Models.

[26] Wennerstrom, H., Hermans, F., Rensfelt, O., Rohner, C., and Norden, L.-A. A long-term study of correlations between meteorological conditions and 802.15.4 link performance. In *Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks* (2013), SECON'13, pp. 221–229.

[27] xpcc microcontroller framework. `http://www.xpcc.io`.

A

Appendix

A.1 List of Abbreviations

ARQ	Automatic Repeat Query
ATX	Advanced Technology eXtended
BCH	Bose-Chaudhuri-Hocquenghem
BER	Bit Error Rate
DCO	Digitally Controlled Oscillator
ECC	Error-Correction Code
FCS	Frame Check Sequence
FEC	Forward Error Correction
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
ISP	In-System Programming
LDPC	Low-Density Parity-Check
LNA	Low Noise Amplifier
LQI	Link Quality Indication
LSB	Least Significant Bit
MOSFET	Metal–Oxide–Semiconductor Field-Effect Transistor
MPDU	MAC Protocol Data Unit
MSB	Most Significant Bit
MSK	Minimum Shift Keying
OQPSK	Offset QPSK
PER	Packet Error Rate
PID	Proportional-Integral-Derivative
PPDU	Physical Protocol Data Unit
PRR	Packet Reception Rate
PS	Polystyrene
QPSK	Quadrature Phase-Shift Keying
RS	Reed-Solomon
RSSI	Received Signal Strength Indication

SNR	Signal-to-Noise Ratio
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver Transmitter
WSN	Wireless Sensor Network

A.2 Experiment Data

(a) Average Packet Reception Rate in %

Receiver	Sender 0	Sender 1	Sender 2	Sender 3
4	3.2	96.9	5.5	100.0
5	81.9	87.2	0.4	99.2
6	98.1	75.2	89.9	0.1
7	96.2	2.0	57.3	44.0

(b) Average Error Free Packets Reception Rate in %

Receiver	Sender 0	Sender 1	Sender 2	Sender 3
4	0.0	69.5	2.7	100.0
5	7.1	10.8	0.0	88.3
6	83.8	2.1	22.1	0.0
7	96.0	0.1	9.8	1.4

(c) Average LQI values

Receiver	Sender 0	Sender 1	Sender 2	Sender 3
4	50	79	77	101
5	71	72	49	87
6	85	67	73	35
7	106	44	65	61

Table A.1 Complete table of link qualifiers from the experiment described in Subsection 5.4.1. Good links (PRR $>$ 90%) are marked green, bad links (PRR $<$ 10%) red. Notice the distinction between PRR in Table (a) and *error-free* PRR in Table (b) in comparison to LQI in Table (c).