Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für
Technische Informatik und
Kommunikationsnetze

Truth be told: Benchmarking BLE and IEEE 802.15.4

Master Thesis

Anna-Brit Schaper
schapera@student.ethz.ch

Computer Engineering and Networks Laboratory
Department of Information Technology and Electrical Engineering
ETH Zürich

Supervisors:
Romain Jacob
Reto Da Forno
Prof. Dr. Lothar Thiele

November 1, 2019

Acknowledgements

I would like to thank Romain Jacob, Reto da Forno and Andreas Biri for the excellent supervision. This project would not have been possible without your valuable input and support. Furthermore, I would like to express my gratitude to Marco Zahner who went out of his way to provide us with access to the anechoic chamber. I also thank Beshr Al Nahas, Carsten Herrmann and Marco Zimmerling for the insights into their research. Finally, I would like to thank the Computer Engineering Group around Prof. Dr. Lothar Thiele for providing the resources I needed to conduct this project.

Abstract

A large body of recent flooding-based communication protocol proposals for wireless sensor networks exploit concurrent transmissions (CT). Many existing protocols are designed to run on the IEEE 802.15.4 physical layer. Thus, there is extensive research on the conditions required for successfully receiving CT over IEEE 802.15.4. While IEEE 802.15.4 is popular in the research community, Bluetooth Low Energy (BLE) has experienced a much wider commercial adoption. BLE's ubiquity makes its physical layers an attractive potential alternative for CT-based protocols.

In this project, we performed an experimental investigation of the conditions required for the successful reception of CT over IEEE 802.15.4 and the five physical layer options supported by BLE. We focused on the impact of the choice of physical layer protocol, differences in packet contents and the time and power offsets between the transmissions. Our experiments were performed in an anechoic chamber using nRF52840 dongles. We tested a wide range of parameter combinations. To make our results more accessible, we created an interactive visualisation of our data.

Contents

Abbreviations

AD advertising data

AHB Advanced High-performance Bus

API application programming interface

BER bit error rate

BLE Bluetooth Low Energy

BR/EDR Basic Rate/Enhanced Data Rate

CI coding indicator

CPFSK continuous-phase frequency-shift keying

CPU central processing unit

COTS commercial off-the-shelf

CRC cyclic redundancy check

CT concurrent transmissions

DMA direct memory access

DSSS direct-sequence spread-spectrum

EEP event end point

FEC forward error correction

FM frequency modulation

FHSS frequency-hopping spread-spectrum

GFSK Gaussian-filtered frequency-shift keying

GPIO general purpose input/output

GPIOTE GPIO tasks and events

IEEE Institute of Electrical and Electronics Engineers

ISM industrial, scientific and medical

IR interrupt

ISR interrupt service routine

LE Low Energy

LED light-emitting diode

LR-WPAN low-rate wireless personal area network

MAC medium access control

MSK minimum shift keying

O-QPSK offset quadrature phase-shift keying

OS operating system

PCB printed circuit board

PDU protocol data unit

PER packet error rate

PHR PHY header

PHY physical layer

PN pseudo-noise

PPDU PHY protocol data unit

PPI programmable peripheral interconnect

PRR packet reception ratio

PSDU PHY service data unit

QPSK quadrature phase-shift keying

RAM random-access memory

RSSI received signal strength indicator

SDK software development kit

SFD start-of-frame delimiter

SIG special interest group

SHR synchronisation header

SIR signal-to-interference ratio

SoC system-on-chip

ST synchronous transmissions

TEP task end point

TERM termination field

UART universal asynchronous receiver/ transmitter

UARTE universal asynchronous receiver/ transmitter with EasyDMA

USB Universal Serial Bus

UUID universally unique identifier

WPAN wireless personal area network

WSN wireless sensor network

Chapter 1

Introduction

In recent years, small, networked embedded systems have permeated our society. From smart homes to industrial automation, many aspects of our lives depend on reliable, low-power short-range wireless communication. Bluetooth Low Energy (BLE) is one of the most pervasive communication technologies used for this purpose. It is implemented on most modern smartphones and many other peripherals. IEEE 802.15.4, which has found widespread use in the wireless sensor network (WSN) research community over the years, presents an alternative. Recently, protocols based on the IEEE 802.15.4 standard have also seen an increase in popularity in commercial applications, particularly in the context of home automation.

The goal of this project was to investigate the performance of both BLE and IEEE 802.15.4 for one particular communication scenario: concurrent transmissions (CT). CT describe a situation in which multiple transmitters within range of the same receiver transmit at the same time. Traditional wireless routing protocols go through great lengths to avoid this condition, as it can result in packet loss caused by destructive interference.

However, since the publication of Glossy [1], the WSN research community has moved away from this approach to protocol design. Instead of expending resources on complex schemes to avoid CT, Glossy and many protocols since have focused on creating conditions under which packets can be received despite being transmitted concurrently.

In the project summarized by this report, we examined experimentally how the required conditions for a successful reception of CT vary for different choices of physical layer protocols, namely the physical layers of BLE and IEEE 802.15.4.

In Chapter 2, we outline some properties of these physical layers that are relevant to CT. On top of that, we provide some background on existing research into the conditions for successfully received CT. The methodology we used for our experiments is described in Chapter 3. To make our results as applicable to real-world scenarios as possible, we used small, low-power, low-cost commercial off-the-shelf (COTS) transceivers which could easily be embedded into many commercial wireless peripherals or deployed in a WSN testbed. Chapter 4 sum-

marises our results. In Chapter 5, we draw some conclusions from our work and outline potential areas of interest for future research.

Chapter 2

Background

In this project, we investigated the conditions for successfully receiving CT for different physical-layer protocols. The first part of this chapters provides a brief summary of the properties of these protocols that were relevant to our experiments. Afterwards, we outline some factors that are known to affect CT and relate them to the protocols.

2.1 Communication Modes

In this section, we introduce the IEEE 802.15.4 and BLE standards and highlight the features of the protocols we used in our implementation and investigation of CT.

2.1.1 IEEE 802.15.4

IEEE 802.15.4 is a wireless communication standard for low-rate wireless personal area networks (LR-WPANs). It targets the low-power and low-throughput operation, which makes it a popular choice for use cases like WSNs and home automation. The standard's scope is limited to the physical and medium access control (MAC) layers. Zigbee [2] and Thread [3] are popular higher level protocols running on top of IEEE 802.15.4.
Besides the cyclic redundancy check (CRC), only physical layer features are required for the implementation of CT over IEEE 802.15.4. The IEEE 802.15.4 standard [4] defines multiple different physical layers (PHYs) for a variety of frequency bands. In this project, we only use the offset quadrature phase-shift keying (O-QPSK) PHY in the 2450 MHz band, which is part of the globally-available 2.4 GHz ISM band. For ease of readability, we will refer to this PHY as the IEEE 802.15.4 PHY or simply as IEEE 802.15.4. In the following we highlight some properties of the PHY. A selection of those properties is summarized in Table 2.1.

PHY		IEEE 802.15.4 2450 MHz O-QPSK
Coding		DSSS
Bitrate	[bit/s]	250 K
Bit Period	[µs]	4
Symbol Rate	[symbol/s]	62.5 K
Symbol Period	[µs]	16
Chip Rate	[chip/s]	2 M
Chip Period	[µs]	0.5
$S/I_{\text{co-channel}}$	[dB]	not specified
Sensitivity	[dB]	-85
Packet Format		Fig. 2.1

Table 2.1: Summary of some properties of the IEEE 802.15.4 O-QPSK PHY in the 2450 MHz band. $S/I_{\text{co-channel}}$ refers to the SIR under co-channel interference required for low PERs. *Sensitivity* is the minimum input power level at the receiver for which PER$<$ 10 % (without interference).

Modulation IEEE 802.15.4 [4] uses direct-sequence spread-spectrum (DSSS) with a spreading factor of 8 to improve reliability (see Appendix A.3.1). Its spreading technique maps 4-bit symbols to nearly-orthogonal 32-chip pseudo-noise (PN) sequences. The chips are then modulated onto a carrier using O-QPSK modulation with half-sine pulse shaping (see Appendix A.2) and a chip rate of $1/T_{ch} = 2\,\text{Mchip/s}$.

Carrier Frequencies IEEE 802.15.4 [4] offers 16 channels, numbered 11 to 26, in the 2450 MHz band. The carrier frequency corresponding to channel k is given by

$$f_c = 2405\,\text{MHz} + 5\,\text{MHz} \cdot (k\text{–}11)\,, \qquad k \in [11, 26] \tag{2.1}$$

The standard specifies a tolerance of ±40 ppm on the carrier frequency. Thus, the frequency deviation should be less than ±100 kHz.

Bitrate The provided bitrate is 250 Kbit/s since the PHY uses a chip rate of 2 Mchip/s and a spreading factor of 8.

Packet Format The structure of a IEEE 802.15.4 packet, which is called a PHY protocol data unit (PPDU), is shown in Figure 2.1. Each packet starts with a synchronisation header (SHR), which consists of a 4 B preamble and a 1 B start-of-frame delimiter (SFD). The SHR is followed by a 1 B PHY header

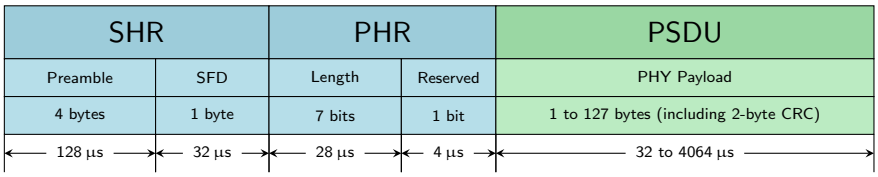


Figure 2.1: Packet structure of a PPDU in the IEEE 802.15.4 O-QPSK PHY.

(PHR). The PHR consists of a 7-bit field, which specifies the length of the PHY service data unit (PSDU) in bytes, and a reserved bit. The PSDU itself is appended behind the PHR and contains the PHY payload. Due to the size of the length field, the PSDU can have up to 127 B. If a CRC is used, the space for user data in the PSDU is reduced by 2 B, the length of the CRC.

Receiver Characteristics The IEEE 802.15.4 standard [4] specifies a receiver sensitivity of −85 dB. By the standard's definition, this means that the receiver must be able to achieve packet error rates (PERs) smaller than 10 % for packets with 20 B PSDUs and input power levels at or above its sensitivity. Note, however, that this only applies when no interference is present. The standard does not specify the amount of tolerable co-channel interference.

2.1.2 Bluetooth Low Energy (BLE)

Bluetooth is a widely-adopted wireless communication standard for wireless personal area networks (WPANs) in the 2.4 GHz ISM band that is managed by the Bluetooth SIG [5]. Bluetooth technology distinguishes between two different radio versions: *Bluetooth Basic Rate/Enhanced Data Rate (BR/EDR)* for continuous data streaming and *Bluetooth Low Energy (BLE)* for low-power, short-burst data transmissions [6]. While both versions support point-to-point communication, BLE also includes protocols for broadcast and mesh network topologies. Thus, BLE lends itself as an alternative to protocols built on top of IEEE 802.15.4 in WSN scenarios.
The BLE standard defines a full protocol stack. To investigate the potential of BLE-based CT, we are predominantly interested in the physical layer and some aspects of the link layer. In Bluetooth, these layers are collectively referred to as the *Controller* [7]. In the following, we will outline some properties of the LE Controller.

Modulation BLE uses Gaussian-filtered frequency-shift keying (GFSK) modulation (see Appendix A.1) with a time-bandwidth product of 0.5 and a modulation index between 0.45 and 0.55 [8].

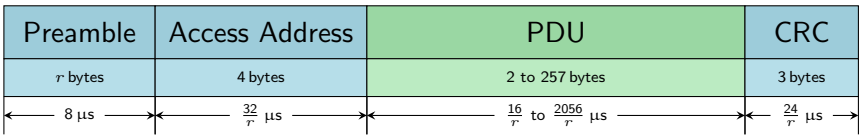
PHY		LE 1M	LE 2M	LE Coded	LE Coded
FEC Coding Scheme		none	none	S=2	S=8
Bitrate	[bit/s]	1 M	2 M	500 K	125 K
Bit Period	[µs]	1	0.5	2	8
Symbol Rate	[symbol/s]	1 M	2 M	1 M	1 M
Symbol Period	[µs]	1	0.5	1	1
$S/I_{\text{co-channel}}$	[dB]	21	21	17	12
Sensitivity	[dB]	-70	-70	-75	-82
Packet Format		Fig. 2.2a	Fig. 2.2a	Fig. 2.2b	Fig. 2.2b

Table 2.2: Summary of some properties of the four different BLE communication modes. $S/I_{\text{co-channel}}$ refers to the SIR under co-channel interference required for a BER$\leq 0.1\,\%$. *Sensitivity* is the minimum input power level at the receiver for which the PER is less than about 5 % (without interference).

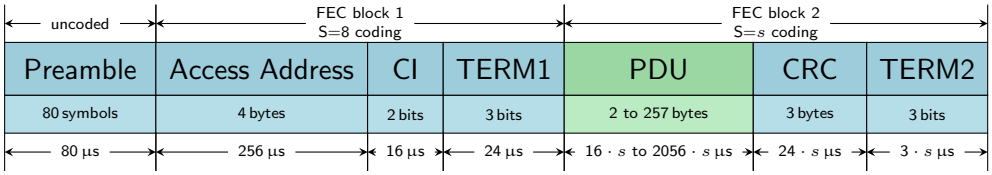
Carrier Frequencies The LE Controller [8] supports 40 different communication channels. Each communication channel is associated with a carrier frequency between 2402 MHz and 2480 MHz. Adjacent carrier frequencies are 2 MHz apart. Three of the channels, namely 37, 38 and 39, are reserved for initial advertising. These *primary advertising channels* correspond to the carrier frequencies 2402 MHz, 2426 MHz and 2480 MHz and were chosen to have minimum overlap with the commonly-used IEEE 802.11 channels 1, 6 and 11 [7]. Standard-compliant BLE systems must keep the carrier frequency drift during any packet below 50 kHz and limit the total frequency deviation (including both the initial offset and drift) to ±150 kHz.

PHY Choices Since the introduction of Bluetooth 5 [8], the LE Controller has offered a choice of physical layer implementations: the classic LE 1M PHY, which is also supported by Bluetooth 4, the LE 2M PHY, which offers a higher data rate, and the LE Coded PHY, which uses error-correcting codes to prolong the communication range [9]. As the LE Coded PHY supports two different coding schemes (S=2 forward error correction (FEC) coding and S=8 FEC coding), we need to distinguish between four different BLE physical layer communication modes. Table 2.2 shows an overview of the four modes.

Bitrate and Symbol Rate Each of the BLE modes has a unique bitrate. Thus, in the following, we will sometimes refer to a mode by its bitrate, i.e. we may call the LE 1M PHY the "1 Mbit mode", the LE 2M PHY the "2 Mbit mode", the LE Coded PHY with S=2 coding the "500 Kbit mode" and the LE Coded PHY with S=8 coding the "125 Kbit mode".
The LE 1M PHY and the LE Coded PHYs all have the same symbol rate of



(a) Packet format for the BLE uncoded modes. r is based on the bitrate of the mode; $r = 1$ for the LE 1M PHY and $r = 2$ for the LE 2M PHY.



(b) Packet format for the LE Coded PHY modes. s refers to the coding scheme employed by the mode, i.e. $s = 2$ for the 500 Kbit mode and $s = 8$ for the 125 Kbit mode.

Figure 2.2: Structure of BLE packets.

1 Msymbol/s. While the LE 1M PHY's bitrate is equal to its symbol rate, the LE Coded PHYs have lower bitrates because one bit is mapped to multiple symbols in the coding process (2 for the S=2 coding scheme and 8 for the S=8 coding scheme). To achieve its higher bitrate, the LE 2M PHY uses a symbol rate of 2 Msymbol/s.

Packet Format The BLE PHYs have slightly different packet formats, as illustrated by Figure 2.2.
While all BLE packets start with a preamble, the length of this preamble differs between the modes. For both uncoded modes, the preamble takes 8 µs to transmit, i.e. it consists of 1 B in the LE 1M PHY and of 2 B in the LE 2M PHY. The coded modes use a longer preamble; it is 80 symbols long, not coded and takes 80 µs to transmit.
In all modes, the preamble is followed by a 32-bit access address. In the uncoded modes, the access address is transmitted at the bitrate associated with the PHY. Both coded modes, on the other hand, use the S=8 FEC coding scheme for the access address. The same applies to the coding indicator (CI) field and TERM1 field, which are transmitted after the access address in the coded modes. Together, the access address, CI and TERM1 field form FEC block 1. Due to its S=8 coding, FEC block 1 is transmitted at a bitrate of 125 Kbit/s in both the 500 Kbit and 125 Kbit mode.
The actual data is transmitted in the protocol data unit (PDU), which can contain up to 257 B. It follows the access address in uncoded packets and FEC block 1 in coded packets. For error detection, a 3 B CRC is added to the packet after the PDU. Uncoded packets end with the CRC. In the coded modes, an additional termination field is appended to the packet. This TERM2 field forms FEC block 2 along with the PDU and CRC. FEC block 2 uses the

FEC coding associated with the communication mode, i.e. S=2 coding in the 500 Kbit mode and S=8 coding in the 125 Kbit mode.

Receiver Characteristics The Bluetooth 5 Core Specification [8] makes some demands on receiver performance. For instance, it specifies the receiver sensitivities listed in Table 2.2 as the receiver input levels above which very low bit error rates (BERs) are achieved. The specific BER limits depend on the payload length, since the same BER with longer payloads will result in more packet errors. Based on the BER limits specified by the standard, the PERs should be below 5 % for input levels above the sensitivity.
An interesting characteristic for CT is the *signal-to-interference ratio (SIR)* for co-channel interference, which we call $S/I_{\text{co-channel}}$. The BLE standard specifies that the BER must be at most 0.1 % for an $S/I_{\text{co-channel}}$ greater than the one listed in Table 2.2 for the respective PHY. Note that the coded modes require a lower ratio than the uncoded modes.

2.2 Concurrent Transmissions on the Physical Layer

Concurrent transmissions (CT) describe a scenario where multiple transmitters in range of the same receiver simultaneously transmit packets. In CT, a successful reception occurs when the receiver is able to correctly decode any one of the transmitted packets. Ever since the publication of Glossy [1], many protocols have been proposed that make use of CT to provide efficient multi-hop broadcast, e.g. [10–17]. These so called *synchronous transmissions (ST)* protocols have mostly been built atop the IEEE 802.15.4 physical layer. As a result, existing research on the conditions for successful CT predominantly focuses on the IEEE 802.15.4 physical layer [1, 18–22]. However, recent work [23] has initiated similar research on CT over the BLE physical layer.
Both IEEE 802.15.4 and BLE modulation use angle modulation (see Appendix A.1 and Appendix A.2). Thus, the signal arriving at the receiver from a transmitter i can be written as

$$s_{rx,i}(t) = A_{rx,i}(t)\cos\left(2\pi f_c t + \theta_{rx,i}(t)\right) \tag{2.2}$$

where f_c is the carrier frequency, $A_{rx,i}(t)$ is the signal's amplitude and $\theta_{rx,i}(t)$ is its phase. When N transmitters are transmitting concurrently, the receiver will receive a superposition $S_{rx}(t)$ of the signals originating from the different transmitters. Using the harmonic addition theorem [24], we can express the received signal as follows:

$$S_{rx}(t) = \sum_{i=1}^{N} s_{rx,i}(t) = A_{rx}(t)\cos\left(2\pi f_c t + \Theta_{rx}(t)\right) \tag{2.3}$$

where

$$A_{rx}^2(t) = \sum_{i=1}^{N} A_{rx,i}^2(t) + 2\sum_{i=1}^{N}\sum_{j>i}^{N} A_{rx,i}(t) A_{rx,j}(t) \cos\left(\theta_{rx,i}(t) - \theta_{rx,j}(t)\right) \quad (2.4)$$

$$\Theta_{rx}(t) = \arctan \frac{\sum_{i=0}^{N} A_{rx,i}(t) \sin\left(\theta_{rx,i}(t)\right)}{\sum_{i=0}^{N} A_{rx,i}(t) \cos\left(\theta_{rx,i}(t)\right)} \quad (2.5)$$

For the successful reception of CT, the receiver needs to be able to recover one of the transmitted signals $s_{tx,i}(t)$ from $S_{rx}(t)$. Let C be the transmitter whose signal is to be received correctly. The signal transmitted by C is given by

$$s_{tx,C}(t) = A_{tx,C} \cos\left(2\pi f_c t + \theta_{tx,C}(t)\right) \quad (2.6)$$

Determining the precise conditions under which a receiver can recover $s_{tx,i}(t)$ from $S_{rx}(t)$ is a complex task. Ultimately, they depend on the specifics of the modulation of the transmitted signal and the decoding techniques used by the receiver. Since a detailed analytical discussion of all the conditions for successful CT is beyond the scope of this project, we will confine ourselves to outlining some general factors that are known to play a role.

Power Delta Equation (2.4) and Equation (2.5) show that the amplitude and phase of the received signal $S_{rx}(t)$ will tend towards the amplitude and phase of the signal with the highest amplitude, which is typically (barring effects introduced by the environment and physical setup) the signal transmitted at the highest power. In fact, due to the way frequency modulation (FM) receivers are designed, if $s_{rx,C}(t)$ is sufficiently stronger than the interfering signals

$$S_{rx,I}(t) = \sum_{\substack{i=1 \\ i \neq C}}^{N} s_{rx,i}(t) \quad (2.7)$$

the receiver will lock onto $s_{rx,C}(t)$ and the interfering signals will be suppressed [25, 26]. This phenomenon is referred to as the *capture effect* or, to distinguish it from delay capture, as *power capture*. It must be noted that the receiver can only capture the strongest signal if it arrives before or during the preamble of the first of the weaker signals [11, 17].

For IEEE 802.15.4, simulations have shown that a power delta between the signal and interference of as little as 2 dB can be sufficient to support high reception rates through the capture effect [19]. However, experimental results have indicated that a 3 dB difference may be required in real systems [21, 22, 27].

According to Al Nahas *et al.* [23], successful reception of CT in the BLE physical layer is possible with a power delta of 8 dB. For the coded BLE 125 Kbit mode, his results indicates that the required difference is even lower; high reception rates were reached at a 4 dB power delta.

Packet Contents Consider the idealised case where all transmitted packets arrive at the receiver with exactly the same phase offset and amplitude. If the packets all have the same contents, constructive interference will produce an $S_{rx}(t)$ that has the same phase as the $s_{rx,i}(t)$ but a higher amplitude. Note that this would be equivalent to receiving a single signal from a stronger transmitter. Clearly, these are ideal conditions for recovering the packet contents. If, on the other hand, the packet contents differ, some destructive interference will occur. How detrimental this is to a successful reception depends on the exact conditions. Power differences caused by other signals interfering constructively and coding may help.

Time Delta Synchronizing multiple transmitters is a difficult task. In real WSNs, transmitting packets from multiple transmitters at precisely the same time is typically not possible. Thus, there will be some time offset τ_{ij} between the signals from transmitters i and j. In general, the phases of the two signals differ, regardless of whether the packet content is the same or not. If the packet from j is delayed by τ_{ij} relative to the (same) packet from i, the phase of j will be

$$\theta_{tx,j}(t) = \theta_{tx,i}(t - \tau_{ij}) \tag{2.8}$$

Intuitively, we would assume that if τ_{ij} is a symbol period or more, the situation would be similar to the case where different packets are transmitted. Without any correlation between symbols introduced by coding and with no power differences that would push the overall phase towards the phase of either i or j, we would not expect to receive either packet.
On the other hand, if the delay is very small, we would still expect the demodulation output at the receiver to fall into the correct decision region. In this case, successful CT would be possible.
For IEEE 802.15.4 modulation, simulations performed by Wilhelm *et al.* [19] and Ferrari *et al.* [1] suggest that any delay less than the symbol period can be tolerated, provided there are no other phase offsets between the signals. On the other hand, Escobar-Molero [28] claims that the maximum acceptable time delta is lower in bandwidth-limited channels when pulse shaping is employed. He suggests a threshold of half a symbol period.
In practical applications, additional phase offsets caused by the environment or carrier frequency differences cannot typically be eliminated. Thus, some authors propose using more conservative bounds on the maximum tolerable time delta. Wilhelm *et al.* [19] recommend limiting τ_{ij} to half a symbol period. Rao *et al.* [18] keep the threshold of one symbol period, but introduce an additional condition that limits $\Theta_{rx}(t)$ to $\pi/4$.
We are not aware of any existing analytical work on tolerable time delta values for CT over the BLE physical layer. However, experimental work [23] indicates that successful CT of signals with relative time offsets are only likely to reliably occur in the coded BLE 125 Kbit mode and with delays of at most a quarter of

a symbol period. For the other BLE modes, the packet reception ratio (PRR) is 75 % or lower for offsets of less than a quarter of a symbol period and drops even further for larger delays.

Coding Wilhelm *et al.* [19] argue that coding can have a large impact on the performance of CT. They suggest that, for certain time offsets between signals, the redundancy introduced by using a *DSSS* technique (see Appendix A.3.1) in IEEE 802.15.4 can aid successful reception of both identical and differing packets, particularly when symbol decisions are made on the raw demodulator output (soft decision decoding) as opposed to a binarised output (hard decision decoding).
Furthermore, *delay capture* can enable the reception of the first received packet if interfering DSSS packets with the same coding arrive with a certain delay, provided the signal strength of the interference relative to that of the first packet is smaller than the receiver's interference rejection margin [19, 29].
We are not aware of analytical research on the effect of coding on CT in the BLE physical layer. However, recent experimental work [23] has shown that the coding of the BLE 125 Kbit mode in particular may improve the reliability of CT.

Carrier Frequency Delta So far, we have assumed the carrier frequency to be the same for all signals. However, in practice, the carrier frequencies are generated by different oscillators, each of which will be slightly inaccurate. Let $f_{c,i}$ be the carrier frequency of transmitter i. If $f_{c,i} \neq f_c$, an additional phase offset of $2\pi\left(f_{c,i} - f_c\right)t$ is introduced to the signal transmitted by i. Another transmitter j will have a different carrier frequency $f_{c,j}$, and thus a different carrier phase offset. Unlike the phase differences introduced by delayed transmissions, the differences between carrier phase offsets are not constant over time:

$$2\pi\left(f_{c,i} - f_c\right)t - 2\pi\left(f_{c,j} - f_c\right)t = 2\pi\left(f_{c,i} - f_{c,j}\right)t \tag{2.9}$$

Due to this varying phase offset, the envelope of the superposition of the signals transmitted by i and j will have a cosine shape with a frequency determined by the difference between $f_{c,i}$ and $f_{c,j}$ [25]. Also, phase jumps of π are introduced where the envelope crosses zero [28]. This interference pattern is called the *beating effect* [23, 28].
Carrier phase offsets can be detrimental to the success of CT [19, 28]. Wilhelm *et al.* [19] argue that the offsets should ideally be kept below 0.4π. However, the carrier frequency cannot always be controlled very precisely, particularly in COTS systems.

Physical Environment An important point to note is that in real systems, the received signal $s_{rx,i}(t)$ always differs from the transmitted signal $s_{tx,i}(t)$.

Thus, even for just one transmitter, successful reception is not guaranteed. There may be noise produced by transmitters that are not part of the CT setup. Even in the absence of other sources emitting radio signals, *multipath* effects can have an influence on reception rates. As the name suggests, these effects occur when a signal reaches the receiver via a number of different paths because it is reflected off surrounding objects. Since the lengths of these paths differ, $s_{rx,i}(t)$ will be a superposition of delayed components of the original signal. The delays mean that the signal components are phase shifted relative to each other. Thus, even for a single transmitter we may have a situation not dissimilar to CT, where the receiver needs to recover a signal from a superposition of different signals with different phases.
The path length also has an effect in the case of multiple transmitters. If transmitters are positioned at different distances from the receiver, additional phase offsets are introduced [18].

Number of Transmitters In the highly idealised case of perfect constructive interference where the received signals from all transmitters are exactly equal, increasing the number of transmitters would be beneficial for successful CT, since the overall signal would be stronger. However, as we have seen, in practical systems, there is always some phase offset between the different received signals. Thus, a high transmitter density tends to add destructive interference and is generally harmful to CT [28].

In this project, we predominantly focus on the impact of the power delta, packet contents and time delta on the performance of CT. Our fourth parameter is the physical-layer communication mode, which is either IEEE 802.15.4 or one of the four BLE modes. The choice of communication mode implicitly sets the coding. The related work we summarised in this section also shows us that the communication mode may have an influence on the power and time delta thresholds for the successful reception of CT. We investigated this relationship as part of our experiments.

Chapter 3

Methodology

Before outlining our methodology, let us reiterate the goal of this project:

Objective To experimentally determine conditions for successful concurrent transmissions (CT) in IEEE 802.15.4 and BLE in a repeatable fashion using small, low-cost, COTS devices.

Based on this objective and the preexisting research on factors influencing the success of CT we have summarised in Section 2.2, we can formulate a set of basic requirements for the setup of our experiments:

R1 The setup must facilitate the *concurrent transmission of packets from two transmitters.*

R2 There must be a receiver which listens for CT and is able to determine whether either of the concurrently-transmitted packets is *received successfully.*

R3 The transmitter and receiver firmware must allow setting either IEEE 802.15.4 or any of the four BLE modes as the physical layer *communication mode* for a transmission.

R4 It must be possible to vary and estimate the received *power difference* between the two transmitted signals.

R5 The setup must provide a way to accurately control the *time offset* between the transmissions of the concurrently-transmitted packets.

R6 The firmware must provide a choice of transmitting either *identical or different packets* from the concurrently-transmitting transmitters.

R7 The experiments must be run in a *controlled environment* with little noise.

R8 The setup must use *small, low-cost, COTS transceivers.*

Note that these requirements, if fulfilled, allow us to investigate the influence of some choices of time delta, power delta, packet contents and communication mode on the success of CT. We do not include the number of transmitters, the physical environment and the carrier frequency offset in our variables. However, with **R7**, we attempt to minimize the influence the effect of the environment and create repeatable conditions. This should make it possible to perform future research investigating the effect of changing the number of transmitters under similar conditions. Unfortunately, **R8** makes it very difficult to study the effect of carrier frequency differences between the nodes, since there is typically no way to precisely calibrate the carrier frequency offset of cheap COTS devices. We leave it up to future work to investigate the influence of the carrier frequency delta. As a result, our experiments have four variables:

V1 communication mode

V2 power delta

V3 time delta

V4 packet contents

To achieve our objective, we need to test if CT are successful for different combinations of values of the four variables, with multiple transmissions per combination. Even if we only choose a few values for each variable, the number of different combinations for all four variables can grow quite large. Thus, it would become very tedious to manually change the setup for each change in combination. Therefore, we introduce additional requirements:

R9 The transmitter and receiver firmware must be able to perform multiple concurrent transmissions in a row and automatically cycle through at least subset of all the different combinations of the settings for the communication mode, power delta, time delta and packet contents.

R10 The receiver must be able to save and output the results (successful/not successful) of multiple concurrent transmissions along with the settings under which they were obtained.

In the following sections, we will describe our methodology based on these 10 requirements. In Section 3.1, we introduce the hardware we used for our experiments and the features it provides. We then proceed give an overview of the software we programmed our hardware with to conduct the experiments (Section 3.2). In Section 3.3, we describe some measurements we carried out to gain a better understanding of the capabilities of our hardware and software setup, particularly with respect to requirement **R5**. Finally, we briefly introduce our physical setup (Section 3.4) and highlight what sort of processing we did on our collected data (Section 3.5).

3.1 Radio Platform

For our experiments, we used Nordic Semiconductor PCA10059 boards [30] as transceivers. The PCA10059 board, also called the nRF52840 Dongle, is a USB dongle carrying a Nordic Semiconductor nRF52840 system-on-chip (SoC), a printed circuit board (PCB) antenna and some peripherals. The nRF52840 [31] is an ARM Cortex-M4 based SoC for WPAN applications.
Our choice of transceiver was mainly informed by requirements **R3** and **R8**. With a size of about 1.5 cm × 4.6 cm and a cost of around 10 $ at the time of writing, the nRF52840 Dongle qualifies as a small, low-cost COTS device. Thus, it fulfils requirement **R8**. As an added benefit, most of the components required for typical WPAN applications are contained within the nRF52840 itself, which makes it relatively simple to transfer existing applications to a custom board.
The main feature of the nRF52840 is its 2.4 GHz multiprotocol radio, which supports all four BLE communication modes specified in Bluetooth 5, IEEE 802.15.4 and two proprietary radio modes. It is possible to switch between these modes at runtime. Thus, the nRF52840 provides the necessary conditions to fulfil requirement **R3**.
The comparatively large memory available on the nRF52840, namely 256 KB of random-access memory (RAM) and 1 MB of flash, provides an excellent basis for the receiver to save results from a large number of experiments, i.e. to comply with requirement **R10**. Also, it means we do not really need to concern ourselves with code size.
The nRF52840 contains a wide range of peripherals, some of which proved to be very useful for our experiment. In the following, we will provide a short introduction to these peripherals based on the Product Specification [31].

TIMER We need some way to measure time in order to fulfil requirement **R5**. The `TIMER` peripheral runs on the nRF52840's high-frequency clock source and provides timers running on a frequency of 16 MHz. The frequency can be reduced by dividing it with a prescaler value. The timers can be up to 32 bits wide, i.e. they can run for $\left(2^{32}-1\right)/f_{\mathrm{TIMER}}$ before rolling over. There are registers to start, stop and reset the timer. It is possible to read the current time into a register by triggering a task or to generate an event once the timer reaches a user-defined value. The peripheral also provides so-called *shortcuts*, which are connections between tasks and events within a peripheral. In the case of the `TIMER` peripheral, these shortcuts make it possible to automatically stop and/or clear a timer without central processing unit (CPU) involvement when it has reached a user-defined value.

GPIO General purpose input/output (GPIO) pins are useful for interacting with the nRF52840 through wired connections. The nRF52840 Dongle exposes

15 pins for general use. Another four pins are connected to the LEDs on the nRF52840 Dongle. The dongle's user-configurable button and a reset button also use general purpose input/output (GPIO) pins.

GPIOTE The GPIO tasks and events (GPIOTE) peripheral allow interacting with GPIOs through tasks and events. When a GPIOTE channel is set up in event mode for a GPIO pin, the pin will act as an input. Depending on the configuration of the GPIOTE channel, events will be generated on either the rising edges, falling edges or all edges of the input signal. On the other hand, when the GPIOTE channel is set up in task mode, the associated pin is configured as an output. The pin's output can than be set, cleared or toggled by triggering the corresponding tasks on the GPIOTE channel.

PPI The programmable peripheral interconnect (PPI) allows peripherals to communicate with each other independently of the CPU through so-called PPI channels. These channels connect an event end point (EEP) in one peripheral to a task end point (TEP) in another peripheral. When the event associated with the EEP occurs in the first peripheral, the PPI system triggers the task associated with the TEP in the other peripheral.
For instance, assume the we want to set a GPIO pin once a timer reaches a certain value. In this case, we would first create a GPIOTE channel in task mode for the GPIO pin. Then, we would set up a PPI channel with the timer event as the EEP and the GPIOTE channel's `SET` task as a TEP. After completing this setup, enabling the PPI channel and starting the timer, we could proceed to do other processing in the CPU; the setting of the pin would happen completely autonomously.
Such setups connecting peripheral events with GPIOTE tasks is useful for determining the relative timing of events by recording and analysing the output on the pins. The PPI can help us achieve reliable timing without having to worry about unforeseen processing delays in the CPU. Note, however, that the signals on the PPI channels are synchronized to the 16 MHz clock. Thus, there will be a delay of up to 62.5 ns between the event occurring on the EEP and the task being triggered on the TEP. Shortcuts, which provide a service similar to the PPI but within peripherals, are not synchronized to the clock and thus do not experience such a delay.

RADIO We have already mentioned the nRF52840's multiprotocol radio. However, in order to understand how we can use it to investigate CT, a more detailed introduction to this peripheral is necessary.
Per requirement **R5**, we need to have tight control over when packets are transmitted. Transmissions typically cannot start instantaneously. In most applications, radios are put into a low-power state when they are not transmitting

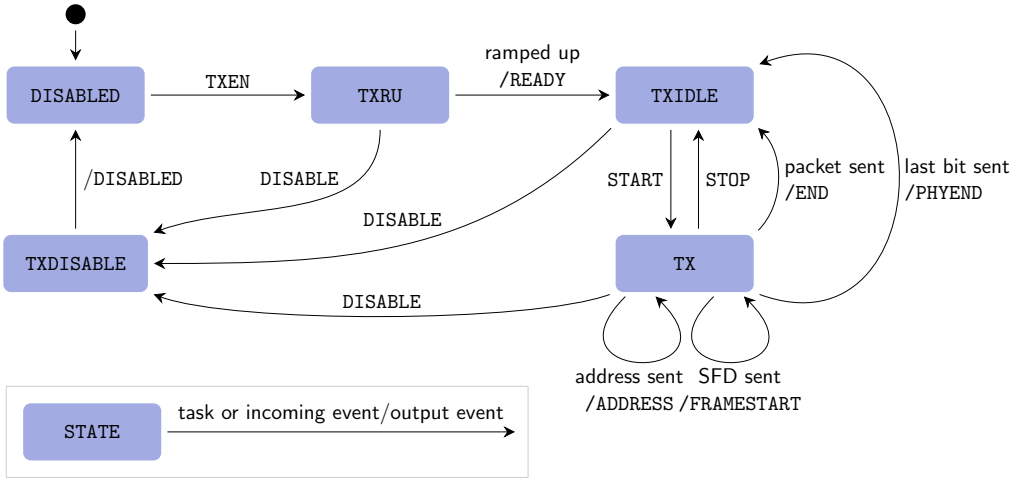


Figure 3.1: Transmitter radio states of the nRF52840. [31]

any packets to save energy. Before a transmission, the radio needs to wake up from this state, which takes some time. Figure 3.1 shows the different states and state transitions the nRF52840's radio can go through over the course of a transmission. There is an analogous system for receptions. Since we are mainly interested in the transmission timing, we will not provide details on the receiver states and refer the reader to the nRF52840 Product Specification [31].
On the nRF52840, the low-power state of the radio is called the `DISABLED` state. The radio can be enabled by triggering the `TXEN` task in software, which makes the radio move into a rampup state named `TXRU`. Once the radio is ramped up, it issues a `READY` event and transfers to the `TXIDLE` state. The manufacturer claims that the rampup time, i.e. the period between triggering the `TXEN` task in software and the `READY` event occuring, is 40 µs with a typical jitter of 0.25 µs [31].
While the radio is in `TXIDLE` state, it transmits the carrier frequency. Triggering the `START` task initiates the transmission of the packet. The `RADIO` peripheral provides a shortcut between the `READY` event and `START` task. While a packet is being transmitted, the radio is in the `TX` state. In the BLE modes, the radio generates an `ADDRESS` event once the access address has been transmitted. Since there is no such access address in IEEE 802.15.4 packets, the radio generates a `FRAMESTART` event after the SFD in IEEE 802.15.4.
For the BLE uncoded modes, an `END` event indicates that the radio has finished transmitting the packet and is moving back to the `TXIDLE` state. For modes that use coding, i.e. the BLE coded modes and IEEE 802.15.4, the `PHYEND` event fulfils the same function.
Upon triggering the `DISABLE` task the radio will go back to the `DISABLED` state via the `TXDISABLE` state. According to the manufacturer's specification [31], disabling the transmitter takes 6 µs in the BLE 1 Mbit mode, 4 µs in the BLE 2 Mbit mode and 21 µs in IEEE 802.15.4. During this time, the radio is in the `TXDISABLE`

state. Once the radio is fully disabled, the DISABLED event is generated. Shortcuts between the END or PHYEND event and the DISABLE task can be used to automatically disable the radio after a transmission.
For the fulfilment of requirement **R5**, we need the time between triggering the TXEN task and the start of the packet transmission, i.e. the transition to the TX state, to be stable. We will investigate this more closely in Section 3.3.1.
Besides the transmission timing, we are interested in the signal power. On the nRF52840, it is possible to select the output power of the transmitter in software. There are 14 different options: −40, −20, −16, −12, −8, −4, 0, 2, 3, 4, 5, 6, 7 and 8 dBm. However, the actual output power varies depending on the supply voltage and temperature. While setting the transmitter output power provides us with a way to change the input power at the receiver, the actual received power mostly depends on the physical setup and environment. Thus, the ability to set the transmitter output power is not enough to fulfil requirement **R4** on its own. We would also need a very accurate model of the path loss in our setup. Alternatively, we could measure the the received power of the signals from the two transmitters for a given transmit power and setup. The RADIO peripheral provides a received signal strength indicator (RSSI) mechanism, which allows us to get an estimate of the received power. The RSSI measurements have a resolution of 1 dB and an accuracy of ±2 dB for input levels between −90 dBm and −20 dBm [31].
In the BLE modes, one can perform an RSSI measurement for a packet by enabling the shortcut between the ADDRESS event and the RSSISTART task. After the packet is received, the measured RSSI can be accessed through the RSSISAMPLE register. Since there is no ADDRESS event in IEEE 802.15.4, we can replace the shortcut with a PPI channel between the FRAMESTART event and the RSSISTART task for this mode.
Requirement **R6** implies that we need to be able to set our own packet content. The nRF52840 allows users to transmit and receive BLE packets complying with the structures shown in Figure 2.2 and IEEE 802.15.4 packets following the format shown in Figure 2.1. Parts of the packets need to be stored in memory, namely the PDU for BLE and the PHR and PSDU (except for the CRC) for IEEE 802.15.4. The rest of the packet contents are set in radio registers. The RADIO peripheral does not enforce the specifications on the packet structure set by the higher levels of the BLE and IEEE 802.15.4 protocols. However, it does add some constraints on the format of BLE PDUs, as shown in Figure 3.2a. In the nRF52840 PDU structure, three variable-length fields are prepended to the custom payload. The size of the three fields can be set in a radio register. The length field is of particular interest. Since the BLE packet structure shown in Figure 2.2 has no such field, a priori, the receiver has no way of knowing how long a packet it is receiving is going to be. While the nRF52840 provides the option to set a static packet length, in many applications, the packet length varies over time. In these scenarios, a length field needs to be included in the packet at a position known to both the transmitter and receiver.

S0	Length	S1	Payload
≤1 byte	≤15 bits	≤15 bytes	2 to (257 - length(S0) - length(Length) - length(S1)) bytes

(a) Format of BLE packets in memory on the nRF52840. Only the PDU is stored in memory.

PHR	PSDU without CRC
1 byte	1 to 127 bytes (or 1 to 125 bytes if a CRC is used)

(b) Format of IEEE 802.15.4 packets in memory on the nRF52840. Only the PHR and PSDU (without the CRC) are stored in memory.

Figure 3.2: Structure of packets in memory on the nRF52840.

To transmit a packet, the `RADIO` peripheral needs to fetch the PDU/PSDU from memory. To achieve this, it uses a module called *EasyDMA*. The radio's EasyDMA module functions as an Advanced High-performance Bus (AHB) bus master connected to the AHB multilayer interconnect on the nRF52840. The AHB multilayer allows multiple bus masters (like the radio's EasyDMA module or the CPU) to access different slaves (RAM blocks) in parallel. Note that the each slave can only be accessed by one bus master at a time. Multiple bus masters attempting to access the same slave simultaneously are served by priority, lower priority bus masters are stalled while the highest priority bus master is using the slave. The CPU has the highest priority. Thus, while the radio is transmitting or receiving, it may be advisable to avoid large memory transfers to and from the same RAM AHB slave as the one containing the memory region reserved for the PDU/PSDU.

UARTE To fulfil requirement **R10**, we need some way to retrieve the collected results from the nRF52840. The universal asynchronous receiver/ transmitter with EasyDMA (UARTE) module provides a way to achieve this. The peripheral provides a simple UART interface that uses EasyDMA to read and write data from and to RAM. UARTE's EasyDMA module has a lower priority than both the CPU and the radio's EasyDMA module. However, this should not play a major role for our setup since the data transfer via UARTE only needs to happen once all receptions are finished.

3.2 Firmware

In this section, we introduce the firmware we used for our experiments. First, in Section 3.2.1, we describe drivers we created in order to simplify the access to the nRF52840 peripherals introduced in Section 3.1, with a particular focus on the radio. In Section 3.2.2, we proceed to give an overview of our main application

which schedules, transmits and receives the CT.

3.2.1 Radio Interface

While higher-level applications can make use of Nordic's implementations of the BLE and IEEE 802.15.4 protocol stacks, we needed more precise control of low-level radio functions. Thus, we decided to create our own radio driver tailored to our requirements. We implemented this driver as part of a partial port of Contiki [32] to the nRF52840 Dongle. Contiki is an open-source operating system (OS) for low-power sensor motes popular in the WSN community.
With this choice, we align ourselves with previous work on CT over BLE [23]. The authors of this work also implemented their firmware in Contiki and have made their code publicly available [33]. Since they also worked with the nRF52840, we were able to reuse some elements of their Contiki port, such as their clock and `rtimer` implementations. In the interest of time, we focused on porting Contiki features to the nRF52840 Dongle that were useful for our project. Besides the radio, clock and `rtimer` drivers, this included a bare-bones serial-line driver relying on the UARTE peripheral. We attempted structure our drivers in such a way that they are extendable and reusable for other purposes.
The heart of our partial Contiki port is the radio driver. Our driver implementation follows Contiki's radio application programming interface (API) [34]. In the following we will provide a high-level overview of the driver and how an application can interact with it.

Radio Settings

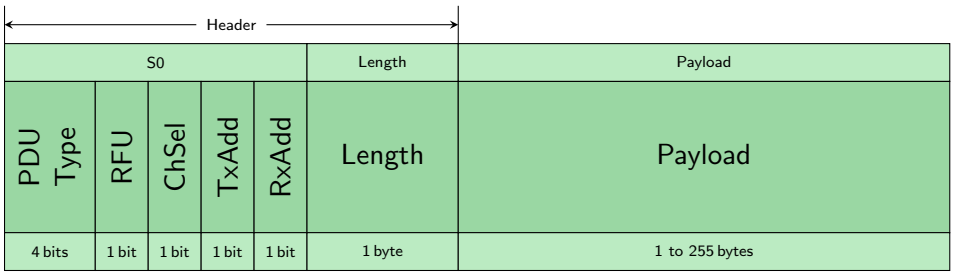
Contiki's radio API allows an application to change radio settings by passing the name of the parameter as well as its new value to the `set_value` function. Our driver implements some of the radio parameters defined by the API, including the channel and transmission power. On top of that, it defines some parameters that allow the user to change settings that are specific to the nRF52840, such as the communication mode. Consult Table 3.1 or a non-exhaustive overview of the radio parameters supported by our driver.
Internally, the driver saves the changed settings and adjusts the peripheral registers to implement the changes. Often, changing one setting may require updates to multiple registers. For instance, when a new communication mode is selected, the packet structure changes as well and the registers setting this structure need to be updated. Also, the new register values may depend on other settings. For instance, a channel number in a BLE mode will not correspond to the same carrier frequency as in IEEE 802.15.4. Thus, particularly after a mode change, the application may need to reset some parameters. If it fails to do so, the driver will use the default mode-dependent settings or, if the mode was used before, the values that were previously set in the mode.

Parameter	*Description*	*Possible Values*
`TXPOWER`	Set the transmission power.	$\{-40, -20, -16, -12, -8, -4, 0, 2, 3, 4, 5, 6, 7, 8\}$ dBm
`CHANNEL`	Set the communication channel number.	BLE: [0,39] IEEE 802.15.4: $[11, 26]$
`MODE`	Set the communication mode.	`RADIO_MODE_{` `MODE_Ieee802154_250Kbit`, `MODE_Ble_1Mbit`, `MODE_Ble_2Mbit`, `MODE_Ble_LR125Kbit`, `MODE_Ble_LR500Kbit`}
`PDU_TYPE`	Set the PDU type (BLE only).	{`ADV_NONCONN_IND`}
`ADVA`	Set the advertiser's address (BLE only).	6 B AdvA

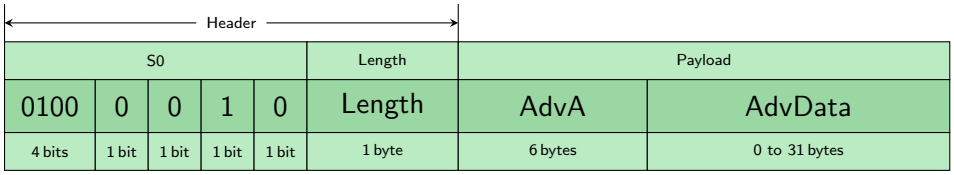
`RX_BEHAVIOUR_ON_PKT_END`	Set how the radio should behave after a packet was received.	`NRF_RADIO_{`
		`RX_RESTART_ON_END`
		`RX_DISABLE_ON_END`
		`RX_IDLE_ON_END`}

`RX_PACKET_RSSI_ENABLED`	Enable or disable RSSI measurements on received packets.	{`true`, `false`}
`LAST_RSSI`	Get RSSI measured for the last received packet.	$[0, -127]$ dB or `RSSI_NONE`

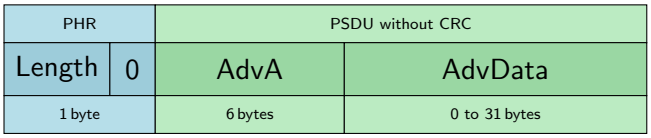
Table 3.1: Selection of radio parameters that can be set and/or accessed through the radio API. When the parameters are passed to the API, their names need to be prefixed by `RADIO_PARAM_`.



(a) General structure of BLE advertising channel PDUs.



(b) Structure of a BLE non-connectable and non-scannable undirected advertising PDU.



(c) Structure of an IEEE 802.15.4 PHR and PSDU mimicking a BLE non-connectable and non-scannable undirected advertising PDU.

Figure 3.3: Structure of advertising payloads in memory on the nRF52840.

In its current version, the driver does not have extensive safeguards against illegal settings chosen by the application. To avoid unforeseen consequences, the user should not make any changes to the settings while the radio is not in the `DISABLED` state, i.e. while it is in the process of transmitting or receiving a packet.

Payload Preparation

To transmit a packet, an application first needs to call the API `prepare` function with a payload and length. The driver will then put this information into a structure that complies with Figure 3.2 and save this structure to a transmit buffer.
To preserve some degree of higher-level standard compliance and allow comparability with previous work [23], we decided to follow the structure of *Proximity Beacon* packets, which are also called *iBeacons* [35]. Proximity Beacons are BLE advertising channel packets that are used to provide location services. All BLE advertising channel PDUs follow the structure shown in Figure 3.3a. A 2 B advertising channel PDU header is prepended to the payload. Note how this format fits into the in-memory PDU structure presented in Figure 3.2a.

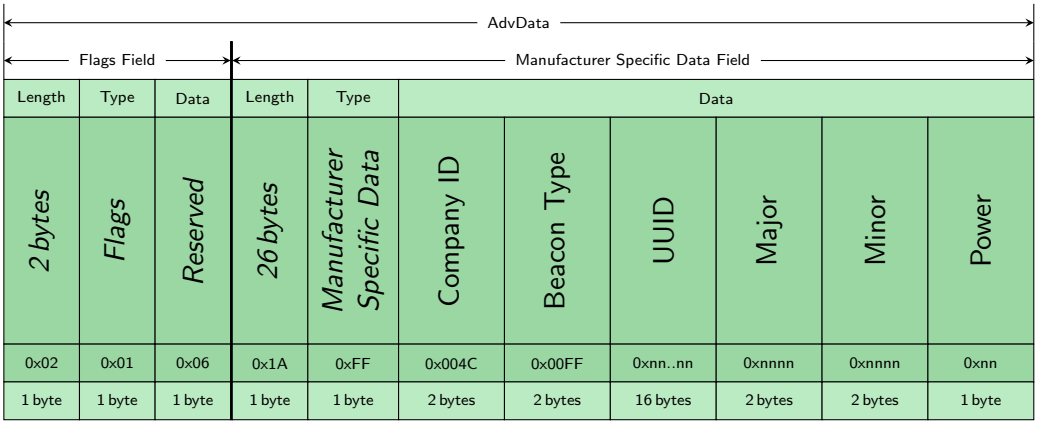


Figure 3.4: Structure of the advertising data (AdvData) block in iBeacon packets [8, 35–37].

The contents of the header and the structure of the payload depend on the PDU type of the packet. The BLE standard [8] defines 15 different advertising channel PDU types. Proximity Beacons are based on the PDU type for non-connectable and non-scannable undirected advertising events, which is also called the ADV_NONCONN_IND PDU type. ADV_NONCONN_IND PDUs follow the structure shown in Figure 3.3b. Their payload contains a 6 B advertiser's address (AdvA) field and an advertising data (AdvData) field that may be up to 31 B long.
The AdvData block consists of zero or more advertising data (AD) structures. Each AD structure begins with a length field, which outlines the size of the structure (excluding the length field itself). This is followed by the structure's data type, which is one of the numbers specified in the Bluetooth Assigned Numbers document [36]. The final part of the AD structure is the actual data. Proximity Beacon AdvData blocks consist of a flag AD structure and an AD structure for manufacturer specific data, as shown in Figure 3.4. The contents of the universally unique identifier (UUID), major, minor and power fields in the manufacturer specific data can be set by the user. Normally, the contents of these fields play specific roles in providing a location service. However, for our purposes, we can set them to whatever is required by our experiment setup.
ADV_NONCONN_IND PDUs are always transmitted using the BLE primary advertising channels. The BLE standard specifies that ADV_NONCONN_IND PDUs may only be sent over the LE 1M PHY. However, in order to get comparable results, we used the same PDU structure across all BLE modes. In fact, we even applied a similar format to our IEEE 802.15.4 packets. As shown in Figure 3.3c, we split the IEEE 802.15.4 PSDU into an AdvA and an AdvData field to mimic the ADV_NONCONN_IND payload.
To simplify sending standard-compliant BLE packets through the radio API, our driver provides parameters for the PDU type and advertiser's address that can

be set with the `set_value` function (see Table 3.1). When a BLE mode is set, the `prepare` function will automatically put the provided payload into the format specified by the PDU type. Because we are using ADV_NONCONN_IND PDUs in all our experiments, the driver currently only supports this PDU type.
Since we use a non-standard format for our IEEE 802.15.4 packets, we have not implemented an equivalent service for this communication mode in the driver. Thus, to achieve the structure shown in Figure 3.3c, the application needs to prepend the AdvA to the AdvData before passing the payload to `prepare`.
Similarly, to create an iBeacon packet, the application needs to put the AdvData into the correct format before calling `prepare`. Our radio driver provides a separate module to simplify the handling of iBeacon AdvData blocks.
In our current implementation of the driver, there is only one transmit buffer. Thus, the application must ensure it does not `prepare` a new packet while the previous one is still being transmitted. While this did not hinder our experiments, it might be a good idea to set up a more sophisticated transmit buffer structure when implementing more complex communication protocols.

Transmission

After preparing the payload, the application can call the `transmit` function to initiate its transmission. The driver will set the `PACKETPTR` to the address of the transmit buffer, enable the transmitter and start the transmission. Once the CPU has initiated the radio rampup procedure it is not involved in the transmission anymore. The transmitter is disabled automatically through a shortcut when the transmission is finished.

Reception

An application can start listening for a packet by calling the `on` function. The driver will then set the `PACKETPTR` to the address of a receive buffer and ramp up and start the receiver. Once the radio is started, the application can check whether a packet is currently being received by calling the `receiving_packet` function. The driver uses the state of the `ADDRESS`/`FRAMESTART` and `END` event registers to determine whether this is the case.
Prior to calling `on`, the application can determine what the radio should do once it has finished receiving a packet through a radio setting. The radio can either remain in the idle state, immediately return into the receiving state, or disable automatically. The first option requires no action from the nRF52840 radio, the latter two are implemented using shortcuts. To turn the radio off manually, the application can use the `off` function.
In the `pending_packet` function, the driver uses the state of the `CRCOK` event register to check whether a packet was received correctly. If there is a pending packet, the application can use the `read` function to read the payload into an

application buffer. Note that `read` should be called before starting a new reception, since currently only one receive buffer is being maintained by the driver. The contents of this buffer will be overwritten by the next received packet.

RSSI Estimation

An application can enable RSSI measurements on received packets by setting the `RADIO_PARAM_RX_PACKET_RSSI_ENABLED` with the `set_value` function. The driver will then estimate the RSSIs of received packets using the techniques outlined in Section 3.1. The application can access the RSSIs of the last received packet using the `get_value` API function. The corresponding parameter is `RADIO_PARAM_LAST_RSSI`.

3.2.2 Application

In the previous section, we introduced the partial Contiki port that formed the backbone of the firmware we used in our experiments. In this section, we provide an overview of the of the part of the firmware that actually conducts the experiments. We do this by explaining how it addresses each of the requirements laid out at the beginning of this chapter.

R1 — Two Concurrent Transmitters In our experiments, we used two nRF52840 Dongles as transmitters. Their firmware uses our partial Contiki port, which we described in Section 3.2.1. The firmware transmits packets using the `transmit` function of the radio API. Since the two transmitters are required to transmit concurrently, we need some sort of synchronisation mechanism. To keep matters as simple as possible and avoid having to deal with additional wireless transmissions, we decided to introduce another nRF52840 Dongle as a trigger: This third dongle triggers the transmissions on the two transmitters by toggling GPIOs that are connected to pins on the transmitters with jumper wires. We linked the transmitter GPIOs to GPIOTE channels in event mode and set up interrupts tied to the GPIOTE input events. Thus, each time a GPIO connected to a transmitter is toggled on the trigger dongle, an interrupt is invoked on the transmitter. In the corresponding interrupt service routine (ISR), the transmitter application clears the event and then immediately calls the `transmit` function.
After initiating the transmission, the transmitter software waits for a set time using an `rtimer`. We configured this time to be long enough for the radio to finish transmitting the packet. In the `rtimer` callback, the firmware updates its state, creates the next payload, prepares it for transmission by calling the `prepare` function and then waits for the next trigger.

The trigger dongle also uses an `rtimer` to leave enough time for the transmitters to perform all these steps before the next transmission is triggered.

R2 — Receiver The fourth and final dongle we used in our experiments adopts the role of the receiver. It must always be in receiving mode when the transmitters are transmitting. To achieve this, the receiver firmware calls the `on` radio API function multiple milliseconds before it expects the next transmission to be triggered.
The receiver can estimate of the time of the next expected transmission because it knows the statically set schedule of the trigger and the transmitters put state information into the packets. Once the receiver receives a packet, it knows at which point in the schedule it is, and thus how much time should pass between the current transmission and the next. Based on the time at which the packet with the state information was received, the receiver can determine at which point to turn on next. The receiver uses an `rtimer` to schedule the next call of the `on` function. Even if the receiver misses multiple subsequent packets, it can independently update its schedule. However, because of clock drift, the receiver does need to receive a new packet once in a while to remain synchronized. To get synchronized in the first place, as soon as the receiver firmware is started, it turns on the radio and keeps it on until it receives the first packet.
The receiver needs to be able to tell whether one of the concurrently transmitted packets was received successfully. As a first check, it calls the `pending_packet` function to determine whether the CRC was correct. If so, it examines the packet contents in more detail. Apart from the state information, the packet contents are either static across all transmissions or randomized with a seed that can be derived from the state. Thus, the receiver can construct the expected payload based on the state information received in the packet. If the constructed payload is identical to the received one, the receiver determines that one of the concurrent transmissions was received correctly.
If either the CRC check or the payload comparison fails, the receiver decides it has missed a packet. The same applies if a certain amount of time has passed after the expected time of the next transmission and no packet start is detected.

R3 — Communication Mode As explained in Section 3.2.1, the transmitter and receiver firmware can select any of the four BLE modes or IEEE 802.15.4 to be the PHY by setting the `RADIO_PARAM_MODE` parameter to the desired mode using the `set_value` radio API function.

R4 — Power Delta It is very simple to set the transmit power in the transmitter firmware; the application can set the `RADIO_PARAM_TXPOWER` parameter to the desired value using the `set_value` radio API function.
However, as mentioned in Section 3.1, setting transmit powers with a certain

offset at the two transmitters does not necessarily mean that the received signals will have the same power difference. We decided to perform RSSI measurements to get an estimate of the received power delta. Unfortunately, we cannot perform these measurements during the CT because the receiver can only measure the RSSI of the superposition of the two signals. Thus, we need to perform separate measurements.
Since we require rather static environmental conditions, it should be enough to measure the RSSIs of the two transmitters once before a batch of transmissions. Recording these measurements allows us to translate the transmit power delta associated with a set of CT to an estimated RSSI delta, i.e. an estimated received power delta, when analysing the results.
To collect the measurements, we used most of the same elements as for the CT experiments. The main difference is that the two transmitters are triggered alternately instead of concurrently. Also, the receiver needs to enable and perform RSSI measurements on received packets using the radio API as outlined in Section 3.2.1.

R5 — Time Delta We have already discussed how we can make the two transmitters transmit roughly concurrently. However, for our experiments, we needed to be able to set an exact time delta between the two transmissions. We achieved this by toggling the trigger GPIOs with an offset.
Our setup to create this offset is as follows: Instead of toggling the GPIOs in the trigger's `rtimer` callback, the application starts a 16 MHz timer in the `TIMER` peripheral. The timer is set up such that it generates an event at time t_0 and another one at time t_1 after its `START` task is triggered. t_0 and t_1 are chosen such that $t_1 - t_0$ is the desired time offset between the transmissions. Once the timer has generated the two events, it is automatically stopped and reset through a shortcut.
The timer events are connected to task mode GPIOTE channels through the PPI system. The GPIOTE channels control the trigger GPIOs. This setup should create the desired offset between the two trigger signals.
However, there are quite a few steps between the timer event on the trigger dongle and the corresponding transmission start on the transmitter dongle. On each of these steps, there might be some time jitter that could distort the actual time offset between the two packets. To get an idea of the potential magnitude of this deviation, we conducted some preliminary tests which are summarized in Section 3.3.1.

R6 — Packet Contents As explained in Section 3.2.1, we used iBeacon packets for our experiments. Thus, the only parts of the payload that could freely be set by the application were the UUID, major, minor and power fields. We used the major, minor and power as well as the last two bytes of the UUID to convey state information to the receiver.

In experiments where two identical packets were to be transmitted, we set the remainder of the UUID to some static value.
On the other hand, when the packets were meant to differ, we transmitted partly randomized packet contents. In this case, for each transmission, the two transmitters calculate seeds based on the current schedule state. The calculation method differs between the two transmitters such that the two seeds for a CT are always different. Based on the seed, the transmitters generate the first 14 B of the UUID from a pseudo-random generator. We chose this method instead of generating fully-random packet contents in order to simplify fulfilling requirement **R2**. Also, we decided against using static differing packet contents in order to cover a wider range of signal combinations.

R7, R8 — Environment and Hardware Some of our requirements are not related to the firmware. Our hardware requirements were addressed in Section 3.1. In Section 3.4 we explain how we tackled our requirement for a controlled environment with little noise in our physical setup.

R9 — Automation Let us address how our firmware setup handles transmitting multiple CT in a row.
We call the process of concurrently transmitting one pair of packets a *round*. Each round has a binary result: Either the transmission is successfully received, or it is not. However, one such result tells us very little about the probability of a successful reception. To produce a more meaningful output, the firmware needs to perform multiple rounds under the same conditions. The accumulation of the results from such a batch of rounds can then be translated into a PRR. We call this PRR value, along with the conditions under which it was obtained, a *sample*. The conditions for a sample consist of the chosen settings for the four variables **V1** to **V4**, i.e. the communication mode, power delta, time delta and packet contents.
To simplify collecting samples under a wide range of combinations of **V1** to **V4**,our firmware automatically changes the conditions after each sample it collects. We call the process of the firmware running through all samples for a set of desired variable combinations a *run*.
Somewhat arbitrarily, we decided to only vary the communication mode, power delta and time delta within a run. Whether the packet contents were the same or different was only changed between runs. The main reason for this approach was to reduce the runtime and results storage space required for a single run.
To ensure all dongles update their communication mode, power delta and time delta settings at the same time, they all need to be aware of how many rounds are left to in the current sample. This information is communicated using the state information sent with the packets in the shape of a round number. The total number of rounds is programmed into the firmware by passing it to the preprocessor, as are all the variable combinations the firmware needs go through.

	Values
Round Interval	35 ms +1000 ms on mode changes +100 ms on power changes
Rounds per Sample/ *RSSI Measurement Configuration*	20
Communication Modes	{BLE 1 Mbit BLE 2 Mbit BLE 500 Kbit BLE 125 Kbit IEEE 802.15.4}
Time Deltas	$\{i \quad \forall i \in [-15, 15]$, $\pm 20 \pm 5 \cdot j \quad \forall j \in [0, 6]$, $\pm 60 \pm 10 \cdot k \quad \forall k \in [0, 4]$, $\pm 120 \pm 20 \cdot l \quad \forall l \in [0, 1]\}$ tick
Transmit Power Deltas	$\{\pm(8 - i) \quad \forall i \in$ $\{-8, -4, 0, 2, 3, 4, 5, 6, 7, 8\}\}$ dB
Transmit Powers	$\{-8, -4, 0, 2, 3, 4, 5, 6, 7, 8\}$ dBm

Table 3.2: Parameter choices for our CT experiments. Note that we use the maximum `TIMER` frequency of 16 MHz and thus 1 tick = 1/16 µs = 62.5 ns.

Besides the round number, the state contains information on the current settings for the communication mode, power delta and time delta. Between transmission, the dongles update their settings based on the current state.
As mentioned in our approach to **R4**, our firmware needs to perform a set of RSSI measurements at the beginning of each run. More specifically, for each combination of communication mode, transmit power and transmitter, it needs to collect the RSSIs of multiple packets. To automate this we used a very similar approach to the one we just described for the CT. The main difference is that the state information consists of the measurement number, communication mode, transmit power and transmitter. Also, during the RSSI collection phase, a round describes the transmission of a packet from just one transmitter.
Table 3.2 shows how we set the parameters for all of our runs. We chose to do 20 rounds per configuration to get a somewhat meaningful result whilst keeping the number of rounds per run to a reasonable number. For the setup shown in Table 3.2, the firmware needs to perform 118 000 CT rounds across 5900 different parameter combinations and 2000 RSSI rounds across 100 different parameter combinations. Performing an entire run, including both the CT and RSSI rounds, takes less than 74 min.

R10 — Extracting Results Due to our physical setup (see Section 3.4), we could not read out results until after a run had completed. Thus, the receiver needed to store all the results obtained during a run. Storing all 2000 RSSI measurements requires 2000 B. If the firmware were to store the results of all 118 000 CT rounds separately, the same way as the RSSI measurements, it would require quite a lot of memory. However, we do not really care which of the rounds for a parameter combination was successful, just how many. Thus, we can introduce a counter variable for each CT parameter combination. The receiver increments this counter when it receives a packet under the conditions specified by the parameter combination. Since the firmware performs 20 rounds per sample, the maximum counter value is 20. This fits comfortably into a single byte. Thus, only 5900 B are required to store all CT results. In total, this results in a memory requirement of 7900 B, which the 256 KB RAM of the nRF52840 should easily be able to accommodate.
However, these 7900 B only store the results themselves and none of the parameter combinations associated with them. As it turns out, this is not a problem, provided the results are stored in the order in which they were obtained. Since the receiver knows the schedule, i.e. the order in which the different parameter combinations were tested, it can recalculate the settings for a given result based on its index in the order.
In fact, this is precisely what the receiver firmware does when the results are read out: One by one, it fetches the results from memory, calculates the associated parameter combination and prints out both using the serial-line driver.
Once a run is finished, the firmware starts listening for a UART input. As soon as it receives an input followed by a line break, it starts writing the results to the serial output. We used an adapter to connect the pins assigned to the UART by the firmware to a USB port on a computer. On the computer's end, we used a simple Python script for the serial communication.

3.3 Platform Profiling

In relation to requirement **R5**, we needed to determine with how much precision we could set the time offset between the transmissions. With the methods described in Section 3.3.1, we determined that the jitter we could expect to incur on the transmission time delta in our experiments is about 124 ns.
While we did not put much weight on the carrier frequency offset, we attempted to chose transmitter dongles for our experiments that were close in frequency. This process is outlined in Section 3.3.2.

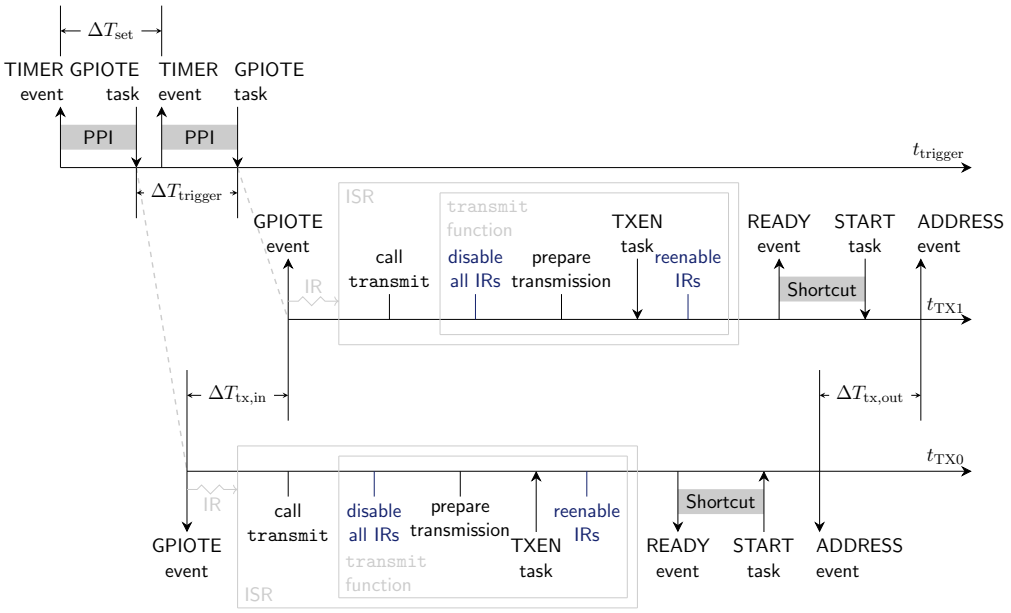


Figure 3.5: Overview of the events and processes leading up to a transmission for a CT round.

3.3.1 Transmission Timing Precision

Per requirement **R5**, the firmware needs to have precise control over the timing of transmissions. Let us assume that we can set the time delta on the trigger dongle with a high degree of accuracy. This is reasonable to assume, since our choices for the time offset ΔT_{set} are very short ($\leq 87.5\,\mu\text{s}$). Thus, there should not be much clock drift during the time delta. Unfortunately, as we have mentioned in Section 3.2.2, this does not guarantee that the actual offset of the transmissions is equally accurate.

In Figure 3.5, we provide an overview of the steps the firmware needs to take between the firing of the trigger timer and the actual transmission. Each of these steps may introduce some jitter, and the imprecisions will add up over all of them. To get an idea of the degree of imprecision we would incur during our experiments, we performed some preliminary timing measurements using Saleae Logic 8 and Logic Pro 16 logic analysers, which have a maximum sample rate of 100 MS/s and 500 MS/s respectively.

The first point at which we could investigate the jitter on the time delta was on the trigger dongle output. To get to this point, the signal needs to travel through a PPI channel from the `TIMER` peripheral to the GPIOTE peripheral and change the output level. To measure the jitter, we connected the 500 MS/s logic analyser to the trigger GPIOs. We perfomed 15 runs of 600 concurrent transmissions each with no time offset and found that for each run, all time offsets $\Delta T_{\text{trigger}}$ were within about 2 ns of each other. This corresponds to the precision of the

logic analyser.
From the outputs of the trigger dongle, the signals need to propagate across the jumper wire to the inputs at the transmitter dongles and be detected by the GPIOTE peripheral. We also attempted to get an idea of a jitter on the time offset at this point, $\Delta T_{\text{tx,in}}$. To measure this, we used the GPIOTE event as an EEP for a PPI channel whose TEP was another GPIOTE channel. During the same set of transmissions as we just described, we also connected the logic analyser to the pin associated with this other GPIOTE channel. For each run, all time offsets $\Delta T_{\text{tx,in}}$ were within about 122 ns of each other. This is quite an increase from the 2 ns range we observed at the trigger output. However, remember that PPI channels are synchronized to a 16 MHz clock. Thus, depending on the time GPIOTE event occurs, the signal may be delayed by anywhere between 0 and 62.5 ns on each transmitter. Therefore, the measured jitter may largely be an artifact of our measurement setup.
In the experiment firmware, the GPIOTE event invoked by the trigger input is not connected to a PPI channel, but generates an interrupt. In the ISR, the firmware clears the interrupt and then immediately calls the `transmit` function defined by the radio API. In the `transmit` function, the CPU performs multiple instructions to prepare the radio for transmission.
Interrupts occuring during this preparation phase are a serious danger to the timing precision. We confirmed this by intentionally generating interrupts during the preparation phase. With these interrupts, we were able to arbitrarily delay the individual transmissions. By placing an infinite loop into the disrupting ISR, we could even prevent the radio from transmitting at all. To prevent such disruptions, we created an option to disable all interrupts at the start of the transmit function and reenable them at the end of the function. Enabling this option only leaves the time between the GPIOTE event and the start of `transmit` function unprotected.
After preparing the transmission, the `transmit` function triggers the `TXEN` task. From this point on, the transmission is handled entirely by the radio. Thus, the firmware can safely reenable interrupts and exit the `transmit` function and ISR. In the meantime, the radio will ramp up and generate a `READY` event once it is done. For our experiments, we have enabled the shortcut between `READY` event and `START` task. Thus the transmission will start automatically after the rampup.
Ultimately, we are interested in the time at which the packet transmission starts. The nRF52840 provides no convenient way to measure this. However, we can just as well investigate the time offset at some other fixed point in the packets. For our measurements, we used the `ADDRESS` events (or `FRAMESTART` events in the case of IEEE 802.15.4) to measure $\Delta T_{\text{tx,out}}$. We set up a PPI channels from these events to GPIOTE outputs and connected the 500 MS/s logic analyser. We used this setup to investigate the jitter on $\Delta T_{\text{tx,out}}$ for all $\Delta T_{\text{set}} \in \{i \cdot 125\,\text{ns} |\ i \in [0, 10]\}$ and across all 5 communication modes. For each combination of mode and ΔT_{set}, we performed 3 runs of 1200 concurrent transmissions each. We found that for

Transmitter	A	B	C
Measurement 0	2405.045 MHz	2405.046 MHz	-
Measurement 1	2405.046 MHz	2405.048 MHz	2405.049 MHz
Measurement 2	-	2405.049 MHz	2405.049 MHz

Table 3.3: Actual carrier frequencies measured for a set carrier frequency of 2405 MHz on the transmitters used for the experiments.

each run, the time offsets were within 124 ns of each other. As before, quite a large part of the jitter could be caused by the PPI system, which we used for our measurements.
For future application, it is also interesting to get an idea of how much of the jitter is caused by the `transmit` function and radio rampup. To investigate this, we made our firmware generate an event immediately after the `transmit` function, which we connected to our 100 MS/s logic analyser using the PPI and GPIOTE peripherals. We then proceeded to measure the time offset between the event at the beginning of the `transmit` function and the `ADDRESS` event (or `FRAMESTART` event) on a single transmitter. For each communication mode, we performed 8 runs of about 800 transmissions each, during which we generated various types of interrupts. In each run, we found that the periods measured were all within about 10 ns of each other, which corresponds to the maximum precision the logic analyser can provide.
This leads us to believe that most of the 124 ns jitter on $\Delta T_{\text{tx,out}}$ is likely caused either by the process of registering the trigger signal input, i.e. the interrupt or our measurement set up.

3.3.2 Carrier Frequency Offset

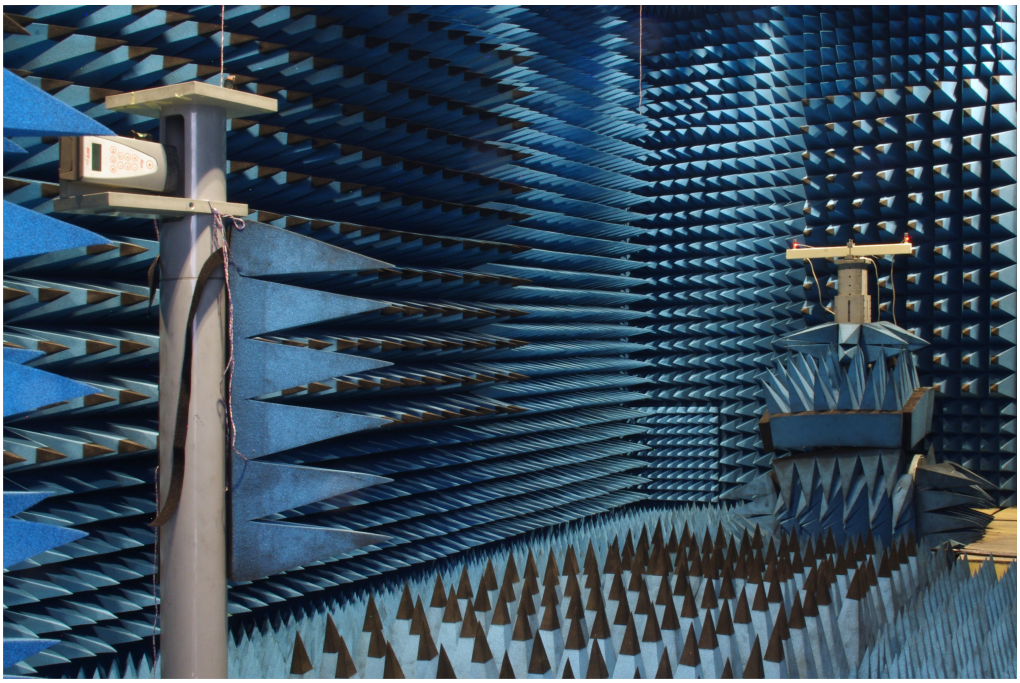
As explained at the beginning of this chapter, we did not focus on the carrier frequency offset in this project. However, to minimize its influence, we chose to use nRF52840 dongles whose frequencies were close to each other. To measure the offset, we transmitted an unmodulated carrier 2405 MHz from each dongle that was available to us. We used a software-defined radio to estimate the actual frequency that was transmitted. The closest frequencies we could identify are listed in Table 3.3. However, the frequencies tended to vary quite a lot over time. Thus, our confidence in these estimates is fairly low.
Nevertheless, based on the measurements, we chose to use two different pairs of transmitters in our experiments: A and B with an estimated offset of 1 kHz to 2 kHz as well as B and C with an estimated offset of 0 kHz to 1 kHz.

3.4 Physical Setup

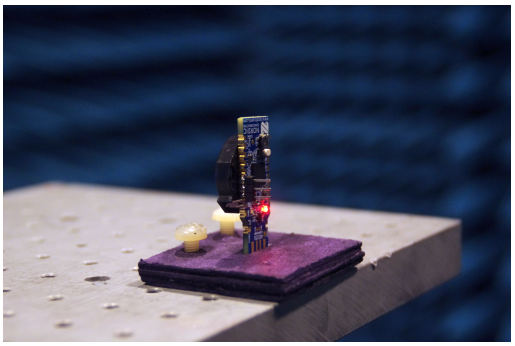
To fulfil requirement **R7**, we needed to find a controlled environment with little noise to run our experiments in. We chose an environment built for precisely this purpose, namely an anechoic chamber.
As outlined in Section 3.2.2, we required three nRF52840 dongles for our experiments: a receiver, two transmitters placed at an approximately equal distance from the receiver, and a trigger connected to the transmitters with jumper wires. Our setup is shown in Figure 3.6.
We used coin cells to power the receiver and trigger dongles. The transmitters, on the other hand, were supplied by the same power bank. We made this choice to avoid unexpected transmit power offsets caused by a difference in supply voltages. Unfortunately, many commercially available power banks require a minimum current draw to continually supply power. Since our transmitters alone did not reach this current draw, we attached an extra device drawing a higher current to the power bank.
During experiment runs, we left and closed the chamber, keeping the environment as undisturbed as possible. We used a timer in the trigger firmware to delay the first transmission. This gave us enough time to leave the chamber after programming the dongles.
Between experiments, we repositioned the transmitters slightly in order to get results for a range of different positions.

3.5 Data Processing

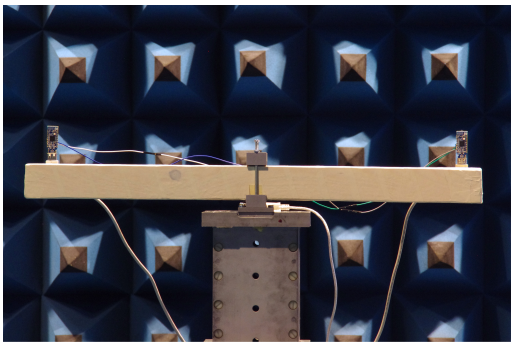
Due to our large parameter space, we collected a lot of data in our experiments. We created a web browser application to get an overview of our results and present them in a helpful way using Dash [38], a Python framework for data visualisation.
Besides organising and displaying the results, our data processing application also had to convert the transmit power settings used for the CT to RSSI values. To achieve this, we calculated the median of all 20 RSSI measurements collected for the transmit power and the associated mode and transmitter at the beginning of the run. This median was the value we mapped the transmit power to. After performing such a conversion for both transmitters involved in a pair of CT, we subtracted the RSSI medians to get an estimate of the received power delta.



(a) Overall setup for the CT experiments. The fixture on the left holds the receiver. The two transmitters are placed on the ends of the horizontal bar on the fixture on the right. The trigger is located between the transmitter behind the horizontal bar. The two fixtures are roughly 4.8 m apart.



(b) Receiver fixture.



(c) Transmitter fixture.

Figure 3.6: Physical setup used for the CT experiments.

Chapter 4

Results

In this chapter, we provide a brief overview of our CT experiments on the parameter space presented in Table 3.2. Due to time constraints not all of our runs covered all parameter choices: Since we were mostly interested in smaller time deltas for which we might expect constructive interference, we did not include some of the larger time deltas in every run. Also, for some of the lower transmit power settings, the power of the received signal was below the receiver sensitivity in some runs, even during the RSSI measurements when just one transmitter was transmitting. We excluded any CT sample for which we were unable to obtain an RSSI estimate from our results.
Due to the large size of the parameter space, we will not cover all of our measurements in this report. Instead, we invite the reader to explore our results using the interactive visualisation we created as part of this project and submitted with this report.
In the remainder of this chapter, we will focus on the general trends we observed in our results. We will use some of the plots from our visualisation to illustrate these trends. In the plots,

- each point corresponds to a PRR sample generated from 20 CT rounds,
- solid lines show medians on the PRR values and
- shaded areas denote 75 % confidence intervals on the medians.

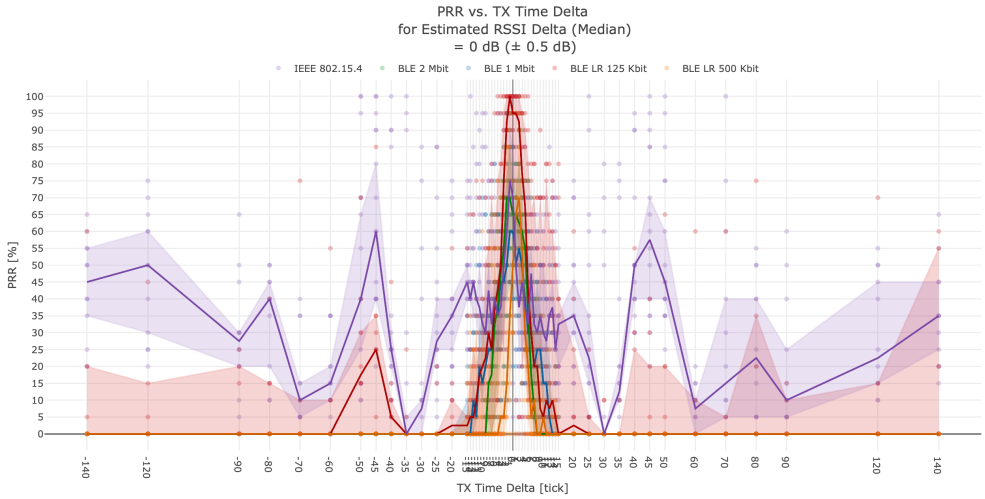
Our analysis is structured as follows: First, we examine the case of no power delta (Section 4.1). Next, in Section 4.2, we look at the performance of CT with large power offsets. In Section 4.3, we determine what constitutes such a "large power offset", i.e. we investigate for which parameter region the behaviour observed in Section 4.2 applies. We also address the CT performance for power deltas that do not fall into either of the parameter regions covered by Section 4.1 and Section 4.2.

4.1 No Power Difference

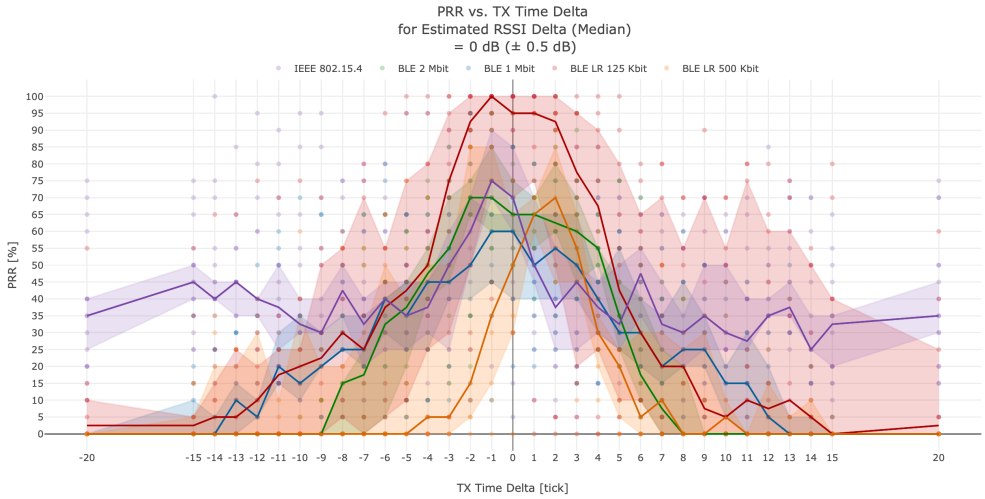
In this section look at the PRRs obtained for a power delta of zero for different time offsets. With no power delta, we expect to have to rely solely on constructive interference and coding for successful packet receptions since the capture effect does not apply. Note however, that RSSI measurements on the nRF52840 only have a granularity of 1 dB and an accuracy of ±2 dB. Thus, we cannot rule out that some power capture may occur, even when we estimate the received signal strength offset to be zero.
We will consider the cases of the same and different packet contents separately.

Same Packets

Our results for CT of identical packets under no power delta are shown in Figure 4.1. First, we consider Figure 4.1b, which focuses on the PRRs for small time offsets. For small time deltas we are clearly able to receive some packets. This can likely be attributed to constructive interference. What is interesting is the difference between the modes.
As expected, the high coding rate of BLE 125 Kbit seems to provide an advantage. If the time delta is 2 tick = 0.125 µs or less, we can still achieve perfect reception rates, even more reliably than for IEEE 802.15.4.
BLE 500 Kbit with its lower coding rate does not perform as well, even for very small time deltas we did not see a single run with a perfect PRR. It is surprising that BLE 500 Kbit seems to perform worse than BLE 2 Mbit. We would expect that its coding and lower symbol rate would give BLE 500 Kbit an advantage. Instead, the range of time deltas at which reception is possible is smaller than for BLE 2 Mbit, as are the median PRRs for each given time delta (except for some time deltas where the PRRs are comparable).
In fact, BLE 2 Mbit performs remarkably well overall, for time deltas smaller than 4 tick = 0.25 µs, it achieves higher median PRRs than BLE 1 Mbit, which has a longer symbol period. However, the range of time deltas for which any reception at all is possible is wider for BLE 1 Mbit.
The results for IEEE 802.15.4 differ significantly from those for the BLE modes. While we observe a slight peak in the median PRR around zero time delta, there is a lot of variation in PRRs across runs. Also, the median PRR does not drop to zero within the range of the small time delta values shown by figure Figure 4.1b. Looking at the results for the full range of time deltas we tested, which is shown in Figure 4.1a, we observe what seems to be some sort of periodic pattern IEEE 802.15.4. This looks very similar to the pattern observed when transmitting different data, which we discuss below.
Coding also seems to help with BLE 125 Kbit reception rates under larger time deltas. While most packets are lost, the reception rates for very large time deltas are larger than those for the other BLE modes.



(a) PRR measurements for the entire parameter range of time offsets.



(b) PRR samples with small time offsets.

Figure 4.1: PRR measurements from 12 runs for CT of identical packets with an estimated power difference of 0 dB ±0.5 dB. 1 tick = 62.5 ns.

Different Packets

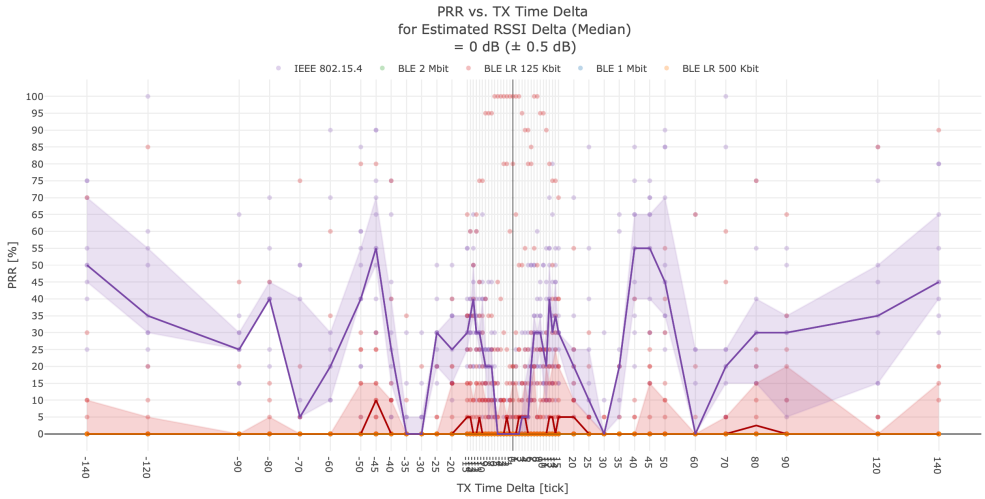
When concurrently transmitting different packets at no power difference, we do not receive anything for the BLE 1 Mbit, BLE 2 Mbit and BLE 500 Kbit modes. As we can see from Figure 4.2, in the BLE 125 Kbit mode, we do occasionally receive some packets. This is most likely made possible by the FEC with the high coding rate. It must also be noted that large parts of our packets are still identical due to the iBeacon structure we have chosen. It would be interesting to repeat our experiments with larger packet that contain more randomized content.
IEEE 802.15.4 is the only mode that enables us to achieve reasonably high reception rates with different packets and zero power delta. We attribute this to its use of DSSS.
In fact, the PRR pattern we observed for IEEE 802.15.4 across different time deltas looks very similar to that observed by Wilhelm *et al.* [19]. Their simulations show that the coding gain provided by the DSSS scheme employed by IEEE 802.15.4 for receivers with soft decision decoding is the highest for time deltas of $4kT_{ch} + 2T_{ch}, k \in \mathbb{Z}$, where the T_{ch} is the chip period. Since $T_{ch} = 8$ tick, we would expect to achieve higher PRRs for time deltas around 16, 48, 80, 112 and 144 tick. Figure 4.2a seems to confirm this.

4.2 Large Power Difference

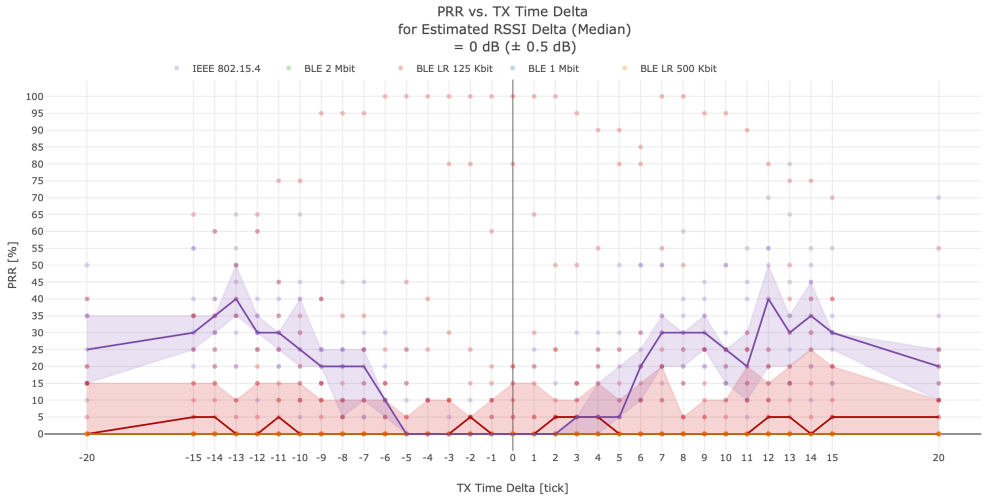
For very large power deltas we achieved perfect PRRs across all time deltas, modes and packet contents, barring a few outliers. We attribute this to power capture. In particular, for the BLE modes, this applies to power deltas above the receiver co-channel selectivity values specified by the Bluetooth standard and listed in Table 2.2. However, we observed that some power deltas below these levels are also sufficient. We examine the bounds for power capture more closely in the next section.

4.3 Medium Power Difference

From our experiments we observed that the minimum power delta required to achieve near-perfect PRRs through the capture effect seems to depend on the mode, packet content and time delta. We illustrate this for some example values in Table 4.1. In our discussion of this power delta threshold and the parameter range between zero and the threshold, we first consider the cases of different packets and the same packets separately.



(a) PRR measurements for the entire parameter range of time offsets.



(b) PRR samples with small time offsets.

Figure 4.2: PRR measurements from 12 runs for CT of packets with differing contents with an estimated power difference of 0 dB ±0.5 dB. 1 tick = 62.5 ns.

Time Delta [µs]	*Recommended Power Delta for PRR* $\geq 90\,\%$ [dB]				
	IEEE 802.15.4	*BLE 1 Mbit*	*BLE 2 Mbit*	*BLE 125 Kbit*	*BLE 500 Kbit*
			Same data		
0	2	3	4	1	3
0.125	3	5	5	2	7
0.25	4	6	6	2	8
0.5	4	8	9	6	12
0.9375	3	10	10	7	12
>0.9375	3	13	11	8	11
			Different Data		
0	4	10	9	2	6
0.125	4	11	9	3	6
0.25	4	11	9	3	9
0.5	5	10	11	4	9
0.9375	4	12	10	6	12
>0.9375	5	12	10	9	11

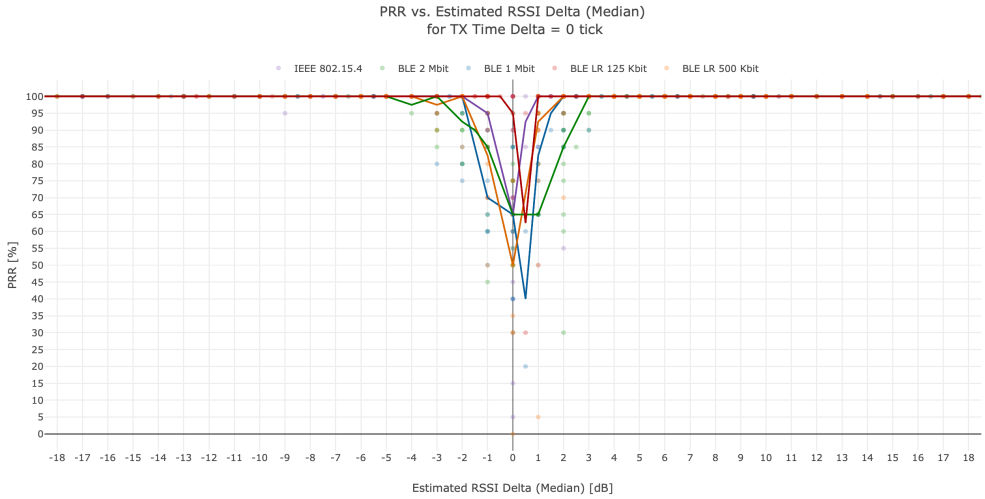
Table 4.1: Minimum power difference required to achieve PRRs $\geq 90\,\%$ using CT for different time deltas. Excludes outliers.

Different Packets

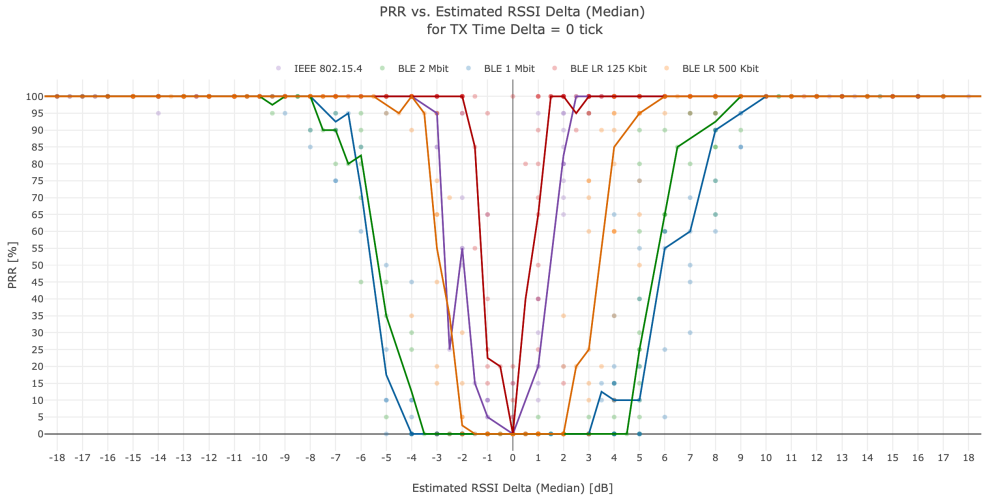
In theory, when different packets are transmitted, we would expect the required power delta for each mode to be the roughly the same across all time delta values. Since there should be no constructive interference, this would be the level at which power capture kicks in. Indeed, we observed this behaviour for the IEEE 802.15.4, BLE 1 Mbit and BLE 2 Mbit modes.
On the other hand, the BLE Coded PHYs produced some surprising results. In these modes, the power delta boundary appears to be dependent on the time delta. We do not have an explanation for this, but assume it is due to their coding. It would be interesting to investigate this further.
Overall, the uncoded BLE modes behave very similarly in terms of power capture. This is also illustrated by Figures 4.3b, 4.4b, 4.5b and 4.6b. Where power capture is concerned, BLE 500 Kbit outperforms the uncoded BLE modes for the time delta range we investigated. However, for larger time deltas, their performance difference shrinks. Power capture is possible at lower power deltas for the BLE 125 Kbit mode than for all other BLE modes. For time deltas $\lesssim 6\,\text{tick} = 0.375\,\mu\text{s}$, it even outperforms IEEE 802.15.4. However, for larger time deltas, IEEE 802.15.4 provides the best power capture performance.

Same Packets

When the transmitted packets are the same, there is a clear trend in the power delta boundary for all BLE modes: The minimum power delta required for good PRRs increases with the time delta. This is not only illustrated by Table 4.1, but also by Figures 4.3a, 4.4a, 4.5a and 4.6a. We assume that constructive interference helps to reduce the capture threshold for small time deltas.
When looking at CT of the same packets in the medium powerd delta range, BLE 500 Kbit is particularly interesting. We saw in Section 4.1 that BLE 500 Kbit seems to perform worse than the BLE uncoded modes under pure constructive interference. On the other hand, our investigation of the power capture of different packets seems to indicate it performs better than the BLE uncoded modes in terms of the capture effect. The question is how BLE 500 Kbit performs compared to the BLE uncoded modes when the two effects work together. Figure 4.7 gives us an indication. Even for small power offsets, the lower performance under constructive interference is not noticeable anymore.
Contrary to the BLE modes, IEEE 802.15.4 does not seem to exhibit a correlation between the minimum power delta required to achieve PRRs $\geq 90\,\%$ for CT with identical packets and the time delta.

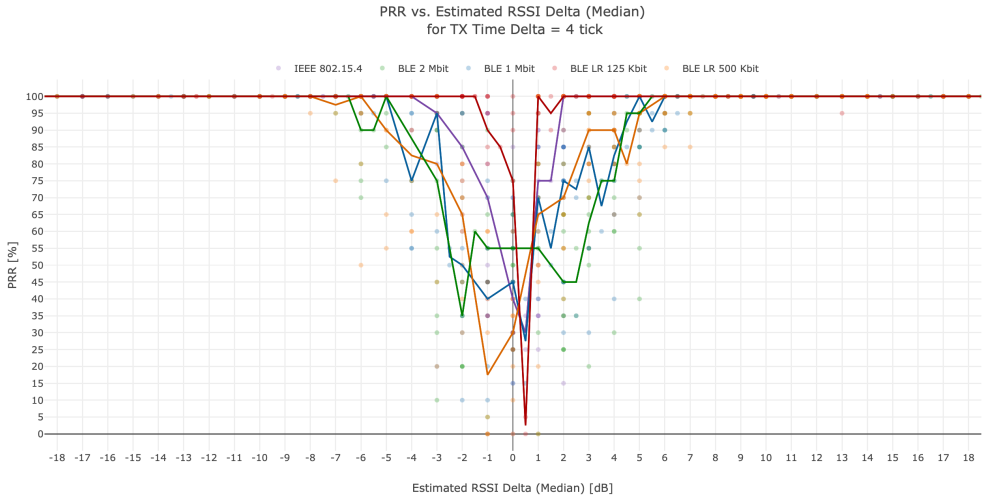


(a) PRR measurements from 12 runs for CT of packets with the same contents.

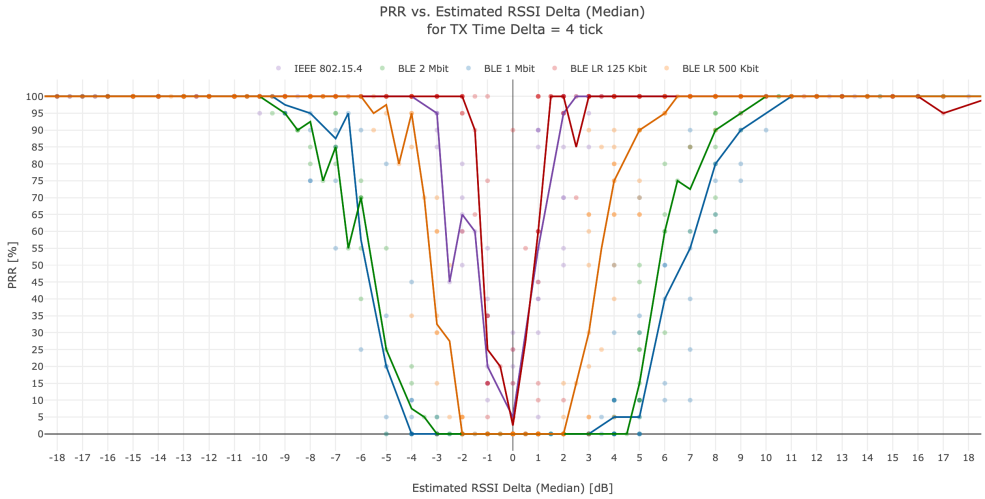


(b) PRR measurements from 12 runs for CT of packets with differing contents.

Figure 4.3: PRR measurements for CT with a time offset of 0 µs.

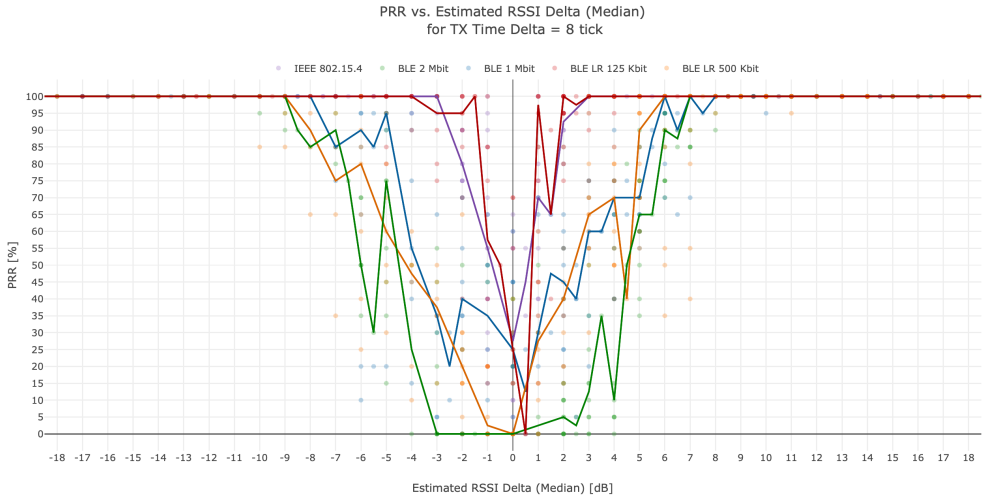


(a) PRR measurements from 12 runs for CT of packets with the same contents.

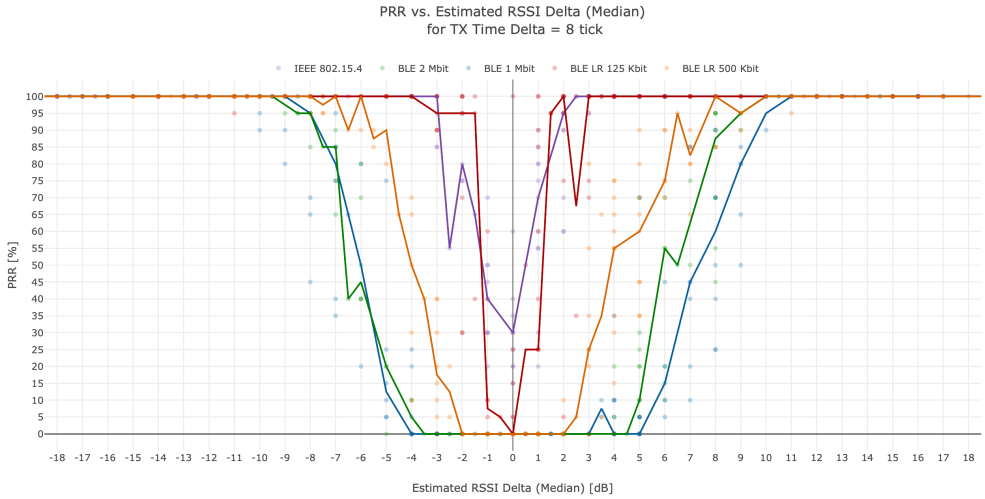


(b) PRR measurements from 12 runs for CT of packets with differing contents.

Figure 4.4: PRR measurements for CT with a time offset of 0.25 µs.

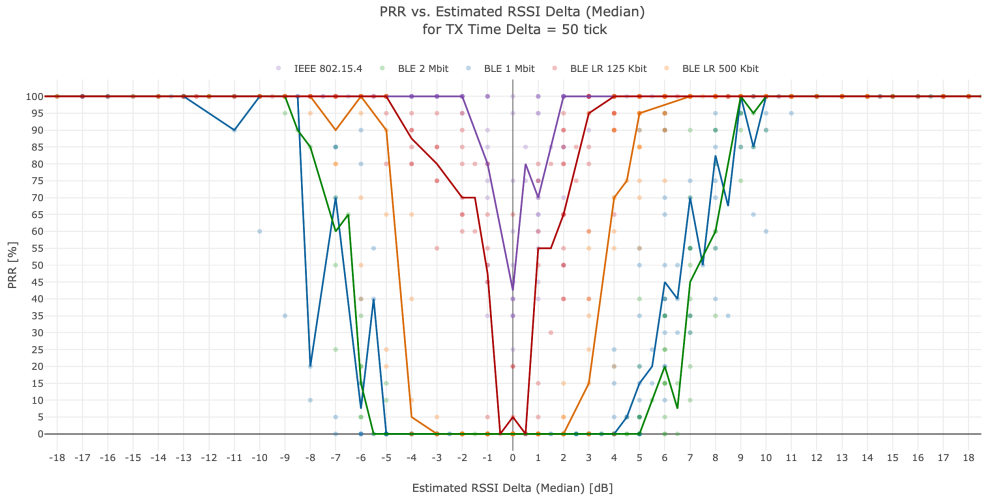


(a) PRR measurements from 12 runs for CT of packets with the same contents.

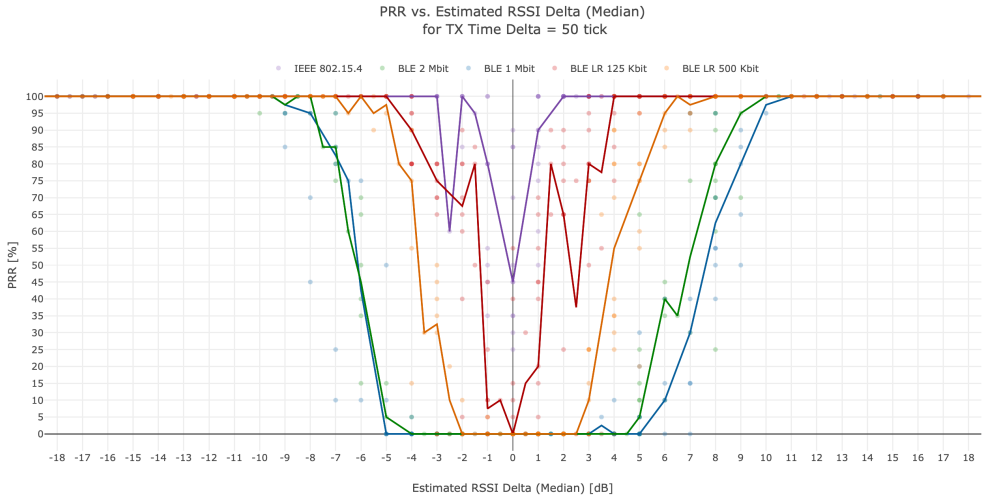


(b) PRR measurements from 12 runs for CT of packets with differing contents.

Figure 4.5: PRR measurements for CT with a time offset of 0.5 µs.

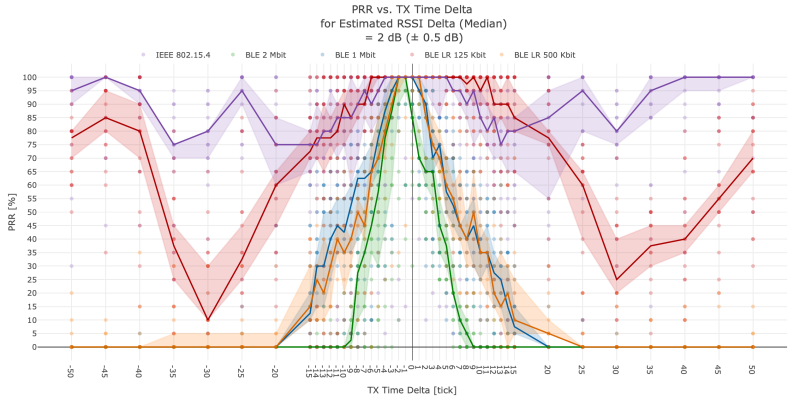


(a) PRR measurements from 12 runs for CT of packets with the same contents.

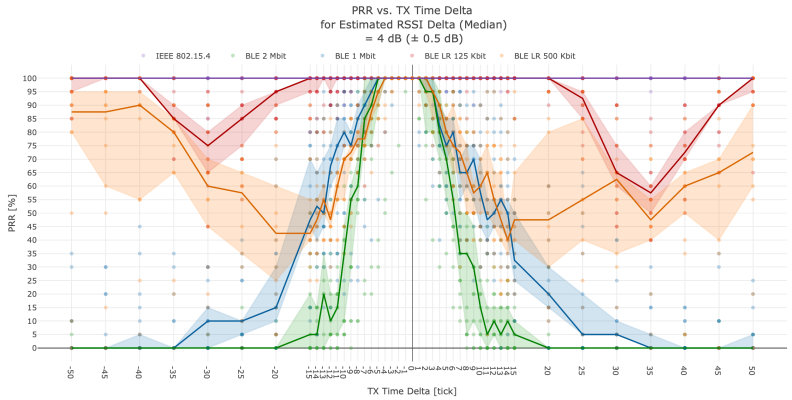


(b) PRR measurements from 12 runs for CT of packets with differing contents.

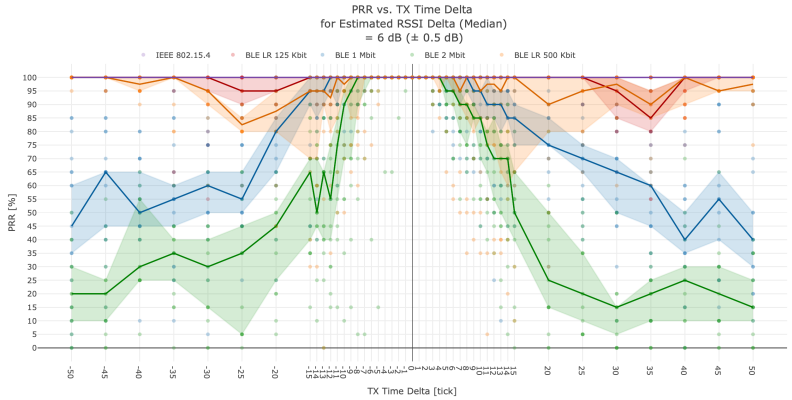
Figure 4.6: PRR measurements for CT with a time offset of 3.125 µs.



(a) PRR measurements for an estimated power difference of 2 dB ±0.5 dB.



(b) PRR measurements for an estimated power difference of 4 dB ±0.5 dB.



(c) PRR measurements for an estimated power difference of 6 dB ±0.5 dB.

Figure 4.7: PRR measurements from 12 runs for CT of identical packets with small time deltas. 1 tick = 62.5 ns.

Chapter 5

Conclusion

In this project, we experimentally investigated conditions for successful CT in BLE and IEEE 802.15.4 using nRF52840 dongles. We focused on the impact of the communication mode, power delta, time delta and packet contents.
Our results demonstrate that a successful reception of CT is indeed possible under either of the following conditions:

- *High power delta*: For very high power deltas, we achieved high PRRs independent of the values of the other parameters, presumably due to the capture effect. For lower power deltas, whether receptions were successful depended on the values of the other parameters.
- *Small time delta, same packets*: For small enough time deltas, we were able to receive concurrently transmitted packets with identical contents across all modes. This was the case even when no measurable power offset was present. However, in this case, only the BLE 125 Kbit mode achieved a median PRR >90 %. The maximum median PRRs for the other modes were between 60 % and 75 %. Even small power deltas were able to increase the performance significantly. An estimated power delta of 2 dB, produced median PRRs of 100 % across all modes for small enough time offsets.
- *IEEE 802.15.4 or BLE 125 Kbit PHY*: Even when neither of the other conditions was fulfilled, such as in the case of no power delta and different packets, we were able to receive CT when using these modes. In fact, IEEE 802.15.4 and BLE 125 Kbit performed better than or comparable to the other modes across all conditions. We assume this is because of their coding.

Based on these observations, it seems reasonable to assume that the key enablers of successful receptions of CT are power capture, coding and, to a lesser extent, constructive interference. It is difficult to predict how combinations of these effects will influence the performance of a pair of CT. Thus, our most significant contribution is the interactive visualisation of our results. It allows a user to

easily explore the impact of a wide range of different combinations on CT.
However, we do not recommend using our results as a quantitative predictor for the performance of CT in practical applications without further analysis. This is because our experiments did not investigate all potential influences on the successful reception of CT. In particular, the number of transmitters, carrier frequency offset, physical setup and environment may play a significant role. We leave the investigation of these effects up to future work, along with the following topics:

- *Theory behind CT over BLE*: An analytical study of CT over BLE, similar to the one performed by Wilhelm *et al.* [19] for IEEE 802.15.4, would be useful to gain an even more detailed understanding of the effects at play. In particular, it would be interesting to learn why the coded BLE modes behave so peculiarly for concurrent transmissions of the same packets at no power difference: While the PHY with $S = 8$ coding seems to benefit significantly from its coding, the PHY with $S = 2$ coding seems to perform worse than the uncoded modes. Also, a theoretical analysis might help us find an explanation for the relationship between the power capture threshold and the time delta in the coded modes for CT of different packets.

- *Performance tradeoffs*: We observed that IEEE 802.15.4 and BLE 125 Kbit achieved the best performance. However, note that these modes also have the lowest bitrates and highest redundancy in their transmissions. Thus, the transmission of a payload will take longer and require more energy than in the other modes. As a result, it could potentially be more efficient to use one of the other modes and retransmit the payload if the original transmission fails. Investigating these kinds of tradeoffs would be very valuable for the design of higher-level protocols based on CT.

Appendix A

Physical Layer Techniques

A.1 Gaussian-Filtered Frequency-Shift Keying (GFSK)

BLE uses binary *Gaussian-filtered frequency-shift keying (GFSK)* modulation, which is based on *continuous-phase frequency-shift keying (CPFSK)*. CPFSK modulation turns symbols into sinusoidal signals of different frequencies. In binary CPFSK, symbols directly represent bits. Thus, a bit 1 is associated with frequency f_1 and a bit 0 corresponds to frequency f_2. Following a notation similar to Haykin's [26], we can write a CPFSK signal for $0 \leq t \leq T_s$ as

$$s(t) = \begin{cases} \sqrt{\frac{2E_s}{T_s}} \cos\left(2\pi f_1 t + \theta\left(0\right)\right) & \text{for symbol 1} \\ \sqrt{\frac{2E_s}{T_s}} \cos\left(2\pi f_2 t + \theta\left(0\right)\right) & \text{for symbol 0} \end{cases} \tag{A.1}$$

where

E_s = energy per symbol
T_s = symbol duration
$\theta(0)$ = phase at time $t = 0$

We can rewrite the frequencies in terms of an unmodulated *carrier frequency* f_c and a *frequency deviation* f_d, which we define as

$$f_c = \frac{f_1 + f_2}{2} \tag{A.2}$$

$$f_d = |f_1 - f2| \tag{A.3}$$

Assuming $f_1 > f_2$ without loss of generality, we can rewrite Equation (A.1) as

$$s(t) = \sqrt{\frac{2E_s}{T_s}} \cos\left(2\pi f_c t + \theta\left(0\right) + 2\pi f_d t m_0\right), \qquad 0 \leq t \leq T_s \tag{A.4}$$

where $m_0 = 1$ if the symbol is 1 and $m_0 = -1$ if the symbol is 0. Assume $m_i \in \{\pm 1\}$ is transmitted during $iT_s \leq t \leq (i+1)T_s$. In CPFSK, the phase $\theta(t)$

needs to be continuous. Thus, we require

$$\begin{aligned}
\theta(t) &= \theta(0) + 2\pi f_d t m_0, && 0 \le t \le T_s \\
\theta(t) &= \theta(0) + 2\pi f_d T_s m_0 + 2\pi f_d (t - T_s) m_1, && T_s \le t \le 2T_s \\
\theta(t) &= \theta(0) + 2\pi f_d T_s m_0 + 2\pi f_d T_s m_1 + 2\pi f_d (t - 2T_s) m_2, && 2T_s \le t \le 3T_s \\
&\vdots \\
\theta(t) &= \theta(0) + 2\pi f_d T_s \sum_{j=0}^{i-1} m_j + 2\pi f_d (t - iT_s) m_i, && iT_s \le t \le (i+1)T_s
\end{aligned} \tag{A.5}$$

We can write a representation of the sequence of transmitted symbols in the form of a sum of rectangular pulses $r(t)$ of duration T_s and amplitude $1/2T_s$:

$$m(t) = \sum_i m_i r(t - iT_s) \qquad r(t) = \begin{cases} 1/2T_s & 0 \le t \le T_s \\ 0 & \text{otherwise} \end{cases} \tag{A.6}$$

Using Equation (A.6) we can rewrite Equations (A.5) as

$$\theta(t) = \theta(0) + 4\pi f_d T_s \int_{-\infty}^{t} m(\tau) d\tau \tag{A.7}$$

It is common to describe a CPFSK modulation in terms of the *modulation index* h, which relates the frequency deviation to the symbol duration:

$$h = 2 f_d T_s \tag{A.8}$$

With the modulation index, we can simplify the signal model for CPFSK to

$$s(t) = \sqrt{\frac{2E_s}{T_s}} \cos\left(2\pi f_c t + \theta(0) + 2\pi h \sum_{i=-\infty}^{\infty} m_i \int_{-\infty}^{t} r(\tau - iT_s) d\tau\right) \tag{A.9}$$

The rectangular nature of symbol pulses causes abrupt changes in frequency. These abrupt changes result in a large transmission bandwidth, which is undesirable. To mitigate this, GFSK passes the input through a *Gaussian pulse-shaping filter* before frequency modulation. The filter smoothes the transitions between symbols. A Gaussian pulse-shaping filter with a 3 dB baseband bandwidth B has an impulse response of

$$h_G(t) = \sqrt{\frac{2\pi}{\ln 2}} B \exp\left(-\frac{2\pi^2}{\ln 2} B^2 t^2\right) \tag{A.10}$$

Its response $g(t)$ when applied to a rectangular pulse

$$\tilde{r}(t) = \begin{cases} 1 & 0 \le t \le T_s \\ 0 & \text{otherwise} \end{cases} \tag{A.11}$$

and normalized to

$$\int_{-\infty}^{\infty} g(t) = \frac{1}{2} \tag{A.12}$$

is given by [39]

$$g(t) = \frac{1}{4T_s}\left(\operatorname{erfc}\left(\pi BT_s\sqrt{\frac{2}{\ln 2}}\left(-\frac{t}{T_s}\right)\right) - \operatorname{erfc}\left(\pi BT_s\sqrt{\frac{2}{\ln 2}}\left(1-\frac{t}{T_s}\right)\right)\right) \tag{A.13}$$

where $\operatorname{erfc}(u)$ is the *complementary error function*

$$\operatorname{erfc}(u) = \frac{2}{\sqrt{\pi}}\int_{u}^{\infty} \exp\left(-t^2\right) dt \tag{A.14}$$

The *time-bandwidth product* BT_s is a design parameter. The signal model for GFSK is the result of replacing the rectangular pulses $r(t)$ with the filtered pulses $g(t)$ in the CPFSK signal model (Equation (A.9)):

$$s(t) = \sqrt{\frac{2E_s}{T_s}}\cos\left(2\pi f_c t + \theta(0) + 2\pi h\sum_{i=-\infty}^{\infty} m_i \int_{-\infty}^{t} g(\tau - iT_s)d\tau\right) \tag{A.15}$$

A.2 Offset Quadrature Phase-Shift Keying (O-QPSK)

IEEE 802.15.4 modulation of chip sequences is based on *offset quadrature phase-shift keying (O-QPSK)* [4]. *Quadrature phase-shift keying (QPSK)* represents information in terms of four different symbols. Each of these symbols is associated with a phase. The four different phase values are equally spaced and typically chosen as $\pi/4$, $3\pi/4$, $5\pi/4$ and $7\pi/4$. For symbol $i \in [1, 4]$, this results in the signal

$$s_i(t) = \begin{cases} \sqrt{\frac{2E_s}{T_s}}\cos\left(2\pi f_c t + (2i-1)\frac{\pi}{4}\right) & 0 \le t \le T_s \\ 0 & \text{otherwise} \end{cases} \tag{A.16}$$

With the definitions of the *in-phase carrier*

$$\phi_I(t) = \sqrt{\frac{2}{T_s}}\cos\left(2\pi f_c t\right), \qquad 0 \le t \le T_s \tag{A.17}$$

and the *quadrature carrier*

$$\phi_Q(t) = \sqrt{\frac{2}{T_s}}\sin\left(2\pi f_c t\right), \qquad 0 \le t \le T_s \tag{A.18}$$

Table A.1: QPSK signal space for Gray-coding.

symbol	chip pair		phase	message point coordinates	
i	even chip	odd chip	[radians]	$s_{i,I}$	$s_{i,Q}$
1	1	0	$\pi/4$	$+\sqrt{E_s/2}$	$-\sqrt{E_s/2}$
2	0	0	$3\pi/4$	$-\sqrt{E_s/2}$	$-\sqrt{E_s/2}$
3	0	1	$5\pi/4$	$-\sqrt{E_s/2}$	$+\sqrt{E_s/2}$
4	1	1	$7\pi/4$	$+\sqrt{E_s/2}$	$+\sqrt{E_s/2}$

we can also express the signal as

$$s_i(t) = s_{i,I}\phi_I(t) + s_{i,Q}\phi_Q(t), \qquad 0 \le t \le T_s \tag{A.19}$$

where

$$s_{i,I} = \sqrt{E_s}\cos\left((2i-1)\frac{\pi}{4}\right) \tag{A.20}$$

$$s_{i,Q} = -\sqrt{E_s}\sin\left((2i-1)\frac{\pi}{4}\right) \tag{A.21}$$

Encoding the pairs of chips into symbols using Gray-encoding yields the mapping shown in Table A.1. With this mapping, the value of an even chip is directly associated with the sign of the in-phase component. The values of odd chips are linked to the sign of the quadrature component.
A closer look at the mapping reveals that a change in both the even and odd chip between symbols results in a phase shift of $\pm 180°$. When the signal is filtered, this can result in large amplitude variations [26, 40]. This also happens for phase changes of $\pm 90°$, which are caused by a change in value of a single chip, but to a much lesser extent. O-QPSK limits phase changes to $\pm 90°$ by staggering the in-phase and quadrature components:

$$\phi_I(t) = \sqrt{\frac{2}{T_s}}\cos\left(2\pi f_c t\right), \qquad 0 \le t \le T_s \tag{A.22}$$

$$\phi_Q(t) = \sqrt{\frac{2}{T_s}}\sin\left(2\pi f_c t\right), \qquad \frac{T_s}{2} \le t \le \frac{3T_s}{2} \tag{A.23}$$

Thus, the signal model for O-QPSK modulation is given by

$$s(t) = \sqrt{\frac{2E_s}{T_s}}\sum_{k=-\infty}^{\infty} c_{2k}\cos\left(2\pi f_c t\right)p(t-kT_s) - c_{2k+1}\sin\left(2\pi f_c t\right)p\left(t-\left(kT_s+\frac{T_s}{2}\right)\right) \tag{A.24}$$

where $c_j = -1$ if the jth chip is 0, $c_j = 1$ if the jth chip is 1 and

$$p(t) = \begin{cases} \frac{1}{\sqrt{2}} & 0 \le t \le T_s \\ 0 & \text{otherwise} \end{cases} \tag{A.25}$$

In the 2.4 GHz band, IEEE 802.15.4 uses a modified version of this basic O-QPSK modulation: Each chip is spread across two chip periods ($T_s = 2T_{ch}$, $E_s = 2E_{ch}$) and half-sine pulse-shapes $\tilde{p}(t)$ are used instead of rectangular ones $p(t)$ [4]:

$$s(t) = \sqrt{\frac{2E_{ch}}{T_{ch}}} \sum_{k=-\infty}^{\infty} c_{2k} \cos\left(2\pi f_c t\right) \tilde{p}(t-2kT_{ch}) - c_{2k+1} \sin\left(2\pi f_c t\right) \tilde{p}\left(t-(2k+1)\,T_{ch}\right) \tag{A.26}$$

$$\tilde{p}(t) = \begin{cases} \sin\left(\frac{\pi t}{2T_{ch}}\right) & 0 \le t \le 2T_{ch} \\ 0 & \text{otherwise} \end{cases} \tag{A.27}$$

O-QPSK with half-sine pulse shaping is equivalent to *minimum shift keying (MSK)* [19, 41]. To be consistent with popular notation when showing this, we reindex the chips such that odd-indexed chips are modulated onto the in-phase carrier and even-indexed chips are modulated onto the quadrature carrier:

$$s(t) = \sqrt{\frac{2E_{ch}}{T_{ch}}} \sum_{j=-\infty}^{\infty} c_{2j-1} \cos\left(2\pi f_c t\right) \tilde{p}(t-(2j-1)\,T_{ch}) - c_{2j} \sin\left(2\pi f_c t\right) \tilde{p}\left(t-2jT_{ch}\right) \tag{A.28}$$

Using the identities

$$\sin\left(\alpha - n\pi + \frac{\pi}{2}\right) = \cos\left(\alpha\right)\cos\left(n\pi\right), \qquad \alpha \in \mathbb{R}, n \in \mathbb{Z} \tag{A.29}$$

$$\sin\left(\alpha - n\pi\right) = -\sin\left(\alpha\right)\cos\left(n\pi\right), \qquad \alpha \in \mathbb{R}, n \in \mathbb{Z} \tag{A.30}$$

$s(t)$ can be rewritten as an MSK signal

$$s(t) = I\left(t\right)\cos\left(\frac{\pi t}{2T_{ch}}\right)\cos\left(2\pi f_c t\right) + Q\left(t\right)\sin\left(\frac{\pi t}{2T_{ch}}\right)\sin\left(2\pi f_c t\right) \tag{A.31}$$

where

$$I(t) = \sqrt{\frac{2E_{ch}}{T_{ch}}} \sum_{j=-\infty}^{\infty} I_j(t) \tag{A.32}$$

$$I_j(t) = \begin{cases} c_{2j-1}\cos\left(\pi j\right) & (2j-1)\,T_{ch} \le t \le (2j+1)\,T_{ch} \\ 0 & \text{otherwise} \end{cases} \tag{A.33}$$

$$= \begin{cases} \cos\left(\frac{\pi}{2}\left(2j+1-c_{2j-1}\right)\right) & (2j-1)\,T_{ch} \le t \le (2j+1)\,T_{ch} \\ 0 & \text{otherwise} \end{cases} \tag{A.34}$$

$$Q(t) = \sqrt{\frac{2E_{ch}}{T_{ch}}} \sum_{j=-\infty}^{\infty} Q_j(t) \tag{A.35}$$

$$Q_j(t) = \begin{cases} -c_{2j}\cos\left(\pi j\right) & 2jT_{ch} \le t \le (2j+2)\,T_{ch} \\ 0 & \text{otherwise} \end{cases} \tag{A.36}$$

$$= \begin{cases} -\sin\left(\frac{\pi}{2}\left(2j+1-c_{2j}\right)+\frac{\pi}{2}\right) & 2jT_{ch} \le t \le (2j+2)\,T_{ch} \\ 0 & \text{otherwise} \end{cases} \tag{A.37}$$

MSK can also be viewed as a special case of CPFSK (see Appendix A.1 or [26]) with modulation index $h = \frac{1}{2}$. To make the frequency modulation more explicit, we can also write $s(t)$ as

$$s(t) = \cos\left(2\pi f_c t + d\,(t)\,\frac{\pi t}{2T_{ch}} + \theta\,(t)\right) \tag{A.38}$$

where

$$d\,(t) = -I\,(t)\,Q\,(t) \qquad \in \{-1, 1\} \tag{A.39}$$

$$\theta\,(t) = \frac{\pi t}{2}\,(1 - I\,(t)) \qquad \in \{0, \pi\} \tag{A.40}$$

More details on the relationship between O-QPSK, MSK and CPFSK can be found in [42].

A.3 Spread Spectrum Techniques

Spread-spectrum techniques are widely used in digital communications to avoid narrowband interference by spreading a data signal across a wider bandwidth than necessary using a spreading code that is independent of the data transmitted. [26] Popular spread-spectrum techniques include *frequency-hopping spread-spectrum (FHSS)* as well as *direct-sequence spread-spectrum (DSSS)*.

A.3.1 Direct-Sequence Spread Spectrum (DSSS)

In *direct-sequence spread-spectrum (DSSS)*, the data signal is multiplied with a *pseudo-noise (PN)* signal of a higher bitrate, which is known to both the sender and receiver. The resulting bits at this higher bitrate are called *chips*. The *spreading factor* describes how many chips form a single data bit. The high-bitrate PN signal has a much wider bandwidth than the data signal. Thus, the power of the resulting signal is spread across a wide frequency band.
At the receiver, the received signal is modulated with the same PN signal. If no noise was present in the channel, this will transform the wideband signal back into the original narrowband data signal. Even if there was additive narrowband interference in the channel, the receiver can recover the original signal. The multiplication with the PN sequence spreads the power of the narrowband noise across a wide frequency band, while the original signal has its power concentrated within a narrow band. Thus, the data and noise signal are clearly distinguishable.
The type of DSSS modulation the IEEE 802.15.4 physical layer [4] uses differs slightly from this conventional notion of DSSS. It employs a 16-ary quasi-orthogonal DSSS modulation technique, where 4 bits form a symbol and each symbol is then mapped to one of 16 nearly-orthogonal 32-chip PN sequences. This corresponds to a spreading factor of 8.

A.3.2 Frequency Hopping Spread Spectrum (FHSS)

In *frequency-hopping spread-spectrum (FHSS)* the data signal is spread sequentially by switching carrier frequencies during the transmission of the signal [26]. BLE uses FHSS [7] to improve reliability. While we do not consider this feature in our investigation of CT in the physical layer, it is worth noting that FHSS can improve the performance of higher level protocols, including those based on CT, as shown by several successful entries to the EWSN dependability competition [43], such as [16, 17].

Bibliography

[1] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh, "Efficient Network Flooding and Time Synchronization with Glossy," in *Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks*. IEEE, 2011, pp. 73–84.

[2] Zigbee Alliance, "Zigbee 3.0 — Zigbee Alliance." [Online]. Available: https://zigbee.org/zigbee-for-developers/zigbee-3-0/ (accessed October 16, 2019).

[3] Thread Group, "What is Thread." [Online]. Available: https://www.threadgroup.org/What-is-Thread (accessed October 16, 2019).

[4] *IEEE Standard for Low-Rate Wireless Networks (802.15.4-2015)*. USA: IEEE, 2016.

[5] Bluetooth SIG, "Bluetooth Technology Website." [Online]. Available: https://www.bluetooth.com (accessed October 16, 2019).

[6] Bluetooth SIG, "Radio Versions — Bluetooth Technology Website." [Online]. Available: https://www.bluetooth.com/bluetooth-technology/radio-versions/ (accessed October 16, 2019).

[7] C. Gomez, J. Oller, and J. Paradells, "Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology," *Sensors*, vol. 12, no. 9, pp. 11 734–11 753, 2012.

[8] *Bluetooth Core Specification*, Bluetooth SIG, December 2016, Version 5.0.

[9] M. Woolley and S. Schmidt, "Bluetooth 5 Go Faster. Go Further," *Bluetooth SIG*, 2017.

[10] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele, "Low-power Wireless Bus," in *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, ser. SenSys '12. New York, NY, USA: ACM, 2012, pp. 1–14.

[11] O. Landsiedel, F. Ferrari, and M. Zimmerling, "Chaos: Versatile and Efficient All-to-all Data Sharing and In-network Processing at Scale," in *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '13. New York, NY, USA: ACM, 2013, pp. 1:1–1:14.

[12] D. Yuan, M. Riecker, and M. Hollick, "Making Glossy Networks Sparkle: Exploiting Concurrent Transmissions for Energy Efficient, Reliable, Ultra-low Latency Communication in Wireless Control Networks," in *European Conference on Wireless Sensor Networks*. Springer, 2014, pp. 133–149.

[13] T. Istomin, A. L. Murphy, G. P. Picco, and U. Raza, "Data Prediction + Synchronous Transmissions = Ultra-low Power Wireless Sensor Networks," in *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*, ser. SenSys '16. New York, NY, USA: ACM, 2016, pp. 83–95.

[14] P. Zhang, A. Y. Gao, and O. Theel, "Less is More: Learning More with Concurrent Transmissions for Energy-Efficient Flooding," in *Proceedings of the 14th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*. ACM, 2017, pp. 323–332.

[15] R. Lim, R. Da Forno, F. Sutton, and L. Thiele, "Competition: Robust Flooding Using Back-to-Back Synchronous Transmissions with Channel-Hopping," in *Proceedings of the 2017 International Conference on Embedded Wireless Systems and Networks*, ser. EWSN '17. USA: Junction Publishing, 2017, pp. 270–271.

[16] A. Escobar, F. Moreno, A. J. Cabrera, J. Garcia-Jimenez, F. J. Cruz, U. Ruiz, J. Klaue, A. Corona, D. Tati, and T. Meyerhoff, "Competition: BigBangBus," in *Proceedings of the 2018 International Conference on Embedded Wireless Systems and Networks*, ser. EWSN '18. USA: Junction Publishing, 2018, pp. 213–214.

[17] A. Escobar-Molero, J. Garcia-Jimenez, J. Klaue, F. Moreno-Cruz, B. Saez, F. J. Cruz, U. Ruiz, and A. Corona, "Competition: RedNodeBus, Stretching out the Preamble," in *Proceedings of the 2019 International Conference on Embedded Wireless Systems and Networks*, ser. EWSN '19. USA: Junction Publishing, 2019, pp. 304–305.

[18] V. S. Rao, M. Koppal, R. V. Prasad, T. V. Prabhakar, C. Sarkar, and I. Niemegeers, "Murphy loves CI: Unfolding and Improving Constructive Interference in WSNs," in *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE, 2016, pp. 1–9.

[19] M. Wilhelm, V. Lenders, and J. B. Schmitt, "On the Reception of Concurrent Transmissions in Wireless Sensor Networks," *IEEE Transactions on Wireless Communications*, vol. 13, no. 12, pp. 6756–6767, Dec 2014.

[20] C. Liao, Y. Katsumata, M. Suzuki, and H. Morikawa, "Revisiting the So-Called Constructive Interference in Concurrent Transmission," in *2016*

IEEE 41st Conference on Local Computer Networks (LCN), Nov 2016, pp. 280–288.

[21] J. Park, J. Jeong, H. Jeong, C. M. Liang, and J. Ko, "Improving the Packet Delivery Performance for Concurrent Packet Transmissions in WSNs," *IEEE Communications Letters*, vol. 18, no. 1, pp. 58–61, January 2014.

[22] C. Gezer, C. Buratti, and R. Verdone, "Capture effect in IEEE 802.15.4 networks: Modelling and experimentation," in *IEEE 5th International Symposium on Wireless Pervasive Computing 2010*, May 2010, pp. 204–209.

[23] B. Al Nahas, S. Duquennoy, and O. Landsiedel, "Concurrent Transmisions for Multi-hop Bluetooth 5," in *Proceedings of the 2019 International Conference on Embedded Wireless Systems and Networks*, 2019.

[24] E. W. Weisstein, "Harmonic Addition Theorem. From MathWorld—A Wolfram Web Resource," http://mathworld.wolfram.com/HarmonicAdditionTheorem.html.

[25] K. Leentvaar and J. Flint, "The Capture Effect in FM Receivers," *IEEE Transactions on Communications*, vol. 24, no. 5, pp. 531–539, 1976.

[26] S. S. Haykin, *Communication Systems*, 4th ed. New York: Wiley, 2001.

[27] D. Son, B. Krishnamachari, and J. Heidemann, "Experimental Study of Concurrent Transmission in Wireless Sensor Networks," in *Proceedings of the 4th international conference on Embedded networked sensor systems*. ACM, 2006, pp. 237–250.

[28] A. Escobar-Molero, "Improving reliability and latency of Wireless Sensor Networks using Concurrent Transmissions," *at-Automatisierungstechnik*, vol. 67, no. 1, pp. 42–50, 2019.

[29] D. Davis and S. Gronemeyer, "Performance of Slotted ALOHA Random Access with Delay Capture and Randomized Time of Arrival," *IEEE Transactions on Communications*, vol. 28, no. 5, pp. 703–710, May 1980.

[30] *nRF52840 Dongle — PCA10059 v1.0.0 — User Guide*, Nordic Semiconductor, January 2019, v1.1.

[31] *nRF52840 Product Specification*, Nordic Semiconductor, February 2019, v1.1.

[32] "The Contiki Operating System." [Online]. Available: https://github.com/contiki-os/contiki (accessed October 18, 2019).

[33] B. Al Nahas, S. Duquennoy, and O. Landsiedel, "BlueFlood." [Online]. Available: https://github.com/iot-chalmers/BlueFlood (accessed October 19, 2019).

[34] "Contiki OS Radio API." [Online]. Available: https://github.com/contiki-os/contiki/blob/master/core/dev/radio.h (accessed October 18, 2019).

[35] Apple Inc., *Proximity Beacon Specification*, September 2015, Release R1. [Online]. Available: https://developer.apple.com/ibeacon/

[36] Bluetooth SIG, "Assigned numbers and GAP — Bluetooth Technology Website." [Online]. Available: https://www.bluetooth.com/specifications/assigned-numbers/generic-access-profile/ (accessed October 16, 2019).

[37] *Supplement to the Bluetooth Core Specification*, Bluetooth SIG, January 2019, Version 8.

[38] "Dash User Guide," Plotly. [Online]. Available: https://dash.plot.ly (accessed October 21, 2019).

[39] A. F. Molisch, *Wireless Communications*, 2nd ed. Chichester, West Sussex, U.K: Wiley, 2011.

[40] S. Pasupathy, "Minimum shift keying: A spectrally efficient modulation," *IEEE Communications Magazine*, vol. 17, no. 4, pp. 14–22, 1979.

[41] S. Gronemeyer and A. McBride, "MSK and Offset QPSK Modulation," *IEEE Transactions on Communications*, vol. 24, no. 8, pp. 809–820, 1976.

[42] F. Xiong, *Digital Modulation Techniques*, ser. Artech House Telecommunications Library. Artech House, 2000.

[43] M. Schuß, C. A. Boano, M. Weber, and K. Römer, "A Competition to Push the Dependability of Low-Power Wireless Protocols to the Edge," in *Proceedings of the 2017 International Conference on Embedded Wireless Systems and Networks*, ser. EWSN '17. USA: Junction Publishing, 2017, pp. 54–65.

Index