Coexistence Study of Different Medium Access Mechanisms Using a Software Defined Radio Testbed

Alexander Pastor

Bachelor Thesis

October 24, 2017

Examiners

Prof. Dr. Petri Mähönen
Prof. Dr.-Ing. Marina Petrova

Supervisors

Prof. Dr. Petri Mähönen
Peng Wang, M.Sc.
Andra Voicu, M.Sc.

Institute for Networked Systems
RWTH Aachen University

The present work was submitted to the Institute for Networked Systems

Coexistence Study of Different Medium Access Mechanisms Using a Software Defined Radio Testbed

Bachelor Thesis

presented by
Alexander Pastor

Prof. Dr. Petri Mähönen
Prof. Dr.-Ing. Marina Petrova

Aachen, October 24, 2017

_____

(Alexander Pastor)

Acknowledgements

I would like to thank my thesis advisors Andra Voicu and Peng Wang for guiding me through the process of formation of this thesis. Specifically, I am grateful for Andra's detailed feedback on earlier versions empowering me to improve on the overall quality. Furthermore, I appreciate the efforts of Peng, who taught me how to use his GNU Radio MAC framework and dealing with hardware and other practical problems. Moreover, I owe special thanks to David Offermans and Neal F. Moran for proofreading.

Alexander Pastor

Contents

ABSTRACT

The demand for higher wireless transmission rates and capacity is growing rapidly. As a consequence, options to more efficiently make use of the limited frequency resources are being explored. Recently, it was proposed that operators of cellular networks should make use of license-free frequency bands that were not originally designated for their purposes. However, due to the exemption of license fees these bands are densely populated and measures for peaceful coexistence with incumbent technologies must be taken. In the case of the license-free 2.4 GHz band multiple technologies already coexist, namely IEEE 802.11 (Wi-Fi), Bluetooth and IEEE 802.15.4 (ZigBee). With this in mind, it is only natural to take mechanisms of technologies into account that were designed for contention-based coexistence, in our case the CSMA/CA protocol used in Wi-Fi devices. This thesis experimentally examines how different medium access control protocols determine the overall performance of the nodes. To this end, we use USRP software-defined radio devices and GNU Radio to employ different medium access control protocols, namely CSMA/CA, 1-persistent CSMA and ALOHA in various combinations on two links, where the focus lies on the influence of CSMA timing aspects. Our measurements reveal that the appropriate choice of timing parameters is crucial to the performance of the devices in different traffic situations concerning throughput and frame delays. On the one hand, it is important that the transmission channel is not idle due to excessive sensing periods in spite of backlogged nodes. On the other hand, it is important to prevent nodes starting transmission prematurely causing collisions due to inaccuracies related to the time granularity of the overall system.

1

Introduction

In this chapter we motivate the thesis and explain its structure by briefly summarizing the contents of each chapter.

A great number of technologies already operate in the unlicensed bands, however their number and density is still expected to increase. One example to this claim is that licensees of dedicated frequency bands aim at extending their bandwidth by making use of unused capacity in unlicensed bands to accommodate the growing number of users and the demand for higher transfer rates. Particularly, LTE Unlicensed aggregates carriers in the license-free 5 GHz band already populated with Wi-Fi devices [1], [2]. However, the original LTE technology was not designed to coexist with other technologies in the same channel. Particularly, LTE in the licensed bands relies on the fact that all access to the physical medium is coordinated by a base station [3]. Another unlicensed band, namely the 2.4 GHz band, is currently much more populated. IEEE 802.11 (Wi-Fi), Bluetooth and IEEE 802.15.4 (ZigBee) devices all coexist in the 2.4 GHz band [4]. The specifications and standards of these three technologies already offer coexistence mechanisms especially in view of rapid network densification [5]. In order to facilitate harmonious coexistence of devices in the same channel the appropriate design of medium access control (MAC) protocols to avoid collisions is decisive, because collisions may render all transmitted data useless. The goal of this thesis is to examine how different MAC mechanisms and the choice of related parameters affect the network performance in terms of throughput and other metrics. Our results are based on measurements with universal software radio peripherals (USRPs) which are physical, programmable devices, making use of the flexibility of software-defined radios. In contrast to other inter-technology coexistence studies, such as [6] and [7] on LTE-U/Wi-Fi coexistence, [8], [9] on Zigbee/Wi-Fi coexistence and [10] on Bluetooth/Wi-Fi coexistence, we focus on timing aspects of CSMA/CA, the MAC protocol used in Wi-Fi, such as interframe spacing, backoff slot duration and contention window.

The rest of the thesis is structured as follows. In Chapter 2 we discuss the theoretical foundations for the ensuing experiments. We classify and introduce a number of MAC protocols considering their strengths and weaknesses. Furthermore, we will briefly introduce the main concepts of the software tool GNU Radio, that we used for our experiments. The purpose of Chapter 3 is to put our work into the context of related work. In Chapter 4 we give an overview of the conducted experiments. Moreover, we define the metrics that we have considered and show how our automated measurement scripts greatly reduce the required user effort to obtain results. Chapter 5 contains the measurement results and our interpretation concerning the fitness of the protocols for harmonious coexistence. In Chapter 6 we discuss the main findings of Chapter 5 and give an outlook on possible starting points for future work.

2

BACKGROUND

In this chapter the theoretical foundations for the succeeding work are treated. Firstly, the Medium Access Control (MAC) layer is introduced in the context of the OSI reference model. Successively, a glance on a number of different MAC protocols and mechanisms is taken, while discussing performance with respect to the challenges and goals in wireless transmission. The chapter concludes with describing the advantages of software-defined radio (SDR) and how GNU (GNU is not Unix) Radio can be used to support SDR.

2.1 MAC PROTOCOLS

2.1.1 MAC Layer in the OSI Model

The OSI (Open Systems Interconnection) model is a layered architecture that divides a telecommunication system into several manageable layers. It features seven layers, where the second layer - the Data Link Layer (DLL) - can be split into two sublayers. The focus of this thesis lies on the lower sublayer, which is MAC. The upper sublayer is Logical Link Control (LLC). Table 2.1 gives a short overview of the responsibilities of each layer. We now take a closer look at the MAC functionalities in IEEE 802.11 (WLAN) networks.

MAC Functionalities The MAC layer provides the functionalities to enable connectionless (datagram style) transfer of data between nodes. It transparently carries the data of the next higher - the LLC layer - as service data unit (SDU). Other important functions include frame delimiting and recognition, addressing of destination stations, conveying the source-address, protection against errors with frame check sequences and controlling the access to physical medium [11]. In this thesis we will only examine physical medium access aspects.

2.1.2 Challenges for Wireless MAC Protocols

Wireless MAC protocols have to tackle a few problems that do not occur in wired data exchange. Among them are the hidden node and the exposed node problem, which will be discussed by reference to Figure 2.1. Further challenges, such as energy limitations will also be delineated.

2.1.2.1 The Hidden Node and the Exposed Node Problem

Suppose that the radio range of the nodes in 2.1 is limited to the neighboring nodes and A would like to transmit to B. If C just started transmitting, A does not hear C

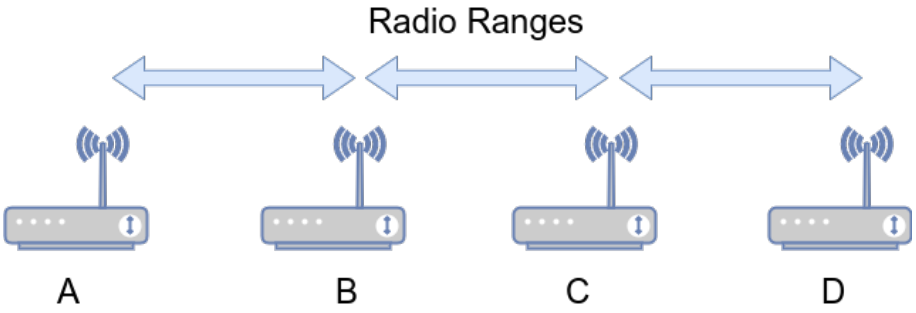


FIGURE 2.1: Setup to explain the hidden and exposed node problem. Each node can only reach its direct neighbors.

and falsely assumes that the channel is idle and start transmitting, which leads to a collision. This is the hidden node problem [13] [14].

For the same configuration, in another scenario B would like to send to A and C is already transmitting to D. B refrains from sending although collisions would only take place between B and C, where it does not matter as both B and C are transmitters. This is the exposed node problem [13] [14] .

2.1.2.2 *Power Problems*

Further challenges when designing MAC protocols include the power conservation when faced with constrained power resources, as e.g. in wireless sensor networks (WSN) where devices rely on batteries for their power supply. Attempts to reduce energy consumption have been made in several specialized, duty-cycle based MAC

Level	Layer	Principal Functionalities
1	Physical Layer	Dealing with mechanical, electrical and timing interfaces of data transmission
2	DLL: MAC Sublayer	Controlling medium access and frame synchronization
2	DLL: LLC Sublayer	Multiplexing to enable different network protocols coexist, flow control and error control
3	Network Layer	Routing and congestion control
4	Transport Layer	Transmission reliability, same-order-delivery, congestion avoidance
5	Session Layer	Token management, dialog control, synchronization
6	Presentation Layer	Abstracting syntax and semantics of transmission, encryption
7	Application Layer	User application protocols, such as http, ftp, smtp and many more

TABLE 2.1: Layers in the OSI model [12].

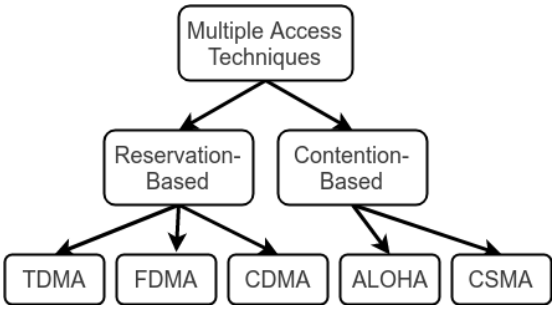


FIGURE 2.2: Classification of MAC techniques as in [16].

protocols for WSN such as Sensor MAC, Timeout MAC and Berkeley MAC as in more detail shown in Section 2.1.6.

As a consequence of the constrained energy resources, WSN are especially susceptible to denial of sleep attacks, a special form of denial of service (DoS) attack, drastically increasing energy consumption and thus reducing the system lifetime. It is due to this fact that security is paramount in biomedical or military fields of application [15].

2.1.3 Classification of MAC Protocols

Traditional MAC protocols can be classified into one of two groups: reservation-based and contention-based as depicted in Figure 2.2 [16]. The difference between them is that in reservation-based protocols a coordinator prevents collisions by assigning physical resources to devices, whereas in contention-based protocols no such infrastructure exists and nodes have to contend for channel utilization, hence the name. Another technique independent from these categories is to employ duty cycles (DC), where nodes continuously alternate between active and inactive periods. According to [17] the appropriate choice of MAC protocol depends on a plethora of design-drivers such as requirements concerning throughput, latency, energy consumption and traffic patterns.

We proceed with discussing representative protocols of the two categories. Thereafter, we take a look at a few protocols that use DC mechanism.

2.1.4 Reservation-Based MAC Protocols

Reservation-based protocols may implement an array of desirable features, but require knowledge of network topology in order to allow each node to communicate with every other on the basis of a centrally coordinated schedule. These features include reduced collisions, fairness among nodes or multiple transmissions at the same time. Since we do not incorporate reservation-based protocols in our experiments we only briefly describe the basic principles of the time-division multiple access (TDMA), frequency-division multiple access (FDMA) and code-division multiple access (CDMA).

TDMA is a representative protocol in this group, which divides time into slots. Each node is assigned to a unique slot during which it may transmit. As a result we obtain collision-free transmission, predictable scheduling delays, high throughput in

heavy load situations and fairness among nodes. However, both the knowledge of topology and tight synchronization require large overheads or expensive hardware [17].

FDMA (FDMA) divides a frequency band into a number of channels. One or more may be assigned to each node. Receivers use bandpass filters to obtain the transmitted signal [16].

CDMA is a digital spread-spectrum technique where multiple transmitters share the same frequency band and transmissions may occur at the same time. In this method transmitted signals are combined (XORed) with special[1] sequences making the transmitted signal's frequency vary in order to avoid interference. The receiver has to follow along this variation of frequency (that is to say know the spreading code) when decoding to retrieve the original data signal, resulting in increasing security as a side effect [16].

It is also possible to combine several of these techniques. Making use of this, the base station (eNB) of LTE in the licensed band coordinates traffic by assigning physical resource blocks (PRB) to devices. A PRB is a combination of a frequency and a time slot based on the reservation techniques of orthogonal FDMA (OFDMA) and TDMA.

2.1.5 Contention-Based MAC Protocols

2.1.5.1 ALOHA

ALOHA is arguably the most simple MAC protocol. Whenever a device wants to send data it just does so. The higher the channel load, i.e. transmissions per time unit, the more likely collisions will occur, which may render all transmitted information useless.

The question is how likely it is that a collision does not occur. In other words, how efficient is an ALOHA channel? Making a statement requires a few preliminary assumptions as shown in [13]:

1. We simplify the calculation by assuming a fixed frame length.
2. The number of packets generated during a frame time is a Poisson-distributed random variable X.
3. The channel load G comprises of two portions: "new" and retransmitted frames.

The probability mass function of the Poisson distribution and thus the probability of k frames being generated during a given frame time amounts to:

$$Pr(X = k) = \frac{G^k \cdot e^{-G}}{k!} \tag{2.1}$$

The probability of zero frames being generated during the transmission of the frame is $Pr(X = 0) = e^{-G}$ (assumption 3). If no collision occurs during the transmission of frame F, no other frame was sent during that transmission. Conversely, F itself

[1] by special we mean that the signal has certain properties such as orthogonality and in some cases pseudo-randomness

did not collide with a frame sent off prior to F. We conclude that the vulnerability period during which collisions may corrupt data is two frame times (assumption 2).

The probability that no frame other than the frame to be transmitted is generated during the two-frame-time vulnerability period is $P_0 = e^{-2G}$. The throughput S is given by $S = GP_0 = Ge^{-2G}$.

The maximum throughput is achieved when $\frac{\partial S}{\partial G} \stackrel{!}{=} 0$:

$$\begin{aligned} \frac{\partial S}{\partial G} &= \frac{\partial}{\partial G} G e^{-2G} && (2.2)\\ &= e^{-2G}(1-2G) && (2.3)\\ &\stackrel{!}{=} 0 && (2.4)\\ \Leftrightarrow \quad G &= 0.5 && (2.5) \end{aligned}$$

This means that for $G = 0.5$ the throughput S reaches its maximum $S_{\text{ALOHA,max}} = \frac{1}{2e} \approx 0.18$. This result is very reasonable, since the transmission of a frame is vulnerable for the duration of two frame times, so the maximum is achieved when sending exactly every second slot, where a slot is equivalent to the frame time.

We note that, the throughput can be doubled with slotted ALOHA. In contrast to pure ALOHA, slotted ALOHA divides time into slots, where transmissions may only commence at the beginning of slots, which effectively halves the vulnerability period to only one slot, since frames transmitted prior to a frame F cannot interfere with F anymore. Thus, $S_{\text{ALOHA,max}} = \frac{1}{e} \approx 0.36$, reached at $G = 1$. However, this comes at the cost of an additional frame delay of t_{slot} in the worst case and $\frac{t_{\text{slot}}}{2}$ in the average case and the need for synchronization.

As shown in Figure 2.3, ALOHA's performance is discouraging and improvements over ALOHA were found.

2.1.5.2 CSMA

The main problem of ALOHA is the negligence of concurrent traffic in the channel. A solution to this problem is offered by "listen before talk" (LBT) mechanisms, which means in order to avoid collisions we make a clear channel assessment (CCA) and refrain from sending should it be busy. This is the simple, yet effective basic idea of carrier sensing multiple access (CSMA) which comes in three basic flavors, i.e. 1-persistent CSMA, non-persistent CSMA and p-persistent CSMA as depicted in Figure 2.4 which is discussed next.

1-Persistent CSMA If the channel is busy, 1-persistent CSMA waits until the channel becomes idle. As soon as the channel is found idle a frame is transmitted with a probability of 1, hence 1-persistent CSMA. If the frame collides with another, the node waits for a random backoff time and then the whole process is started all over again.

Despite being a substantial improvement over ALOHA, this protocol has at least two problems [13]:

- Provided propagation delay is zero or negligible, collisions can still occur. Imagine a three-node-scenario with nodes A, B and C. A is transmitting, while

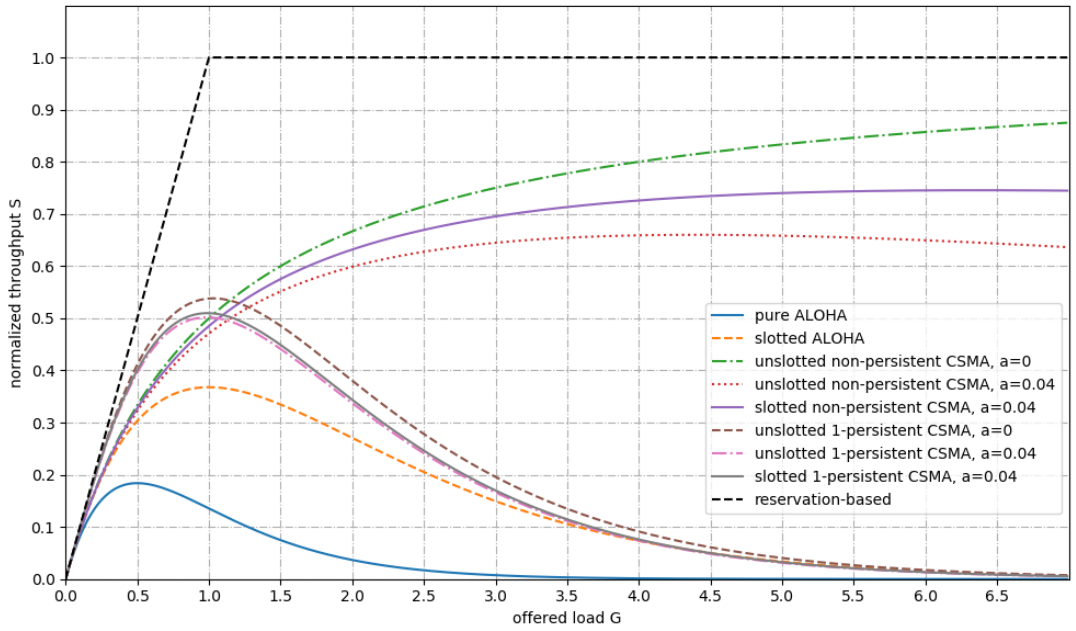


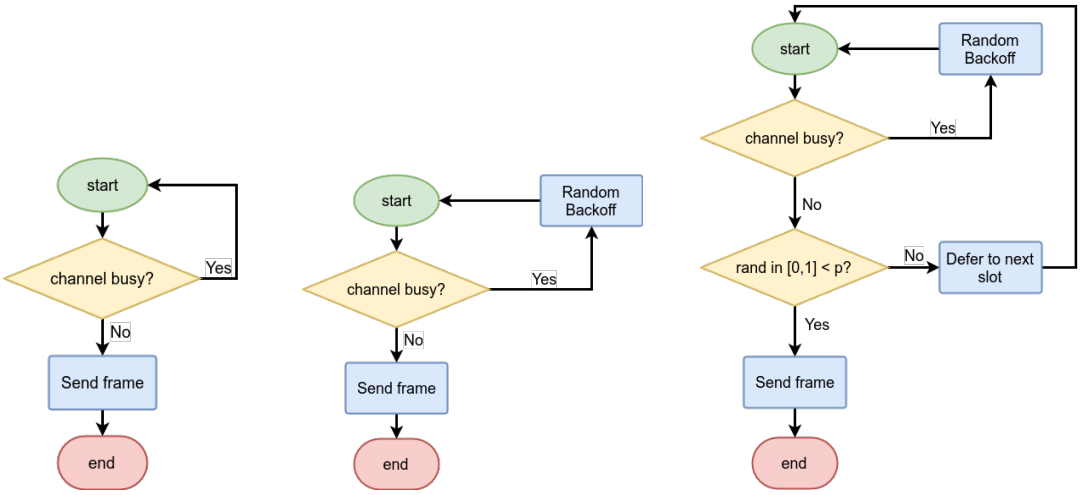
FIGURE 2.3: Normalized throughput over offered load according to formulae in [13], [16], [17], with $a = \tau/T_p$, where τ is the maximum propagation delay and T_p the packet transmission time and under the assumptions made in Section 2.1.5.1.

B and C are waiting for their turn. Once A finishes transmission B and C simultaneously start their transmissions leading to collision.

- If propagation delay is not negligible the protocol suffers from an additional problem. In another scenario A has just begun sending. B assumes the channel is idle and send off his frame, since, due to the propagation delay, B has not yet heard of A. This is why propagation delay may significantly hamper the performance of this protocol.

Non-Persistent CSMA In order to alleviate 1-persistent CSMA's problem with several nodes trying to seize the channel as soon as it becomes idle, a less greedy attempt is made with non-persistent CSMA. Instead of continuously sensing the channel until it becomes idle, the nodes wait a random backoff time until they listen again. As a result, this protocol leads to better channel utilization with the downside of higher delays.

P-Persistent CSMA P-persitent CSMA is a protocol for slotted channels. Whenever a node A wishes to send a packet, the channel is sensed. If the channel is found idle the node transmits its packet with a probability of p. With a probability $1 - p$ the node defers its transmission to the next slot. This process is repeated until either the packet is sent or the channel is found busy again. In the latter case A acts **as though** a collision had taken place and waits a random time until starting again [13].



(a) 1-persitent CSMA (b) non-persistent CSMA (c) p-persistent CSMA

FIGURE 2.4: The three basic flavors of CSMA according to [13].

This flavor of CSMA can be regarded as a compromise between 1-persistent CSMA and non-persistent CSMA, where the choice of p determines the greediness. The smaller p, the less greedy and thus the closer p-persistent CSMA approximates non-persistent behavior. An appropriate choice of p can get the best out of both mechanisms: minimal delays as in 1-persistent CSMA, as well as high channel efficiency as in non-persistent CSMA.

2.1.5.3 CSMA with Collision Detection

A way to further improve the CSMA protocols is to immediately cancel transmissions once a collision is detected. There is no point in continuing these transmissions, as the transmitted data is lost in any case and aborting the transmission saves bandwidth, time and energy.

CSMA with Collision Detection (CSMA/CD) is used on wired LANs and serves as basis of the wide-spread Ethernet. However, this mechanism is not extensively made use of in wireless networks. Concerning the reason, it is cardinal to understand that collision detection is an analog process. A collision is detected by comparing the energy or pulse width of the received and transmitted signals, which premises transmission and reception taking place simultaneously. This condition is seldom met for wireless nodes, which are mostly half-duplex. The reason for this lies in the conservation of energy, since wireless signals spread in all directions around their origin and thus degrade exponentially with the distance. Furthermore, wireless channels are typically much more noisy than their wired counterparts and suffer from multipath fading. To make up for the loss in signal strength we would have to employ expensive signal processing in order to recover fainter signals. Alternatively, we could increase the transmit power, but this increases interference with other nodes, as well as energy consumption.

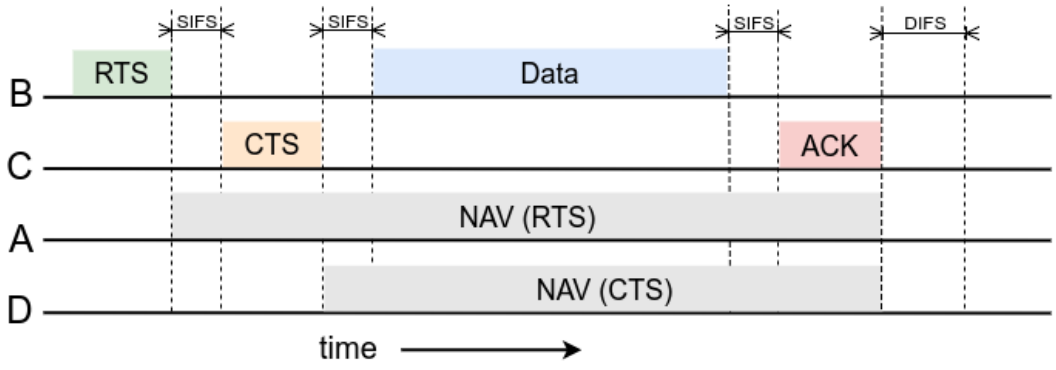


FIGURE 2.5: Virtual carrier sensing in CSMA/CA, as described in [13] and [14].

2.1.5.4 CSMA with Collision Avoidance

IEEE 802.11 is a set of physical layer (PHY) and MAC specifications for wireless local area networks (WLANs). When the dominant mode of operation, the so-called distributed coordination function (DCF) is employed CSMA/CA is used in the MAC layer, which we discuss next in accordance to [14] and [16].

As depicted in Figure 2.5 there are specific intervals of given length between each of the frames. Varying lengths of these interval types serve the purpose of prioritizing certain frames over others.

The short interframe spacing (SIFS) is the interval until the next control frame or next fragment (of a fragmented data frame) may be sent. SIFS is designed to allow one node out of the two nodes in dialog to have a higher priority to access the channel than uninvolved nodes. The longer interval DCF interframe spacing (DIFS) is the interval after which any station may try to seize the channel for their transmission.

For the sake of completeness, we briefly mention two further intervals used in IEEE 802.11, namely point coordination function interframe spacing (PIFS) and extended interframe spacing (EIFS). If IEEE 802.11 operates in an alternative mode of operation, where a node acts as point coordinator of traffic the standard prescribes an interval of length PIFS to allow the controlling node to send certain control (beacon and poll) frames. EIFS is used to report the reception of a bad or unknown frame and due to the low priority of this action is the longest interval among the mentioned four.

Physical carrier sensing takes place in these intervals. If a node wants to transmit a packet and the channel is sensed busy in one of these intervals then the node defers its transmission and launches the binary exponential backoff (BEB) procedure [14]. With BEB a node picks a slot in the so-called contention window (CW). The picked slot is just a random integer and the contention window is a range lower-bounded by zero and upper-bounded by $2^{n+m} - 1$, where m is a fixed integer, $2^m - 1$ called minimum CW (CW_{min}) and $n = 0$ for the moment. After picking the slot the node waits for $t_w = \#\text{slot} \cdot t_s$, where #slot is the number of the slot, t_s a constant called backoff slot duration or simply backoff slot. After t_w has elapsed the channel is sensed again. In the case it is busy again the whole BEB procedure is repeated with n incremented by 1, thus the CW doubled, hence **binary exponential** backoff. The motivation for CW_{min} is to greatly reduce the chance that two contending nodes pick the same slot in the first round of BEB defeating the purpose of BEB. The BEB mechanism is also used if

collisions occur. Once a data frame is transmitted a timer is started, which is canceled when the corresponding ACK is received. If the timer runs out, i.e. no ACK was received, a collision is assumed and a round of BEB precedes the next try.

Beside physical carrier sensing, another mechanism, namely virtual carrier sensing using RTS/CTS exchange is optionally employed to mitigate the problems caused by hidden nodes. In order to explain these mechanisms we refer to the setup of Figure 2.1. Figure 2.5 visualizes the chain of events whose explanation follows.

B wants to send to C, hence issues a request to send (RTS). Every node receiving the RTS remains silent, except for C that in response to the RTS creates a clear to send (CTS) frame. Not only B receives this CTS frame, but also D, a hidden node from B's point of view. Upon reception of CTS D is silenced as well. Therefore, RTS/CTS is addressing the hidden node problem. RTS/CTS are frames of 30 bytes length containing the length of the frame that, in this case, B wants to transmit. Based on this length, A and D setup so-called network allocation vectors (NAV), which are node-internal timers reminding A and D that the channel is still in use. This mechanism is called virtual carrier sensing because nodes defer their transmission based on the information received through other frames.

2.1.5.5 Licensed Assisted Access

For both license assisted access (LAA) as well as LTE-U, the unlicensed band is only used to enhance the downlink rate in LTE traffic. The procedure of allocating additional carriers is called carrier aggregation (CA), and due to its limitation to downlink traffic more specifically supplemental downlink (SDL) CA. All control traffic is still sent through the licensed bands as it may exclusively be used by the licensee and thus is generally more reliable in terms of quality of service [2]. One principal approach to ensure harmonious coexistence of LTE and Wi-Fi in the unlicensed band is License Assisted Access (LAA), relying on LBT. Since the LBT mechanism of LAA largely resembles CSMA/CA[2] it seems quite natural to assume it will coexist better with Wi-Fi than LTE-U which uses DCs as discussed in Section 2.1.6.1 [18].

2.1.6 Duty-Cycle MAC Protocols

In duty-cycle MAC schemes nodes repeatedly alternate between active and inactive phases. In some protocols, especially in those designed for WSNs, nodes may sleep when inactive to reduce idle listening and thus energy consumption. Due to increased contention during active phases these protocols are mostly designed for limited contention traffic situations as in WSNs. The fraction of an active period in a cycle is called duty factor.

2.1.6.1 LTE-U

LTE-U uses Carrier Sense Adaptive Transmission (CSAT), which tries to avoid primary channels of Wi-Fi transmissions and other LTE-U operators. If that is not possible, duty-cycles are dynamically adapted depending on Wi-Fi medium utilization

[2]It actually resembles CSMA/CA hybrid coordination function (HCF) enhanced distributed channel access (EDCA), where data packets with higher priority have a higher chance of being sent.

(MU). If MU is below a certain threshold the DC is increased. If it is between that threshold and a higher one, the DC is kept constant, otherwise it is decreased [2].

2.1.6.2 Sensor MAC (SMAC)

In SMAC the active period is divided into to a synchronization and a data transmission phase. During sync phase nodes transmit SYNC packets. Nodes receiving SYNC packets adopt the schedule carried by the packet and broadcast into their neighborhood. Nodes that follow the same schedule form a virtual cluster. Borderline nodes between virtual clusters adopt multiple schedules and thus have an increased duty factor. During contention period SMAC features the RTS/CTS exchange and fragments data frames, which are transmitted in a burst to reduce collision likelihood. The duty factor per schedule is **predetermined** on the basis of expected load as the result of an optimization problem on the competing goals of reducing idle listening and contention. The higher the duty factor the more idle-listening and the less contention occurs [17] [19].

2.1.6.3 Timeout MAC (TMAC)

While TMAC shares the same principle of schedule establishment with SMAC nodes adaptively vary duty factors depending on expected traffic. Furthermore, TMAC shifts all communication to the beginning of the active period. This allows nodes to sleep earlier should no traffic be detected during a certain time period. In variable load situations TMAC saves as much as five times more energy compared to SMAC at the cost of increased latency [17].

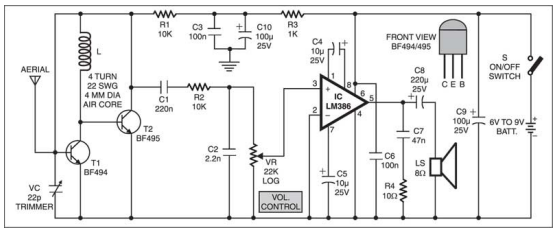
2.1.6.4 Berekeley MAC (BMAC)

Still, TMAC maintains common active phases at high energy expenses. BMAC drops the requirement of maintaining common active phases. Instead payload is preceded by extended preambles such that every receiver is able to reliably detect packets. This has the effect of shifting energy expenses from the receiving to the sending side, which saves energy in low load applications such as surveillance. Furthermore, in BMAC CCA is based on outlier detection, instead of thresholding like in CSMA, further reducing energy use [20].

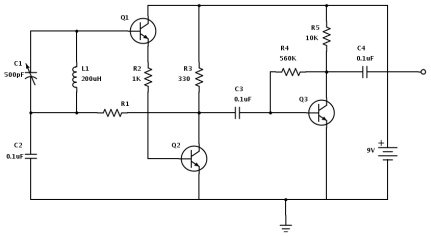
2.2 SOFTWARE-DEFINED RADIO

Traditional radio equipment is "hardware-defined", i.e. that the signal processing runs on a specialized electrical circuit. This has the potential advantages of efficient energy use and cheap production at the cost of limited flexibility in operation.

In SDR signal processing components such as filters, amplifiers, modulators, detectors and many more are implemented in software and mostly run on general-purpose processors, sometimes in combination with digital speech processors (DSPs) and field programmable gate arrays (FPGAs). Since changes to the SDR components can be made simply by rewriting the code, SDR development is much more flexible (and cheaper) than the design of equivalent electrical circuits.



(a) FM Receiver [21]



(b) AM Receiver [22]

FIGURE 2.6: Simple do-it-yourself (DIY) radio receiver circuit diagrams.

While the limitations of hardware-defined radios are acceptable for a number of applications, such as e.g. self-made radio receivers as shown in Figure 2.6, it is very desirable to get rid of these limitations for rapid prototyping of new technologies including but not limited to cognitive radio, software-defined antennas and wireless mesh networks. In the case of this thesis SDR simplifies studying the influence of different MAC mechanisms.

2.2.1 GNU Radio

The GNU Radio (GR) project is dedicated to the evolution of a free and open-source software development kit (SDK) enabling both the creation of actual software-defined radio, as well as simulated signal processing. Written in C++ and Python, GNU Radio also comes with the intuitive graphical software GNU Radio Companion (GRC) that allows creating block diagrams called flowgraphs simply by connecting signal processing blocks into a directed graph. Its target user market is not merely limited to research and industry, but also encompasses academia, government and private users [23].

A proprietary, well-documented alternative to GNU Radio is LabVIEW developed by National Instruments [24]. LabVIEW takes a purely graphical approach similar to GRC relying on block diagrams, but lacks the freedom of user-defined block creation with a programming language such as C++ or Python without extra efforts, such as buying a Python integration toolkit [25].

Mathworks MATLAB/Simulink also provides a communication systems toolbox. However, the devices we used are not on the list of officially supported devices [26].

2.2.2 Flowgraphs and Blocks

The two most basic concepts of GNU Radio are flowgraphs and blocks. As mentioned in 2.2.1 flowgraphs are directed graphs, whose nodes are functional blocks and whose vertices determine the direction of data flow [27].

The behavior of these blocks is programmed in either Python or C++, where the latter is recommended for performance-critical applications, which is also why the blocks in our flowgraphs are all written in C++. If performance is less critical Python is a superior choice since it is more concise and allows faster prototyping as there is no need for compilation. Each block generally serves exactly one purpose for the sake

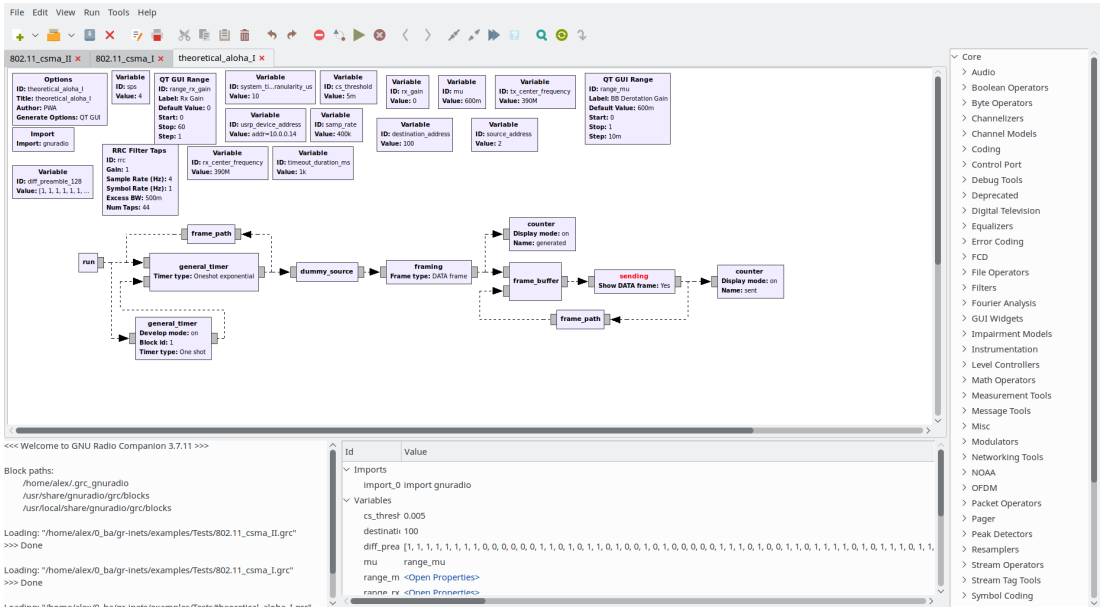


FIGURE 2.7: GNU Radio Companion GUI.

of modularity. Blocks in turn can be composed of an arbitrary number of inner blocks, making extensive use of the modularity and hiding implementation complexity from the user, much like a blackbox in electrical circuits. These composed blocks are called hierarchical blocks. In our case the complete PHY layer is hidden in hierarchical blocks called "sending" and "receiving".

Blocks are connected through ports, which can either be input or output ports. Depending on which types of ports a block has, it can either be a source, sink or neither of the former. Each input port only consumes data of a specific data type. Similarly, each output port only produces data of a specific data type. The set of types ranges from integers, floating point and complex numbers to messages and a bunch of others. Since each block implements a certain function these ports can be regarded as input parameters and return values of a function, respectively.

2.2.3 *Message Passing and Stream Tags*

When designing packet-based protocols, such as MAC protocols it is of tremendous importance to be able to detect packet data unit (PDU) boundaries. For this purpose GR provides an asynchronous message passing system. A synchronous alternative is to attach so-called stream tags to the "infinite" stream of data. The former method is the right choice when designing MAC protocols due to the asynchronous nature of packet delivery [27] [28].

2.2.4 *Polymorphic Types and SWIG*

Polymorphic types (PMT) are opaque data types that enable safe information exchange across blocks by serving as generic containers of data. Self-evidently, the original data type must be retained as a PMT class member. For thread-safety reasons PMT are immutable. We make extensive use of PMTs when passing messages. As an

aside, note that the Python PMT class has some powerful tools unavailable its C++ counterpart, making use of Python's weak typing [28].

Simplified wrapper and interface generator (SWIG) is a software that helps to connect code written in C or C++ to a variety of scripting languages, such as in our case Python. This is achieved by generating a Python module from the C/C++ code with the help of an interface file. This "compatibility layer" is necessary, because blocks can be written in either Python or C++ as mentioned earlier.

2.2.5 GNU Radio Modules

A GNU Radio module is a set of blocks that are grouped together based on their purpose. A basic GNU Radio installation already provides a lot of modules, but almost all of them are associated with the PHY layer. The PHY layer used in this thesis was implemented in GNU Radio flowgraphs by Julian Arnold. For the MAC layer we use flowgraphs based on blocks of a GNU Radio out-of-tree (OOT)[3] module programmed by Peng Wang. The changes we made to the MAC layer flowgraphs include additional blocks to capture the metrics as described in Section 4.4 and shutting off self-reception during frame transmission.

[3] An OOT module is an external module not provided in the standard setup of GR.

3

Related Work

This chapter introduces work related to the thesis. We will highlight similarities and differences of various studies to our work.

In the following section we discuss approaches and results of studies that examined the coexistence of different technologies in the same frequency band. Studies concerning inter-technology coexistence are based on at least one of the following: theoretical analysis, simulation or measurements with physical devices. Due to the fact, that we also carry out experimental research using USRPs, we are more interested in studies based on the latter. Furthermore, studies based on measurement with real devices have the benefit of better reflecting system level details of the technologies and providing insight for real-world deployments. However, as pointed out in [6] vendor-specific properties of the test hardware must be taken into account since they may exert great influence on the measurement results. Some studies propose new mechanisms for one of the technologies which we do not do.

LTE-U/Wi-Fi Coexistence Although a great number of simulation-based studies [1], [29], [30], [31] exist on this topic we will confine the discussion to two studies [6], [7] based on measurements with physical devices. Both studies evaluate LTE Unlicensed /Wi-Fi coexistence based on LTE-U using srsLTE, an open-source SDR library to implement the PHY layer of LTE. Another common feature of both studies is the use of USRPs as LTE nodes.

In [6] the testbed comprises of several Wi-Fi and LTE links, for which they used Ettus USRP B210 boards (LTE) and low-power single-board computers from Soekris (Wi-Fi). In order to detect vendor-specific performance issues they decided to use two different sets of wireless NICs from Atheros and Broadcomm. In their study the influence of the following parameters was examined: LTE-U duty cycle, Wi-Fi and LTE TX power, LTE bandwidth, LTE central frequency (i.e. LTE and Wi-Fi spectrum overlap). Their main results can be summarized as follows:

- Wi-Fi throughput is inversely proportional to LTE duty cycle.
- Wi-Fi TX power has little impact on Wi-Fi throughput.
- The influence of LTE bandwidth and central frequency on Wi-Fi throughput depends very much on the vendor of the NIC card. As a consequence, more experimental research with physical devices from different vendors is strongly recommended.

The testbed in [7] consists of one LTE base station (eNodeB or eNB) and one user equipment (UE), one Wi-Fi access point and five other Wi-Fi nodes. Their Wi-Fi network was based on embedded PCs equipped with commodity wireless adapters.

The LTE nodes were based on desktop computers with Ettus USRP B210 RF front ends running the open-source driver UHD. An interesting detail is that they also used GNU Radio. The following parameters were subject of interest: duty cycle, Wi-Fi power settings Wi-Fi MCS (modulation and coding scheme) and packet size. The metrics measured were satisfied load in percent, total Wi-Fi throughput, Wi-Fi jitter and LTE packet loss. Their main findings can be summarized as follows:

- The duty cycle patterns are a main influence on achievable Wi-Fi throughput. Particularly, shorter duty cycles decrease jitter, which is important for real-time applications. On the other hand longer duty cycles offer superior throughput due to reduced overhead.
- LTE suppresses Wi-Fi transmissions if the TX power levels are comparable and no duty cycling is employed.
- If Wi-Fi TX power is increased, Wi-Fi load negatively impacts LTE throughput. There is no panacea strategy ensuring maximum Wi-Fi throughput operating under different MCSs and packet sizes. LTE performance is unaffected by Wi-Fi contention levels.

Both studies are similar to our work in so far that they use SDR with real hardware to experimentally evaluate inter-technology coexistence. However, the examined parameters of their studies are mostly related to power, frequency and duty-cycle, whereas we focus on CSMA/CA timing aspects.

ZigBee/Wi-Fi Coexistence In the MAC layer ZigBee uses the CSMA/CA protocol in nonbeacon-enabled mode or a mixture of CSMA/CA and TDMA in beacon-enabled mode. If upper layers detect that the throughput degrades below a certain threshold the MAC layer will be instructed to perform an energy scan through all available channels after which follows a switch to the channel with the lowest detected energy [9], [8]. The comprehensive study [9], which is based on theoretical analysis, simulation (using Matlab/Simulink) and measurement with real devices take an approach that differs from ours. Instead of relying on the CSMA/CA algorithm in contention situations they try to avoid sharing the same channel with Wi-Fi. They conclude that adhering to certain deployment rules or alternatively appropriate channel management guarantee good coexistence of Wi-Fi and ZigBee:

- A frequency offset of 8 MHz between the Wi-Fi and ZigBee channel central frequencies with a distance of 2 m between Wi-Fi and ZigBee nodes is always sufficient. In such a case adjacent channel interference is negligible.
- Alternatively a distance of 8 m between Wi-Fi and ZigBee nodes is always sufficient.
- If the former two rules are not applicable smart channel management can drastically reduce interference with Wi-Fi.

In [8] a SDR testbed with USRPs and GNU Radio is deployed to evaluate the influence of a proposed mechanism, namely cooperative busy tone, on the throughput

of ZigBee and Wi-Fi. The idea is that a separate ZigBee node schedules a busy tone whenever a transmission between nodes is desired to enhance the visibility of ZigBee to Wi-Fi nodes.

The ZigBee Alliance white paper [32] shows that ZigBee can coexist well with Wi-Fi in home networks if the Wi-Fi load is low. However, as Wi-Fi load increases to medium and high loads ZigBee throughput decreases severely in [33] and [34]. All three of these papers are based on measurements with physical devices, but none features SDR. Furthermore, the focus in these papers lies on different traffic patterns, a subject we only touch.

Bluetooth/Wi-Fi Coexistence Bluetooth is a coordination-based technology where a master device and up to seven active slave devices form a piconet using adaptive frequency hopping, which is a type of frequency hopping spread spectrum, which is a CDMA technique. With the pseudo-random frequency hopping scheme Bluetooth may interfere with Wi-Fi nodes. The simulation-based study [10], in contrast to our work, proposes two algorithms to avoid overlapping of Bluetooth with Wi-Fi in the time and the frequency domain, respectively, rather than evaluating the influence of parameter variation on standardized mechanisms. The key idea of the first algorithm is to adjust the Wi-Fi packet length to fit in between two Bluetooth packet transmissions. The second algorithm induces the Bluetooth master node to schedule data packets with appropriate durations to skip the frequencies of the hopping pattern that are expected to drop on the IEEE 802.11 band. The similarities of this study to our work is limited to examining coexistence mechanisms.

4

MEASUREMENT METHODOLOGY

This chapter describes methods and scenarios of the measurements we have taken. Firstly, the measurement testbed is discussed. Secondly, we define central terms to guard against misapprehensions. Thereafter, we give an overview of the MAC protocols we empirically evaluated, implemented as GNU Radio flowgraphs. Subsequently, measurement metrics which we use in chapter 5 to analyze the performance of the protocols are formally defined with reference to the flowgraphs. Thereafter, an overview of the semi-automatic measurement script system designed to automate, therefore accelerate the process of file system management, data processing and result plotting is given. Eventually, we discuss the quality norms of the measurements.

4.1 MEASUREMENT TESTBED

The setup consists of two USRP2s from Ettus Research and two USRP 2920s from National Instruments. The first two USRPs are programmed as receiver and sniffer, respectively, whereas the latter two as transmitters, as depicted in 4.1. Each USRP was connected to a gigabit switch through a LAN cable. The scripts running on the devices were launched from a local computer with the IP 134.130.223.151, which was remotely controlled from a laptop. Both transmitters sent their data to the single receiver. Hereafter, we call the node pair 10.0.0.9-10.0.0.6 link 1 and 10.0.0.3-10.0.0.6 link 2. Tables 4.1 and 4.2 contains other necessary configuration parameters to reproduce the measurement results.

4.2 MEASUREMENT PROTOCOLS AND SCENARIOS

Next, we define some central terms used throughout the remainder of the thesis and then present the measurement scenarios.

Traffic Saturation If not specified otherwise all transmitters are backlogged, i.e. we have *saturated* traffic. In that case the time between the generation of each packet is constant and well below the RTT. When we use the term *unsaturated* the time between packets generated by the `dummy_source` is exponentially distributed with $\frac{1}{\lambda} = 200ms$. These packets are then buffered in a `frame_buffer`. For single link scenarios this leads to Poisson-distributed traffic.

Measurement and Repetition In this thesis *measurement* refers to a period of 500 seconds comprising of five *repetitions* with a duration of 100 seconds each.

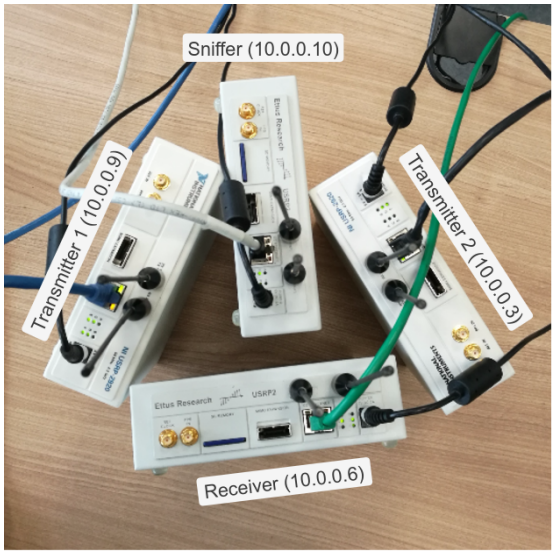


FIGURE 4.1: Photo of the measurement setup. The transmitters 10.0.0.3 and 10.0.0.9 transmit their data to the receiver 10.0.0.6. Their conversations are overheard by the sniffer 10.0.0.10.

Function	TX Gain	RX Gain	Source Address	Dest. Address	IP Address
Receiver	4 dB	10 dB	X	any	10.0.0.6
Sniffer	0	0	any	any	10.0.0.10
Transmitter 1	5 dB	0	Y	X	10.0.0.9
Transmitter 2	9 dB	0	Z	X	10.0.0.3

TABLE 4.1: Device-specific setup parameters.

Baseline and Coexistence Measurements When referring to the term *baseline* measurement we mean that only one of the two links was active. If both links were active we refer to a *coexistence* measurement. The baseline measurements serve two purposes: confirming that the devices work properly and for comparison with the coexistence measurements. A more detailed description on baseline measurements can be found in Section 4.6.

Measurement Scenarios A scenario is a combination of MAC protocols employed on the two links as depicted in Table 4.3. We distinguish between two types of scenarios. In "same MAC" scenarios the same MAC protocol is employed on both transmitters. In "different MAC" scenarios different MAC protocols are employed on the transmitters.

Pure ALOHA We implement the pure ALOHA protocol based on the theory in Section 2.1.5.1 as GNU Radio flowgraph depicted in Figure 4.3.2 and described in Section 4.3.2.

Layer	Parameter	Value	Comment
PHY Layer	MCS	QPSK	BPSK optional, unused
	Carrier frequency	450 MHz	420 MHz - 480 MHz work as well
	Sampling rate	400 k/s	
MAC Layer	Frame size	1000 bytes	max. supported by OS
	Payload size	837 bytes	
	Timeout	100 ms	RTT $\leq$68 ms
	CSMA CS threshold	0.001 PU	for RX/TX gains as in Table 4.1
	Max. retransmissions	6	

TABLE 4.2: General setup parameters.

CSMA/CA We implement CSMA/CA based on the theory in Section 2.1.5.4. The flowgraph is depicted in Figure 4.3.3 with the corresponding description found in Section 4.3.3. We are **not** featuring the optional IEEE 802.11 RTS/CTS exchange, realize DIFS and SIFS with `general_timers` and the backoff with the `backoff` blocks. In two of three measurement variants we set SIFS, DIFS and backoff slot (BO) times to different scaled versions of their values[1] prescribed in the IEEE 802.11g standard. We refer to these two variants as the high parameter values (DIFS $= 15$ ms, SIFS $= 3$ ms, BO $= 6$ ms) and the low parameter values (DIFS $= 5$ ms, SIFS $= 1$ ms, BO $= 2$ ms). The third variant are the medium parameter values based on the low parameter set but with DIFS $= 9$ ms. We employ the same $CW_{min} = 31$ and $CW_{max} = 1023$ as the IEEE 802.11g standard.

1-persistent CSMA For 1-persistent CSMA we use the same flowgraph as for CSMA/CA and set SIFS and backoff slot times to zero. In contrast to theoretical p-persistent CSMA we sense the channel for DIFS instead of a minimal number of samples. More accurately, we could describe the protocol as "1-persistent CSMA-like with fixed sensing duration", but for brevity's sake we refer to it as 1-persistent CSMA.

4.3 GNU RADIO FLOWGRAPHS

A GNU Radio flowgraph is a directed graph, each of whose vertices called blocks implements a certain functionality of the MAC protocol. The edges determine the direction of data flow as discussed in Section 2.2.2.

4.3.1 Receiver and Sniffer

Figure 4.3.1 shows the two-way handshake receiving logic. After frame integrity is checked by the `frame_check` block and the type is confirmed to be data frame an acknowledgment is generated by the `frame_type_check` block. The `frame_probe` blocks record the times when the frames reach certain positions in the flowgraph

[1]The values of DIFS, SIFS and BO in the IEEE 802.11g standard are DIFS $= 50\,\mu$s, SIFS $= 10\,\mu$s, BO $= 20\,\mu$s, $CW_{min} = 31$, $CW_{max} = 1023$ [35].

Scenario Type	Link 1	Link 2
Same MAC	ALOHA CSMA/CA (3 variants) 1-persistent CSMA	
Different MAC	ALOHA	CSMA/CA
	unsaturated ALOHA	CSMA/CA
	CSMA/CA	CSMA/CA
	1-persitent CSMA	unsaturated ALOHA
	1-persistent CSMA	CSMA/CA

TABLE 4.3: Measurement Scenarios.

representing the occurrence of events such as frame reception, passed or failed frame integrity check and more. Note that the address check is disabled so that the receiver may receive frames from any transmitter.

The sniffer (flowgraph in Figure 4.3.1) consists only of a single `frame_probe` block, which records detected power above noise level during the whole measurement. The sniffer provides valuable insight of what is actually going on in the channel from a "neutral" point of view - neutral in the sense of:

- A clear distinction between the transmitters can be made according to the received energy levels, since the sniffer is located between the transmitters and transmission gains were chosen accordingly.
- Sensing the channel is possible during the whole measurement time, because the sniffer is never sending.

In a nutshell, the sniffer is a valuable debugging and verification tool as described in more detail in section 4.4.

4.3.2 Pure ALOHA Transmitter

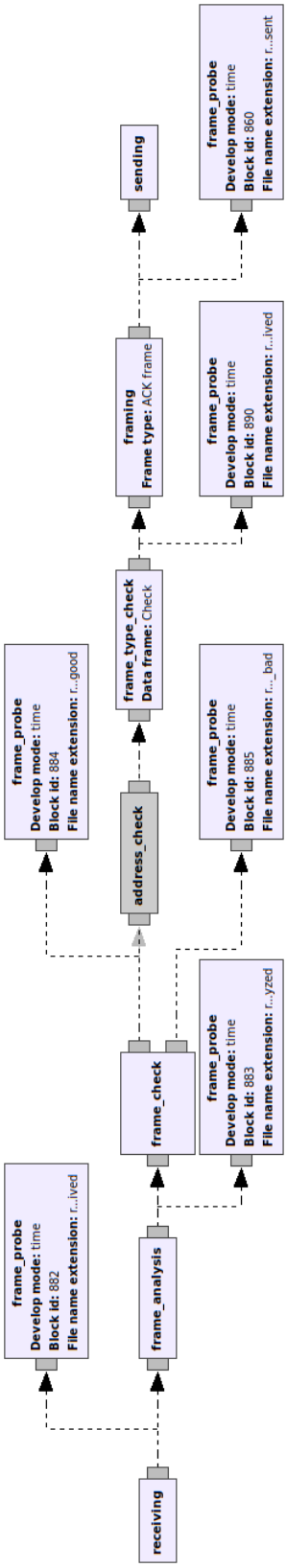
The flowgraph, whose discussion follows, is depicted in Figure 4.3.2. The `run` block enables us to start several transmitters exactly at the same time, which is useful if we execute the flowgraphs manually without the automated measurement scripts. Payload is generated in the `dummy_source` block, packed into a frame in the `framing` block and buffered in the `frame_buffer` block. The interval between generated frames is determined by a `general_timer` block, which we trigger either in constant or exponentially distributed intervals. Self-reception is prevented by shutting down the receiver when about to send a frame through the sending block. As soon as the data packet is sent off the `timeout` block receives a copy of the data frame. If the timeout timer is reset by a recevied ACK before it runs out the next frame in the buffer is dequeued, otherwise the data is forwarded to the `resend_check` block. If the maximum number of retransmissions, in our case 6 has not been reached a retransmission is issued, otherwise the frame is dropped without substitution.

4.3.3 CSMA Transmitter

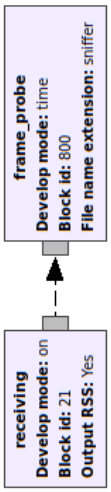
The CSMA transmitter (Figure 4.3.3) is based on the ALOHA transmitter, but features extra mechanisms as described in section 2.1.5.2, which is discussed now. The flowgraph aims at resembling IEEE 802.11 DCF and features CCA through thresholding in the `carrier_sensing` block. Despite the fact that this block has the feature of adaptively determining an appropriate carrier sensing threshold we chose a fixed value of 0.002 power units (PU) [2]. This choice was made to make sure that ALOHA transmission power levels were not confused with noise during the adaptive CSMA noise floor detection period.

DIFS and SIFS are realized through `general_timer` blocks with the respective values. The design, as depicted, does not feature the RTS/CTS exhange.

[2]Power unit is a linear-scale unit read out via the UHD driver.



(a) Receiver



(b) Sniffer

FIGURE 4.2: GRC Receiver Flowgraphs.

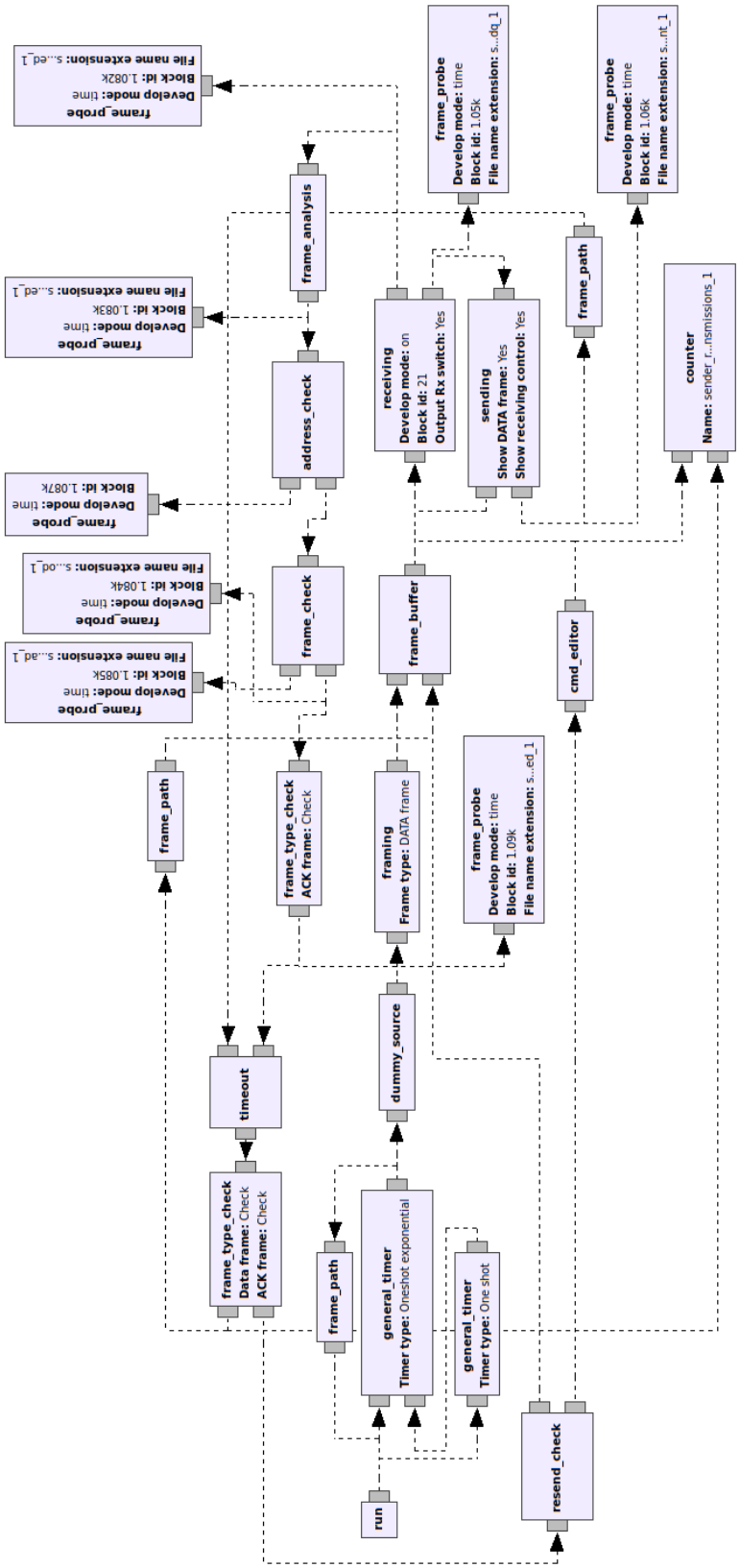


FIGURE 4.3: GRC Pure ALOHA Transmitter Flowgraph.

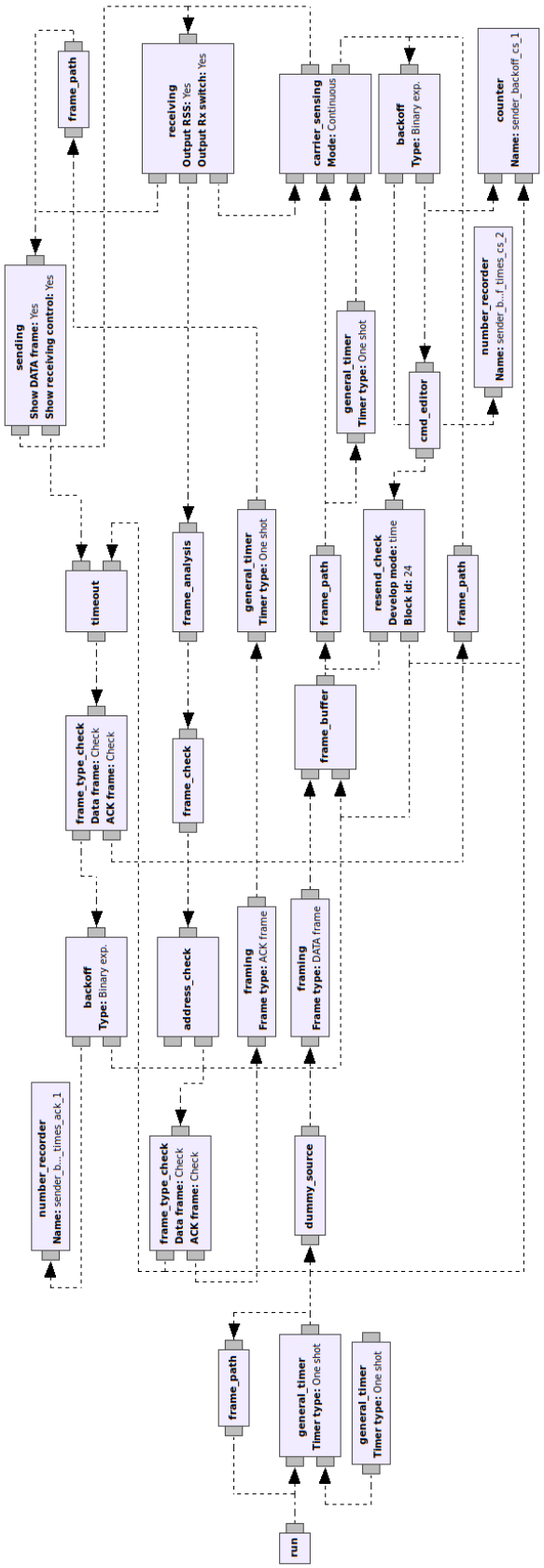


FIGURE 4.4: GRC CSMA Transmitter Flowgraph.

4.4 MEASUREMENT METRICS

All recorded metrics are defined in this section. Furthermore, we describe how the metrics were obtained and verified. All metrics were originally captured with at least one of the following blocks: `frame_probe`, `counter`, `number_recorder` and `time_probe` as depicted in figures 4.3.1, 4.3.2 and 4.3.3. In particular, we used the `frame_probe` block to record files with timestamps - and in the case of the sniffer also with energy levels - when frames reach positions in the flowgraph that are associated with certain events, such as data transmission or ACK reception at the transmitter. The `counter` block was used to count how often a certain frame was retransmitted and how often the `backoff` block was activated. The `number_recorder` was used to capture backoff times and the `time_probe` block to verify frame durations.

4.4.1 Throughput

We define throughput as the mean useful data (payload and headers disregarding retransmissions) transmission rate in the unit kbit/s. We obtain this metric simply by counting the number of ACKs received at the transmitter, multiply it with the frame length of 8 kbits and divide it by the measurement duration. The calculations are done in `throughput.py` making use of the CLI tool `wc` to count lines. Aggregate and single throughputs are our main metrics to judge a protocol's efficiency or how well a certain combination of protocols can coexist under different conditions, respectively.

4.4.2 Round-Trip Time

We define round-trip time (RTT) as the mean time from the buffer dequeuing a data frame until ACK reception. If the variable `rtt_mode` is set to `rtt`, then the calculation excludes retransmitted frames. If `rtt_mode` is set to `frame_delay` instead then retransmitted frames are taken into account. How we obtain these metrics is best explained with the code in Listing 4.1, where `ack_received_times` and `data_sent_times` contain the timestamps of the frames.

```
1  # pointer onto data frame which we use for frame delay calculation
2  data_pos = 0
3  for k,ack in enumerate(ack_received_times):
4      for l,data in enumerate(data_sent_times):
5          # go to 1st data frame that is sent after the respective ack
6          if data > ack:
7              if self.rtt_mode == "rtt":
8                  # the data frame before current position l
9                  # must be the frame the ack corresponds to
10                 rtt += [round(ack - data_sent_times[l-1],5)]
11             if self.rtt_mode == "frame_delay":
12                 # the pointer onto our frame delay reference
13                 # is used to calculate frame delay
14                 rtt += [round(ack - data_sent_times[data_pos], 5)]
15             # set new reference point for frame delay calculation
16             data_pos = l
```

```
17              # break loop to look at next ack frame!
18              break
```

LISTING 4.1: The method used in `rtt_alternative.py` to calculate RTT and frame delay

Another way to calculate the RTT is by recording the number of retransmissions of each data frame and subtracting the element with the correct offset in `data_sent_times` from each ACK reception time. We do not show any code (which can be found in `rtt.py`) here, because this has not been used to create any of the plots, although it has been verified to return the same results as the first method.

4.4.3 Packet Loss and Retransmissions per Frame

Obtaining both metrics involves data processed in `rtt.py`, which is why they are calculated there as well. Specifically, we make use of the lists containing the timestamps of ACKs and data frames as well as the number of retransmissions per frame. We define packet loss as $1 - \frac{n_{\text{ACKs}}}{n_{\text{data}}}$, where n_{ACKs} and n_{data} are the number of ACK packets received and data packets sent by the transmitter. Retransmissions per frame are obtained simply by recording them with a `counter` block, which is incremented whenever the `timeout` runs out and reset when an ACK is received.

4.4.4 Backoff Time

The script `backoff.py` sums up three different backoff times: Firstly, the backoff due to negative CCA, i.e. a busy channel. Secondly, we capture the backoff times after successful transmissions to give other nodes a chance to seize the channel. Lastly, we sum up the two to obtain the total backoff. The total backoff duration reflects the efficiency of the CSMA protocols in dependency on the parameters DIFS, SIFS and backoff slot length [3]

4.4.5 Packet Durations & Channel Occupation

The channel occupation chart as in Figure 5.7(c) provides an approximated logical view on the channel in the fashion of a Gantt chart. Blue patches represent data frames, red patches ACKs and black patches the reception of ACKs. The chart is only a (good) approximation of the channel occupation because the width of the patches are fixed and defined in `channel_occupation.py`. The duration of DATA and ACK frames was previously recorded with the `time_probe` blocks as the difference between the times when the frame was dequeued from the buffer and when it was received by the receiver. As expected, the frame durations were very stable. A data frame took 40 ms to be transmitted and an ACK frame took 7 ms. The variation of frame duration was in the range of 1-2 milliseconds for data frames and in the sub-millisecond range for ACK frames. Depending on the time limits chosen for the plot this may be well below the plot's resolution, which is why the time axis of such plot is limited to a range of a few seconds at most.

[3] E.g. for two CSMA transmitter it would be ideal to have both transmitters back off for around 50% of the transmission time to give each other a chance to transmit.

4.4.6 Channel Energy Level

The energy levels (measured in a linear-scale, non-negative power unit) over the channel observed by the sniffer (and processed in `sniffer.py`) help us to verify a multitude of metrics. We can verify frame durations, round-trip time, backoff time (for saturated traffic) and logical channel occupation[4]. Even collisions are clearly visible and which transmitter caused them. Throughput ratio among transmitters can easily be verified by representing data as CDF, e.g. if two identical MAC protocols run under identical circumstances then we expect a CDF with a step where the height of the "energy columns" to the left and right have equal height, i.e. both transmitters have sent an equal number of data packets.

4.5 MEASUREMENT SCRIPT SYSTEM

The elaborate script-system was an integral part of the work and enables future users to much more quickly gain results based on automation, since it is no longer necessary to manually execute flowgraphs, manage captured files, run data processing scripts, sync files with github. Instead everything is automatically done for them. The transmission of every frame and data processing step can be traced back with the log files. Furthermore, to accommodate the need for comparison, a retrospective evaluation of any set of measurements `belated_evaluation.py` was created. The user only needs to add a few lines to the script as shown in Listing 4.2.

```
measurement     = [714,715,728,646]
links           = [1,2,1,2]
boxplot_xticks  = [
    "CSMA\nDIFS=15ms\nSIFS=0ms\nBO=0ms\nLink 1\n Coexistence",
    "unsaturated ALOHA\n Link 2\n Coexistence",
    "CSMA\nDIFS=15ms\nSIFS=0ms\nBO=0ms\nLink 1\n Baseline",
    "unsaturated ALOHA\n Link 2\n Baseline"]
```

LISTING 4.2: Evaluation of measurements with `belated_evaluation.py`. In `links` we denote the link we used in the corresponding measurement (compare Figure 4.1).

We now discuss in detail how the script system works by reference to Figure 4.5. Starting with the user calling `measurement_n.sh`, where n is the ID of the link, general settings are "imported" from `measurement_n.conf`. If the user sets `remote_measurement` to 1 in the conf file then `remote_measurement_n.sh` synchronizes the files on the remote machine with the github repository, then executes `measurement_n.sh` remotely. Subsequently, `measurement.sh` for link n works on open "jobs" from the `jobs_open_n` directories and optionally puts them into the `jobs_done_n` directories after completion. In the job files important variables

[4]Frame durations can be verified by reading the time values from the x-axis. RTT and frame delay as per definition in Section 4.4.2 are obtained in the same way. Theoretically, for saturated traffic, we could add up all times where the energy level is zero to obtain backoff times. Logical channel occupation can easily be derived provided the energy level of each frame type is distinct and known.

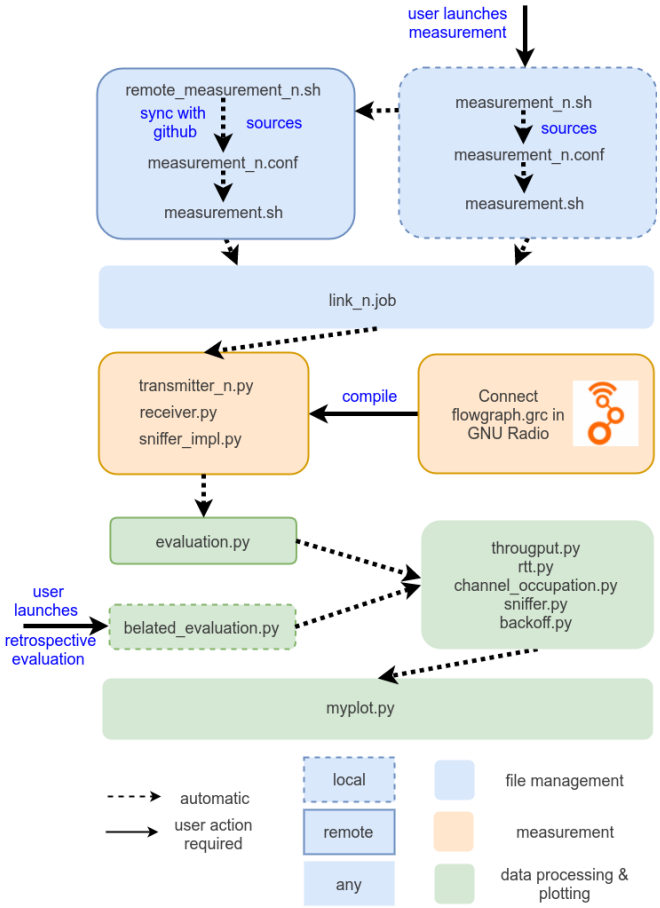


FIGURE 4.5: The three phase measurement script system.

such as duration, repetitions and flowgraph scripts of the measurement are defined. Any variable set in the conf file can be overwritten in the job file since they both are just exporting variables and were separated for semantic reasons only. After the measurement concluded `evaluation.py` coordinates data processing which eventually leads to plotting based on Matplotlib as defined in `myplot.py`.

4.6 QUALITY NORMS

Statistical Reliability As mentioned in Section 4.2, each measurement features five repetitions of 100 seconds duration. Compared to a single measurement of 500 seconds this has the downside that script and hardware initialization (about 1.1 seconds per repetition as can be seen in 4.7) has a negative impact on the accuracy of some metrics, particularly throughput, but was easier to implement as this way five data points are provided in a natural way. The statistical quality could be slightly improved by adding more repetitions and increasing the measurement time.

Data Processing Making use of modularity, multiple sets of test data were created for each data processing script (i.e. metric calculation and plotting scripts), provided

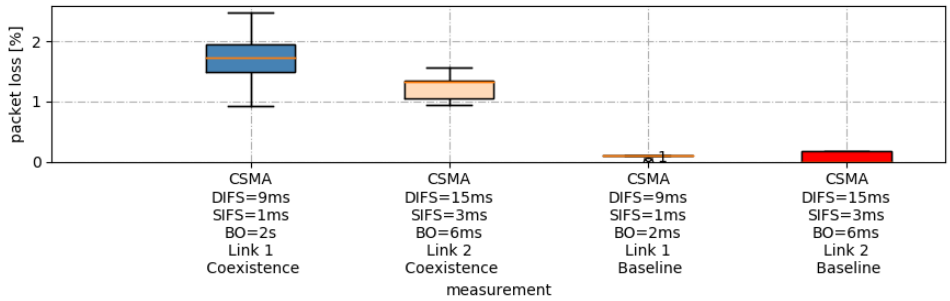
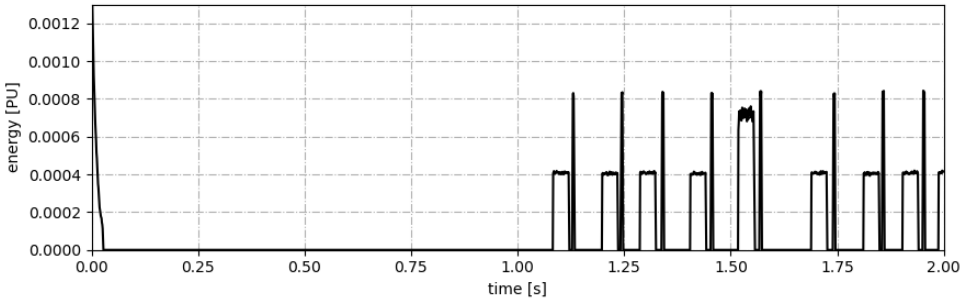


FIGURE 4.6: Packet loss plot. We only carry out coexistence measurements, if we have less than 0.2% mean packet loss in the baseline measurements.



(a) channel energy

FIGURE 4.7: Channel energy plot. The first 1.1 seconds of delay in each measurement repetition are caused by hardware initialization and script delays.

to and processed by the script and compared with manually calculated results in a similar fashion as GNU Radio quality assurance tests discussed in [36]. Intermediate data processing steps were printed to the console and logged in the log files where applicable. Additionally, experimental results were checked for plausibility.

Hardware Functionality For each device and protocol variation single link baseline measurements were carried out. Not only can we compare the results of device/protocol combinations to two link scenarios, but also assure that the devices are configured and work correctly. RX/TX gains, when necessary, were tweaked each time the hardware was restarted until no or very little ($\leq$ 0.2% mean) packet loss was observable in single links scenarios. Despite all efforts to find a combination of RX/TX gains and distances between nodes, where no packet loss would occur in single link configurations and at the same time the sniffer detects distinct energy levels for each packet type there still remains some degree of imperfection as depicted in Figure 4.6. This problem of packet loss is generally worse for link 1, because it is farther away from the receiver as is shown in Figure 4.1.

5

Measurement Results

In this chapter we present and discuss the measurement results for different combinations of MAC protocols employed on the two links. We first assess the measurement results where both senders employ the same MAC protocols. Subsequently, we do the same for several combinations of different MAC protocols.

5.1 Same MAC Protocol For Both Links

For the results presented tfhroughout this section, both transmitters executed identical flowgraphs. Generally, when both links use the same MAC protocol we expect to see comparable results for each link over a sufficiently longer period of time, although small variations are also expected due to statistical and hardware-related effects and inaccuracies.

5.1.1 ALOHA

For two links with saturated ALOHA traffic we expect zero aggregate throughput, since each and every packet collides. Figure 5.1 confirms this assumption, since both links have zero individual throughput. The corresponding packet loss of 100% is depicted in Figure 5.2. A reference value that can be read off Figure 5.1 is the throughput of a standalone saturated ALOHA link, which is about 130 kbps. This means that the combined throughput of multiple nodes in this channel with the same underlying PHY layer can never exceed 130 kbps and we can assess how well different protocols coexist and how much efficiently they make use of the channel by comparing their aggregated throughput to this value.

Furthermore, a physical view on the channel from the sniffer's perspective is provided in Figure 5.3(b). Due to the fact that the two transmissions of the senders are not

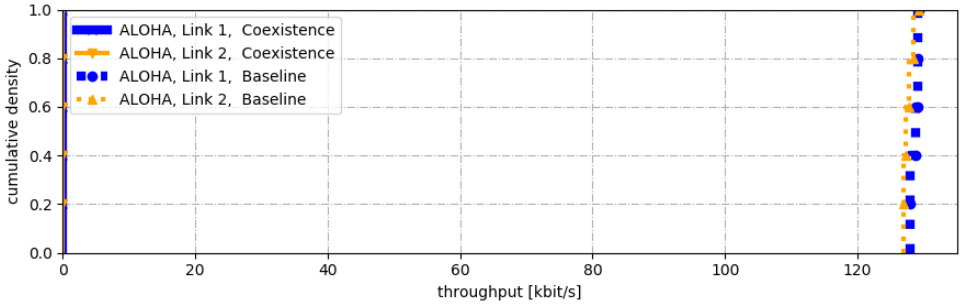


Figure 5.1: Throughput for two links with ALOHA.

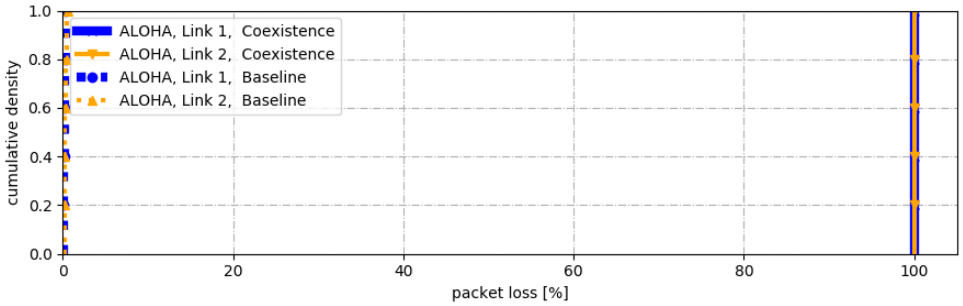
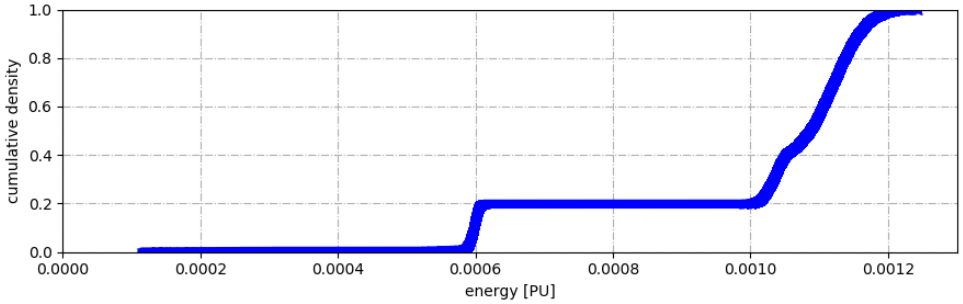
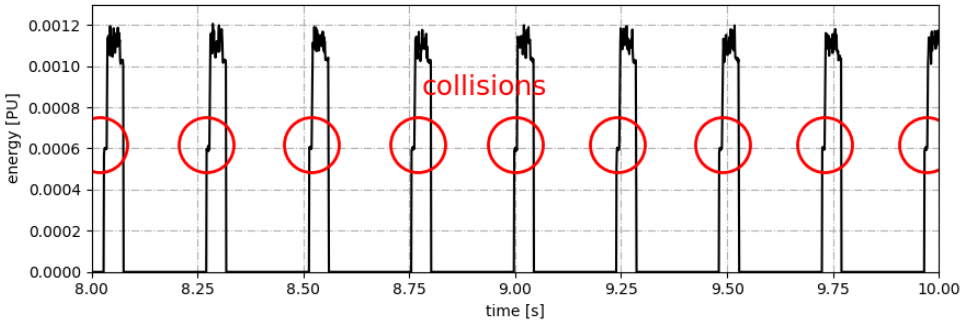


FIGURE 5.2: Packet loss for two links with ALOHA.



(a) channel energy CDF



(b) channel energy

FIGURE 5.3: Observed channel energy for two links with ALOHA.

completely overlapping we can see that we do not have a single sender with observed transmission energy level around 0.0011 PU, but instead two transmitters, where the observed energy level of the first transmitter is around 0.0006 PU, which the channel energy CDF in Figure 5.3(a) confirms. Note that the share of this particular energy (0.0006 PU) in the CDF is so high, because the transmission overlap does not stay constant throughout the whole measurement, but slightly varies with each repetition.

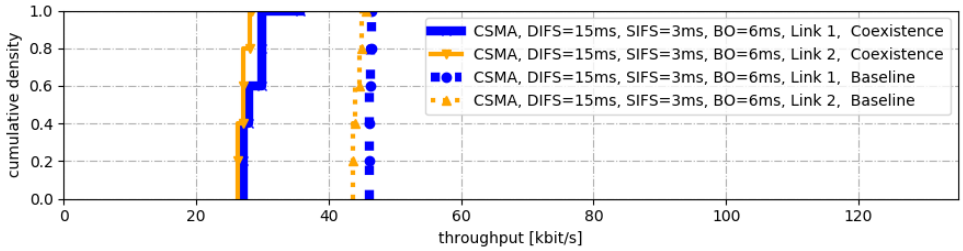


FIGURE 5.4: Throughput for two links with the high parameter CSMA/CA variant.

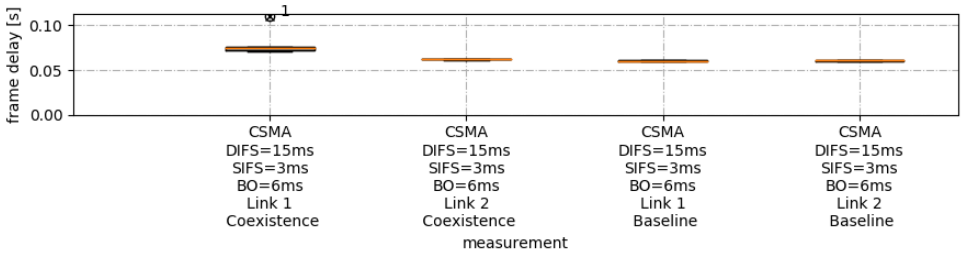


FIGURE 5.5: Frame delay for two links with the high parameter CSMA/CA variant.

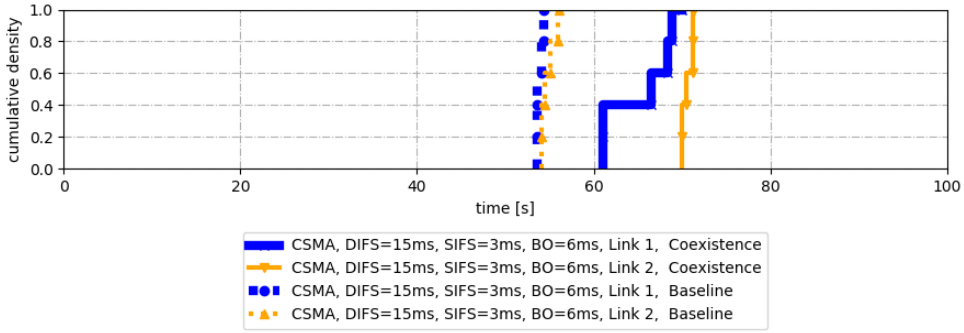
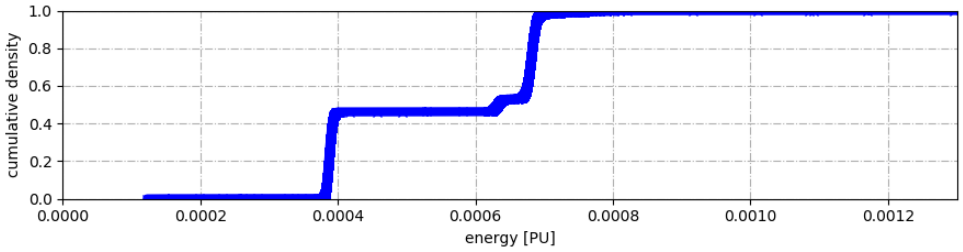


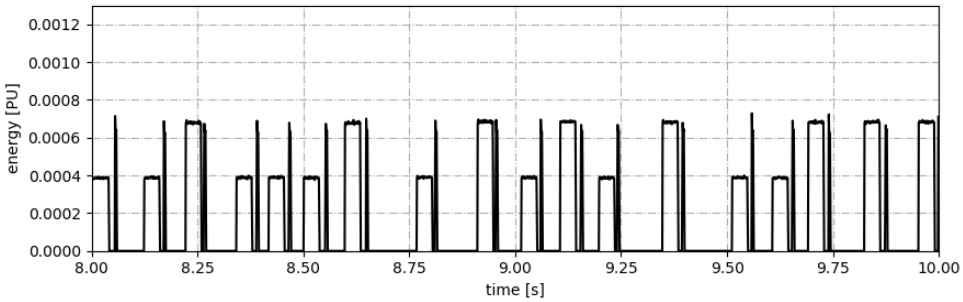
FIGURE 5.6: Backoff for two links with the high parameter CSMA/CA variant.

5.1.2 CSMA/CA With High Parameter Values

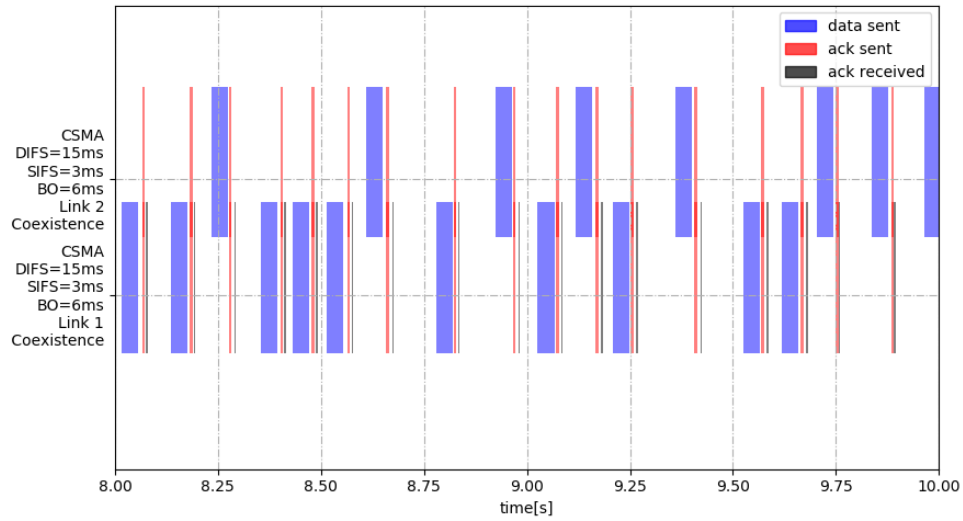
First of all, with "high parameter values" we mean we chose high values for DIFS, SIFS and backoff slot time (BO), from which we expected that they lead to good coexistence of the two links. In particular, we chose DIFS $= 15$ ms, SIFS $= 3$ ms, BO $= 6$ ms. Figure 5.4 shows that the throughput drops to roughly 60% ($\approx$ from 45 kbps to 27 kbps) when two links are active at the same time. Also note, that the aggregate throughout of both nodes, about 55 kbps, is less than half the channel capacity of 130 kbps. Figure 5.5 shows that the frame delay roughly stays the same, where the deviation of the first link comes from packet loss related to hardware problems as described in Section 4.6. Figures 5.7(a)(b) illustrate the same from the sniffer's point of view, as the even throughput among senders is reflected in Figure 5.7(b) with the two observed energy levels of the senders being 0.0004 PU and 0.0007 PU. 5.7a. Logical channel



(a) channel energy CDF



(b) channel energy



(c) logical channel occupation

FIGURE 5.7: Observed channel energy for two links with the high parameter CSMA/CA variant.

occupation which corresponds to the channel energy plot is depicted in Figure 5.7(c). Figure 5.6 shows the cumulative backoff times with values of around 70 s per 100 s repetition for the coexistence scenario. These values are much higher than 50 s (half the measurement time), indicating the potential to achieve higher throughput by reducing backoff times.

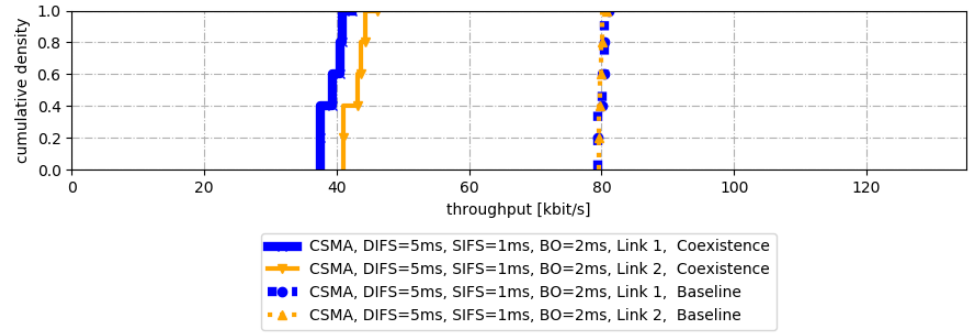


FIGURE 5.8: Throughput for two links with the low parameter CSMA/CA variant.

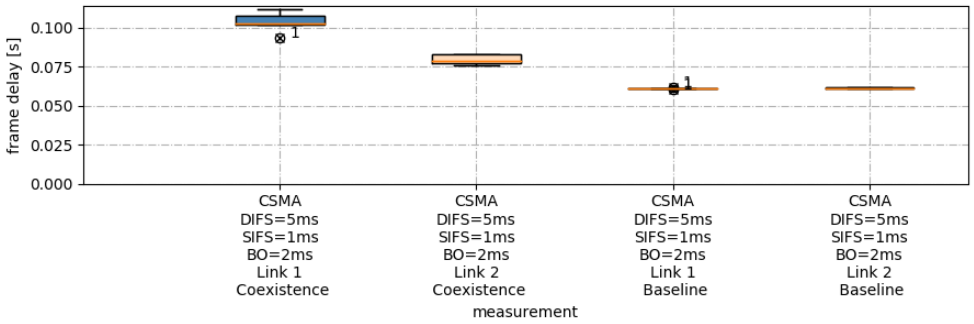


FIGURE 5.9: Frame delay for two links with the low parameter CSMA/CA variant.

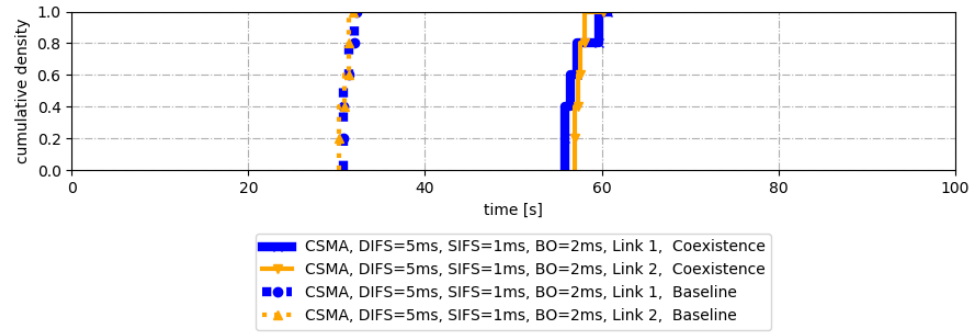


FIGURE 5.10: Backoff times for two links with the low parameter CSMA/CA variant.

5.1.3 CSMA/CA With Low Parameter Values

The next aspect we were interested in is to what extent we can scale down DIFS, SIFS and BO and still retain collision-free transmission. To this end, we reduced the values to DIFS $= 5\,\text{ms}$, SIFS $= 1\,\text{ms}$, BO $= 2\,\text{ms}$[1]. Reducing these values, especially BO increases the throughput, with $CW_{\text{min}} = 32 \cdot BO$ and uniformly distributed random

[1]We chose these values because they are close to the hardware capabilities in terms of time granularity as observed in practice.

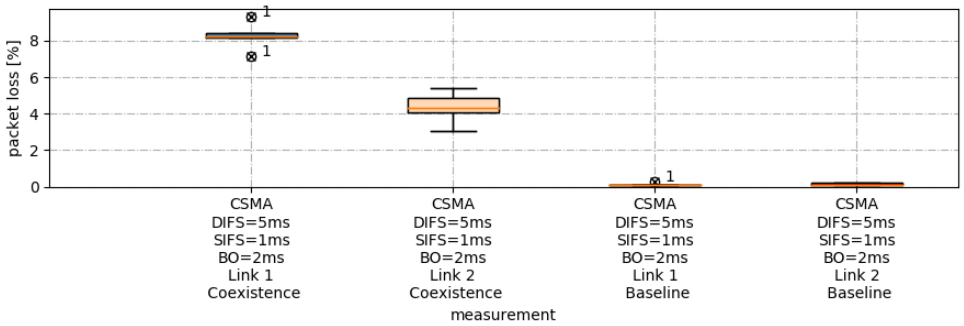
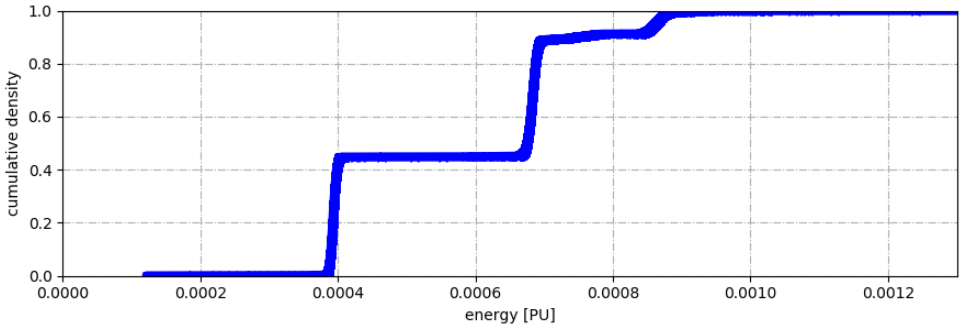


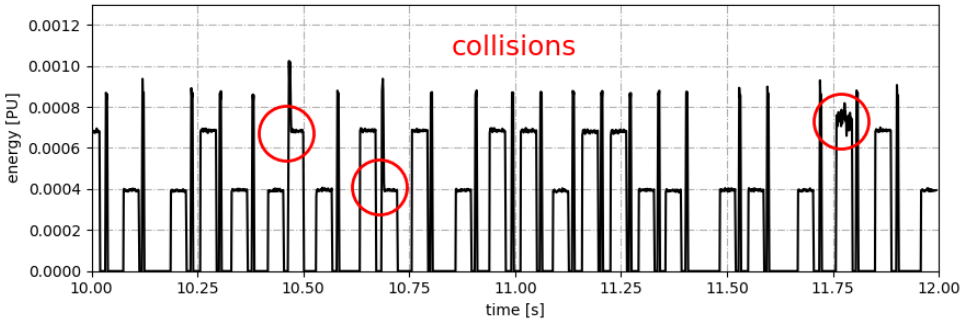
FIGURE 5.11: Packet loss for two links with the low parameter CSMA/CA variant.

choice of the backoff slot, we expect a mean delay of $16 \cdot BO = 32ms$ in the first backoff round, which is rather large compared to DIFS and SIFS.

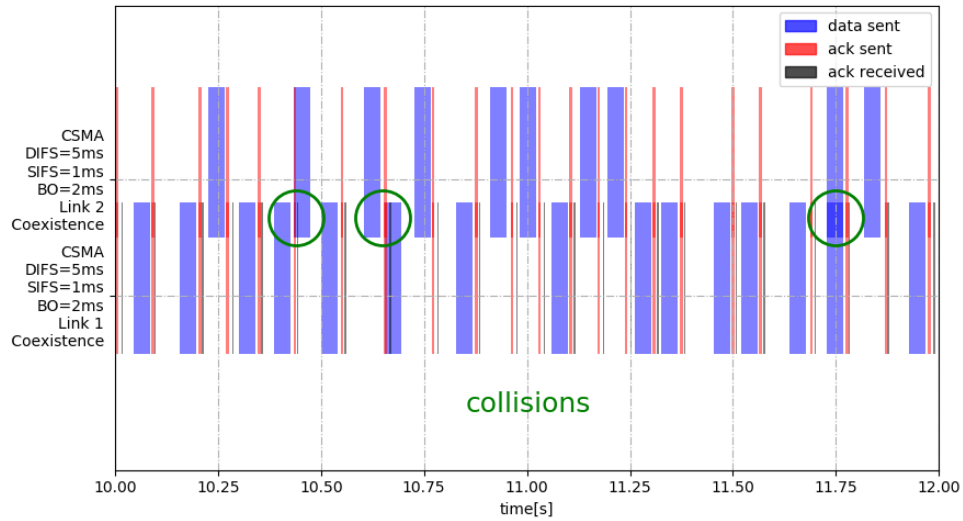
Indeed, comparing throughput using the high parameter values (Figure 5.4) with the low parameter values (Figure 5.8) yields about one and a half times the throughput ($\approx$ 40 kbps compared to 27 kbps) for cutting the backoff. Still, the aggregate throughput of about 80 kbps is much below the channel capacity of 130 kbps. A problem occurs with collisions of ACKs and consecutive data frames of the transmitter to whom the ACK was not destined caused by the low DIFS in conjunction with hardware delays. The first two collisions that occur in the time window depicted in Figure 5.12(b) are of this type. The third one is caused by both transmitting at the same time due to backoff slot alignment. The collisions lead to the increased frame delays of Figure 5.9, where the additional packet loss (Figure 5.11) of link 1 reflected in the 20ms increased frame delay compared to link 1 is caused by the lower signal-to-interference-plus-noise ratio (SINR) of link 1 compared to link 2.



(a) channel energy CDF



(b) channel energy



(c) logical channel occupation

FIGURE 5.12: Observed channel energy and logical occupation for two links with the low parameter CSMA/CA variant.

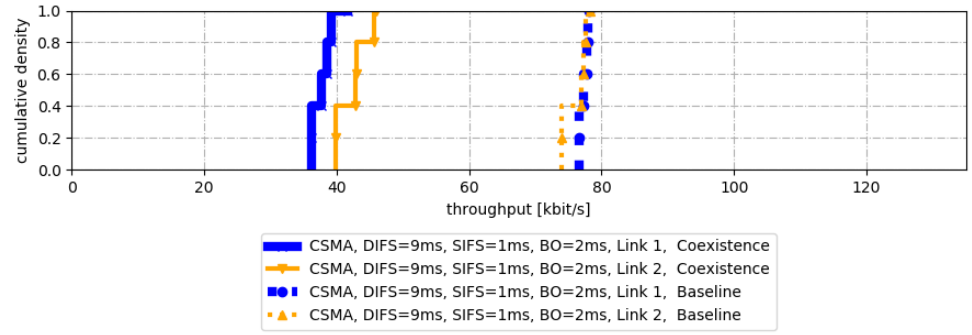


FIGURE 5.13: Throughput for two links with the medium parameter CSMA/CA variant.

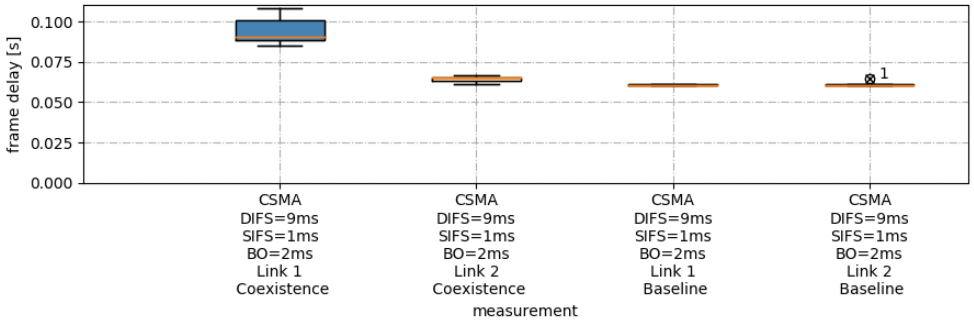


FIGURE 5.14: Frame delay for two links with the medium parameter CSMA/CA variant.

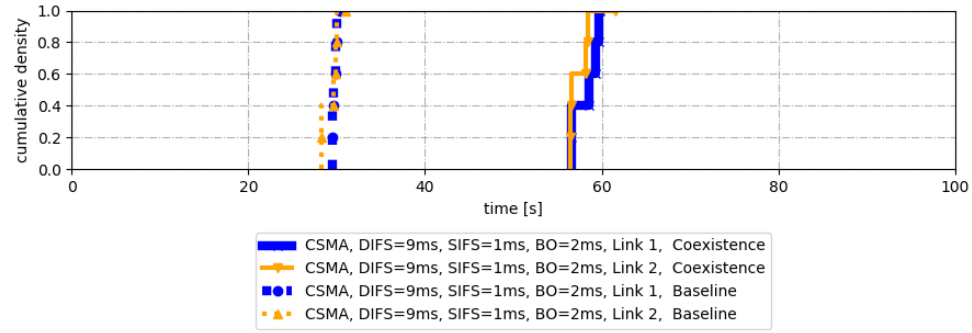
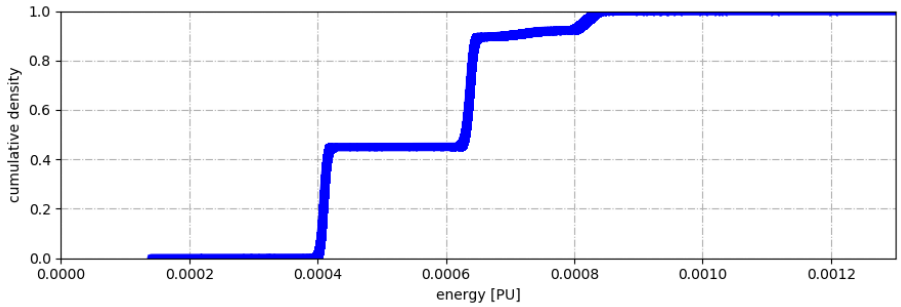


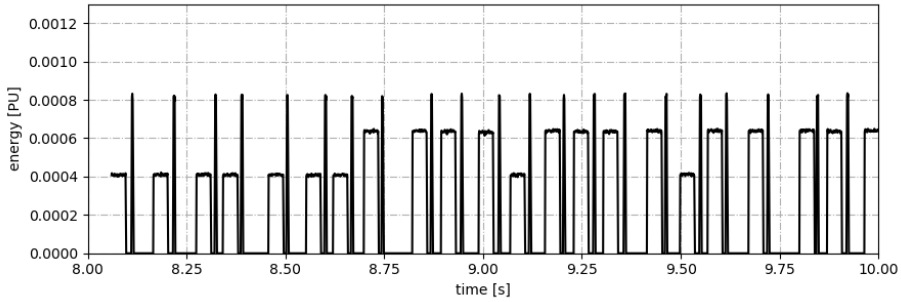
FIGURE 5.15: Backoff times for two links with the medium parameter CSMA/CA variant.

5.1.4 CSMA/CA With Medium Parameter Values

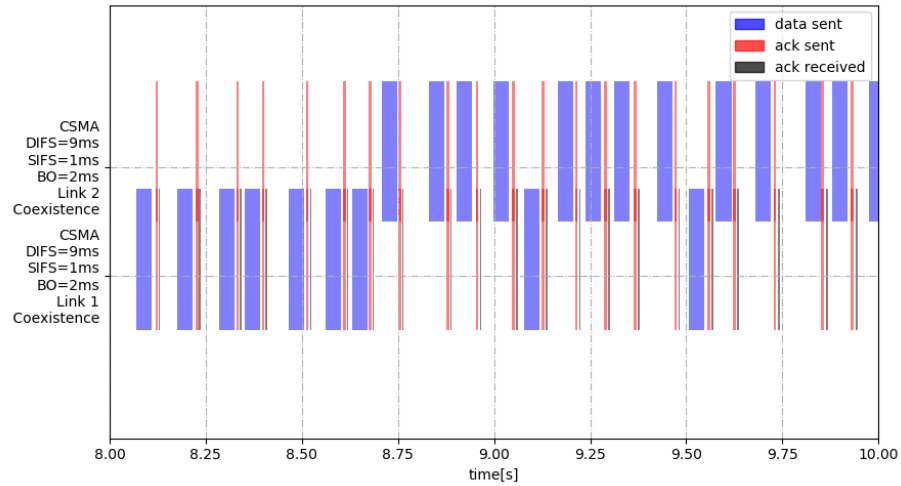
Due to the collisions of ACKs with the data packets of transmitter 1 as described in Section 5.1.3 we increased DIFS to 9 ms, kept SIFS to 1 ms and BO to 2 ms and indeed, as shown in Figure 5.14 the frame delay of link 2 is close to the baseline level as result of avoiding collisions. However, since collisions as a result of low DIFS occurred quite



(a) channel energy CDF



(b) channel energy



(c) logical channel occupation

FIGURE 5.16: Observed channel energy and logical occupation for two links with the medium parameter CSMA/CA variant.

infrequently in the last scenario, the metrics (Figures 5.13, 5.15, 5.16) roughly stay the same as in the previous scenario (Figures 5.8, 5.10, 5.12).

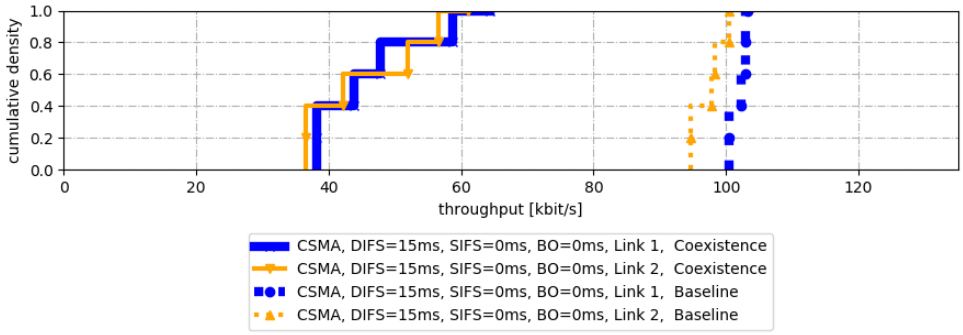


FIGURE 5.17: Throughput for two links with 1-persistent CSMA.

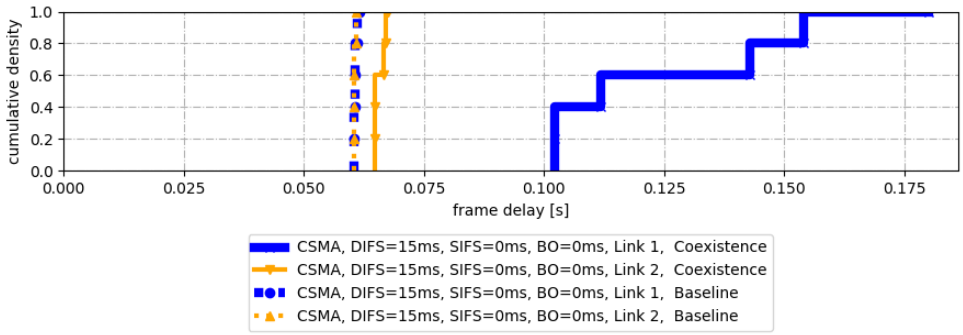


FIGURE 5.18: Frame delay for two links with 1-persistent CSMA.

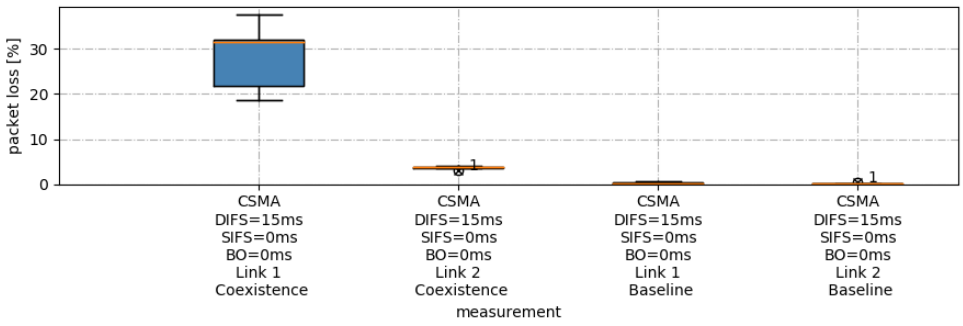


FIGURE 5.19: Packet loss for two links with 1-persistent CSMA.

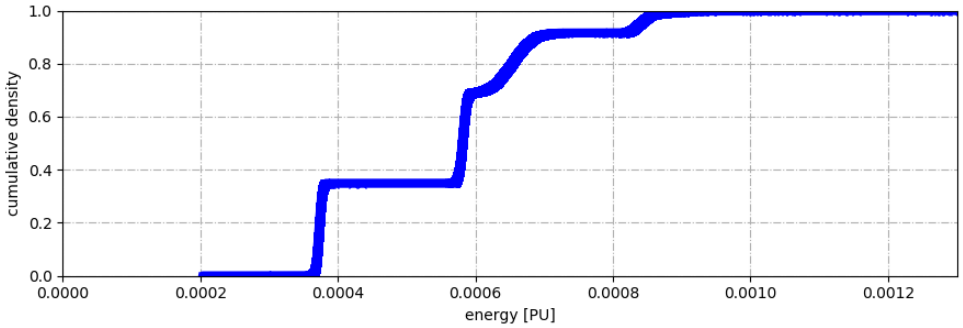
5.1.5 1-persistent CSMA

In the following experiment we examined if a fixed sensing duration DIFS is enough for harmonious coexistence. As can be seen in Figure 5.20(c) we encounter the problem described in Section 2.1.5.2, namely that both[2] transmitters try to seize the channel at the same time once a transmitter finished their transmission. If the time granularity of the system was finer, that is to say its timing accuracy was even higher none of

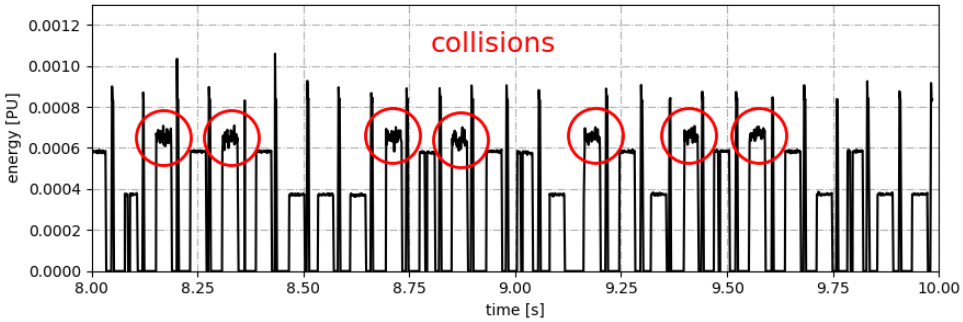
[2]There are actually three transmitters, since the receiver is transmitting ACKs.

the nodes would start to transmit slightly earlier, leading to even further deteriorated throughput than in Figure 5.17. The higher packet loss (Figure 5.19) and thus higher frame delay (Figure 5.18) of link 1 compared to link 2 can be elucidated by the fact that the SINR of transmitter 2 is bigger than the SINR of transmitter 1[3] as can be surmised from Figure 5.20(b). The extra bend in the channel energy CDF (Figure 5.20(a)) roughly from 0.0006 to 0.0007 PU is a consequence of the interference which is also very visible in Figures 5.20(b)(c).

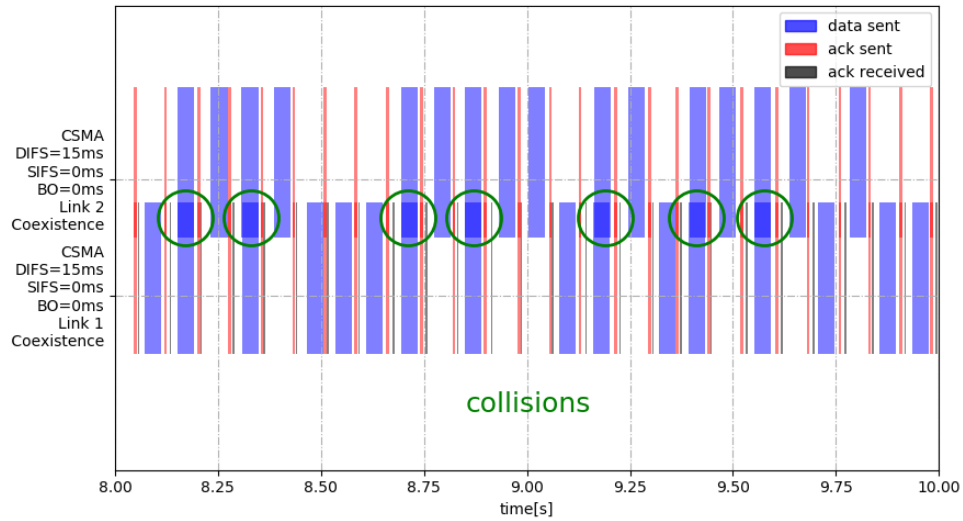
[3] Also, the SINR of the receiver's ACK signal is higher than all others.



(a) channel energy CDF



(b) channel energy



(c) logical channel occupation

FIGURE 5.20: Observed channel energy and logical occupation for two links with 1-persistent CSMA.

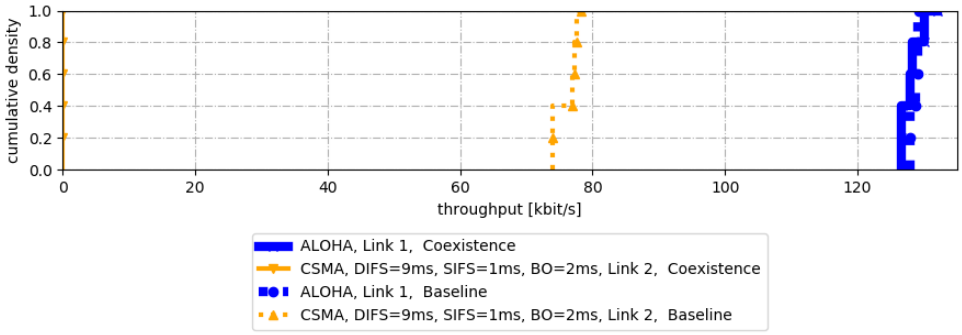


FIGURE 5.21: Throughput for one link with ALOHA and one link with the medium parameter CSMA/CA variant.

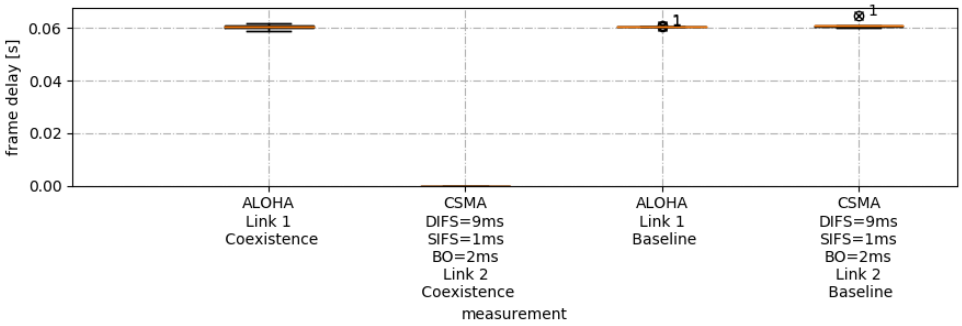


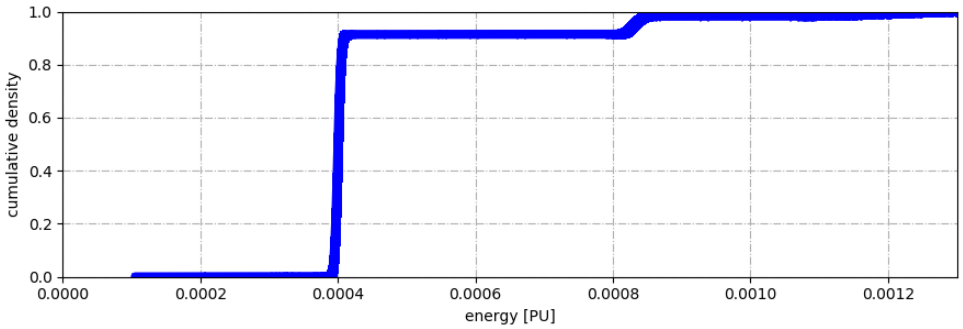
FIGURE 5.22: Frame delay for one link with ALOHA and one link with the medium parameter CSMA/CA variant.

5.2 DIFFERENT MAC PROTOCOLS FOR BOTH LINKS

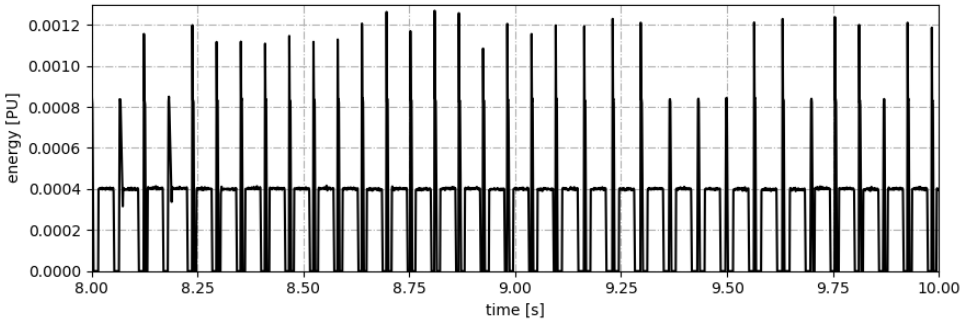
5.2.1 ALOHA and CSMA/CA

In this experiment we aimed at an experimental confirmation that saturated ALOHA traffic would cause CSMA/CA to stay silent during the whole measurement time. This result is confirmed by Figures 5.21 through 5.23. The throughput and frame delay[4] (Figures 5.21 and 5.22) of CSMA/CA are zero, whereas for saturated ALOHA node throughput and frame delay almost perfectly match the baseline values. The channel energy plots in Figures 5.23(b) and 5.23(a) show that only link 1 and the receiver are transmitting during a time window of 2 s, which also is generally true for the whole measurement duration.

[4]The frame delay is zero, because no frame has ever been sent by the CSMA/CA node.



(a) channel energy CDF



(b) channel energy

FIGURE 5.23: Observed channel energy for one link with ALOHA and one link with the medium parameter CSMA/CA variant.

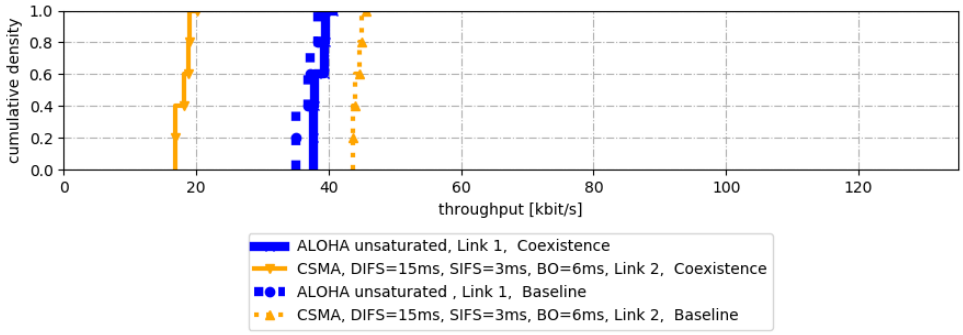


FIGURE 5.24: Throughput for one link with unsaturated ALOHA and one link with the high parameter CSMA/CA variant.

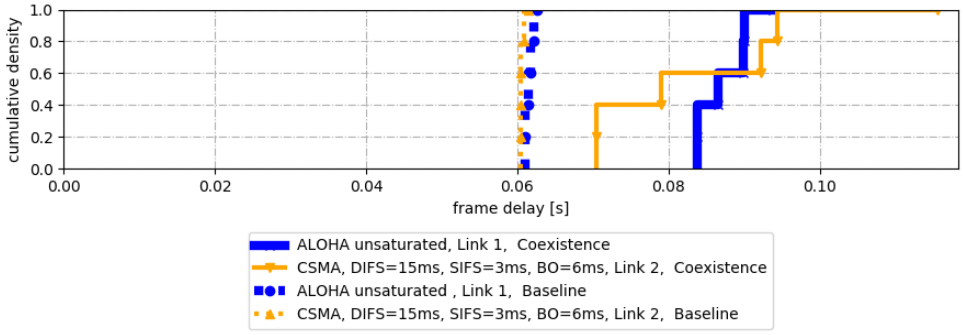


FIGURE 5.25: Frame delay for one link with unsaturated ALOHA and one link with the high parameter CSMA/CA variant.

5.2.2 *Unsaturated ALOHA and CSMA/CA*

The next experiment investigates whether relatively low load unsaturated[5] ALOHA can coexist with CSMA/CA. The throughput of the CSMA/CA transmitter is reduced to $\frac{18\,\text{kbps}}{45\,\text{kbps}} = 40\%$ of the baseline value as depicted in Figure 5.24, while the ALOHA throughput approximately remains the same.

Only if the CSMA/CA node transmits a frame before the ALOHA node a collision can occur, which happens during seconds 8-10 of the measurement as shown in Figure 5.27(c) from a logical point of view or in Figure 5.27(b) from a physical point of view. With that in mind, we can explain why the frame delay of CSMA/CA varies much more than the frame delay of ALOHA (Figure 5.25). The number of ALOHA packets generated during data frame transmission time (or any other period of time) is Poisson-distributed and thus the number of collisions, whereas the likelihood of collision from the point of view of an ALOHA frame is dependent on the backoff

[5]We still refer to exponentially distributed time between each packet with $\frac{1}{\lambda} = 200ms$, which gives us roughly $G_{\text{ALOHA,unsat}} \approx \frac{38\text{kbps}}{130\text{kbps}} \approx 0.3$, where 38 kbps is the baseline throughput of unsaturated ALOHA and 130 kbps the baseline throughput of saturated ALOHA. We can use this approximation, because the offered channel load of ALOHA is independent from other transmitters and saturated ALOHA approximatively consumes the whole channel capacity.

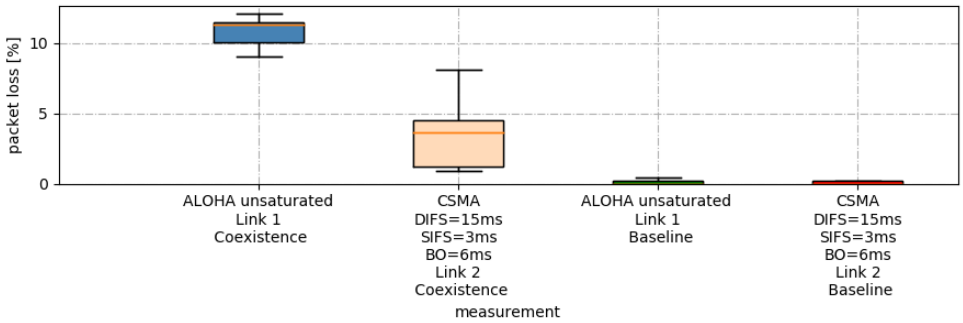
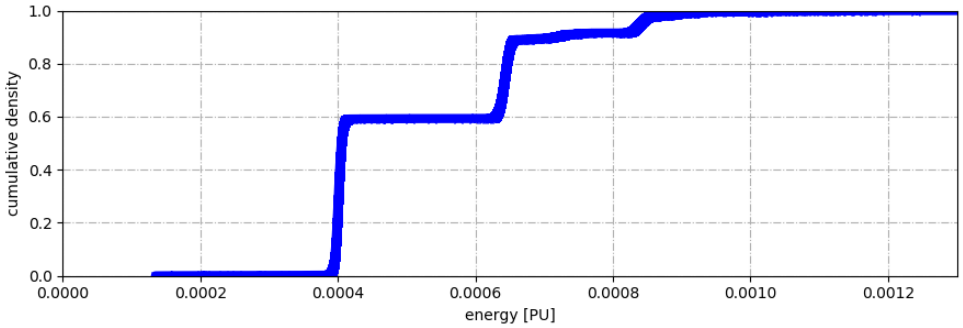
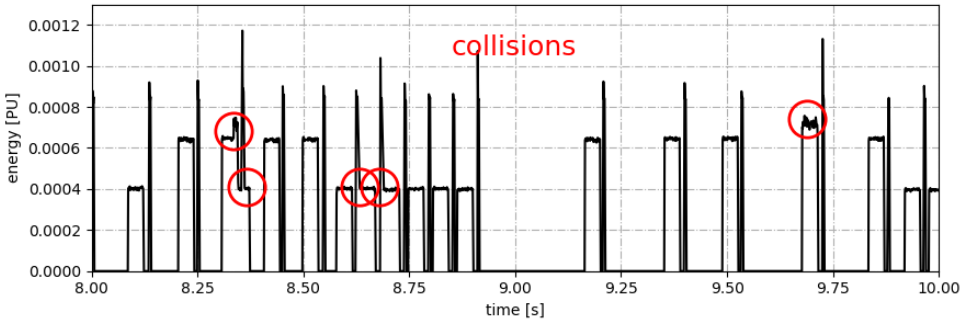


FIGURE 5.26: Packet loss for one link with unsaturated ALOHA and one link with the high parameter CSMA/CA variant.

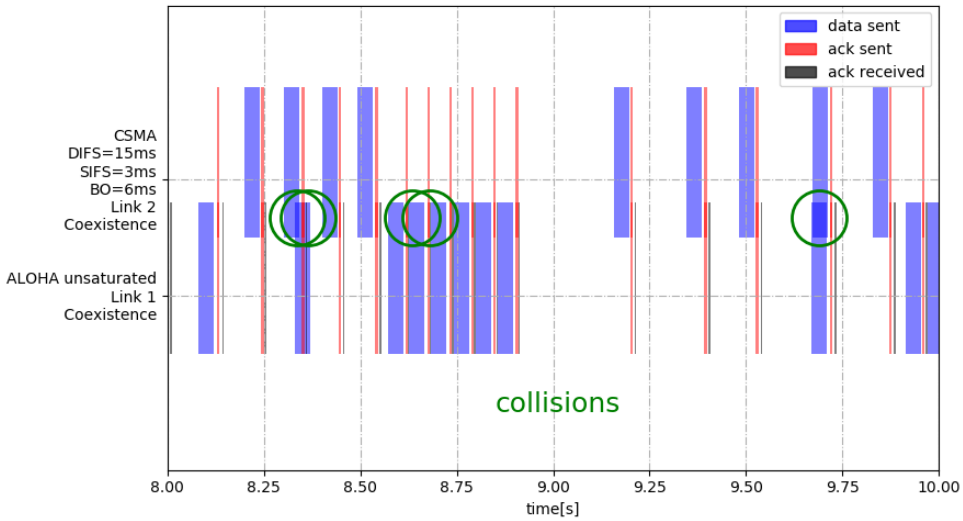
which in our case is uniformly distributed. The same explanation also applies for the differing variances in packet loss as depicted in Figure 5.26, whereas the differing values are because ALOHA recklessly pushes its packets into channel, while CSMA/CA does not interfere with ALOHA packets when it senses energy in the channel.



(a) channel energy CDF



(b) channel energy



(c) logical channel occupation

FIGURE 5.27: Observed channel energy and logical occupation for one link with unsaturated ALOHA and one link with the high parameter CSMA/CA variant.

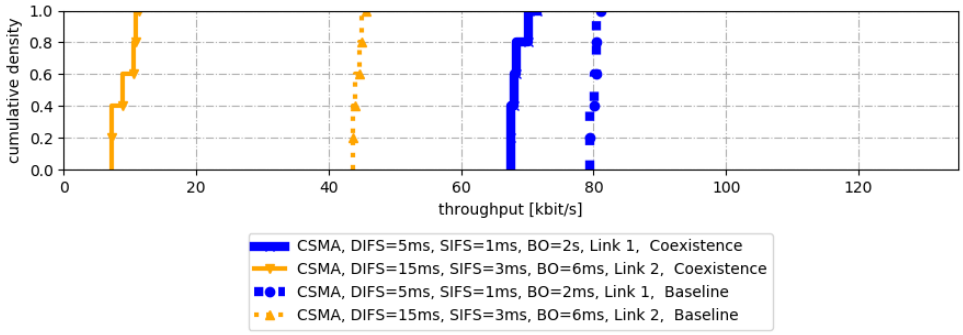


FIGURE 5.28: Throughput for one link with the low parameter CSMA/CA variant and one link with the high parameter CSMA/CA variant.

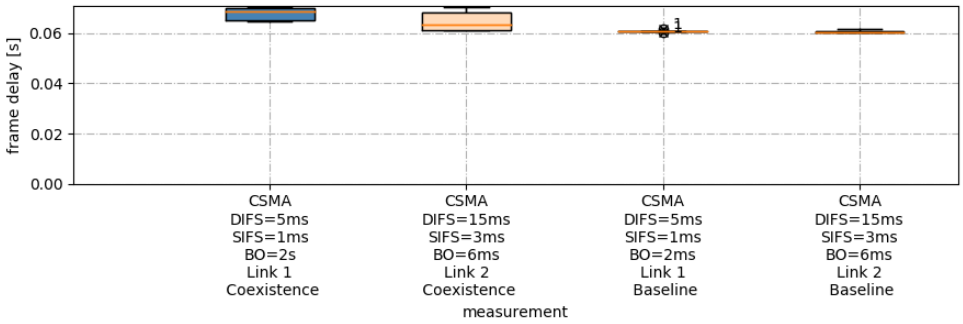


FIGURE 5.29: Frame delay for one link with the low parameter CSMA/CA variant and one link with the high parameter CSMA/CA variant.

5.2.3 *Two Variants of CSMA/CA*

The goal of our next experiment is to examine how CSMA/CA with different parameter values behave in the same channel. Link 1 uses the low parameter values, whereas link 2 uses the high parameter values. The throughput of link 2 as depicted in Figure 5.28 drops to one seventh of the baseline, whereas it only drops by 20% for link 1. The reason for this is that with reduced DIFS and BO the chance to grab the channel increases as is shown in Figures 5.31(a)(b)(c). The frame delay increases only marginally as shown in Figure 5.29, which is due to the packet loss depicted in Figure 5.30. The mean packet loss is for link 2 is a little below the expected value of about 1.2%, whereas for link 1 it is above that value, which is because the SINR of link 2 is higher than for link 1. The expected packet loss comprises of two components. The first component is the baseline packet loss for this measurement, which is around 0.2%. The second component is the chance that both transmitters choose the same time to transmit their packet when they sensed the channel idle, which amounts to $\frac{1}{CW_{\min}} \cdot \frac{\mathrm{BO}_1}{\mathrm{BO}_2} = \frac{1}{96} \approx 1.0\%$. An idea to reduce the chance of collisions due to this phenomenon is to configure the links to have backoff slot durations that are mutually prime, i.e. have no common divisor, which probably only works if the duration of a backoff slot is big compared to the time granularity of the whole system.

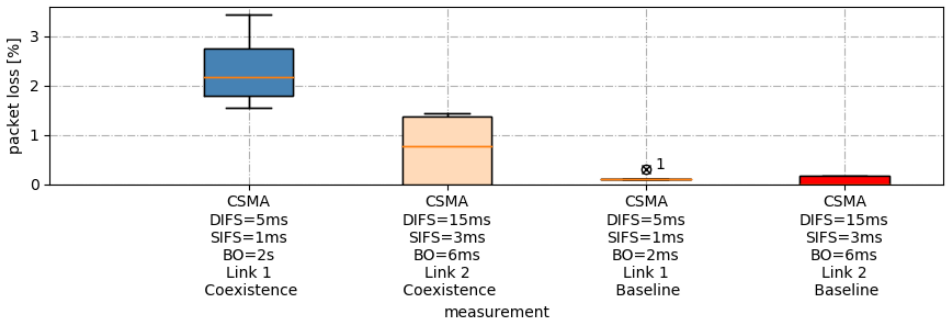
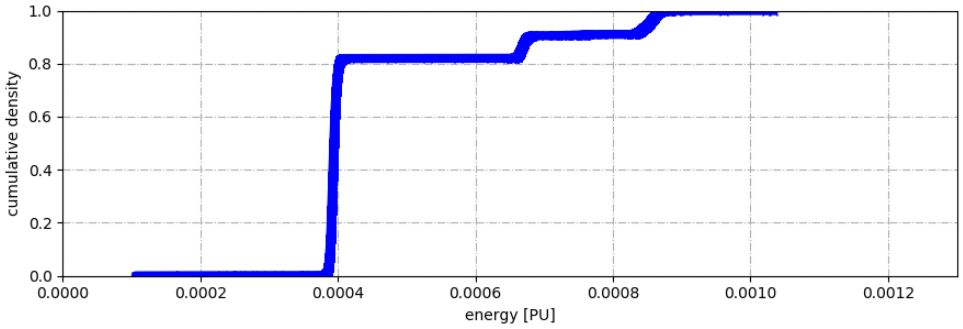
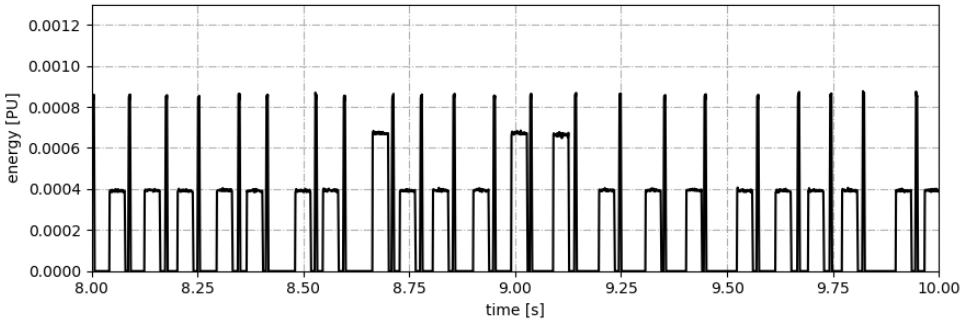


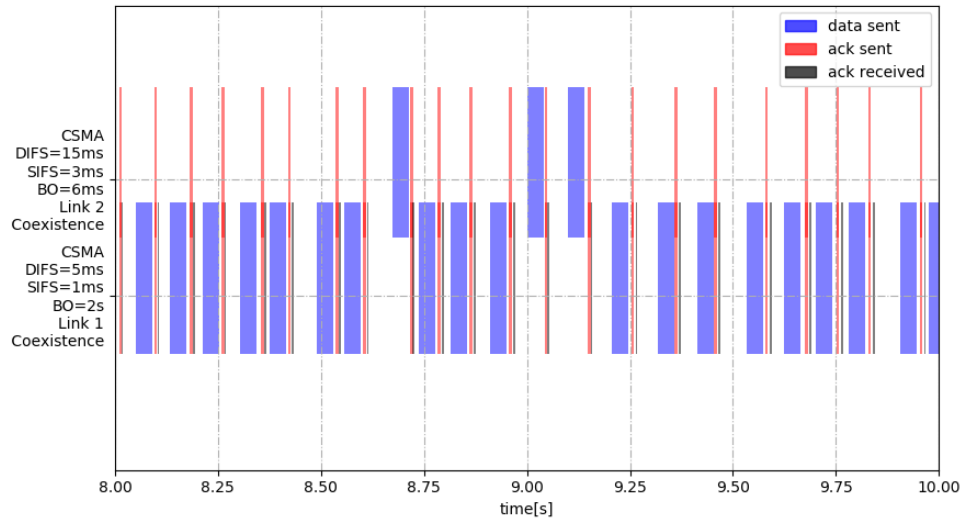
FIGURE 5.30: Packet loss for one link with the low parameter CSMA/CA variant and one link with the high parameter CSMA/CA variant.



(a) channel energy CDF



(b) channel energy



(c) logical channel occupation

FIGURE 5.31: Observed channel energy and logical occupation for one link with the low parameter CSMA/CA variant and one link with the high parameter CSMA/CA variant.

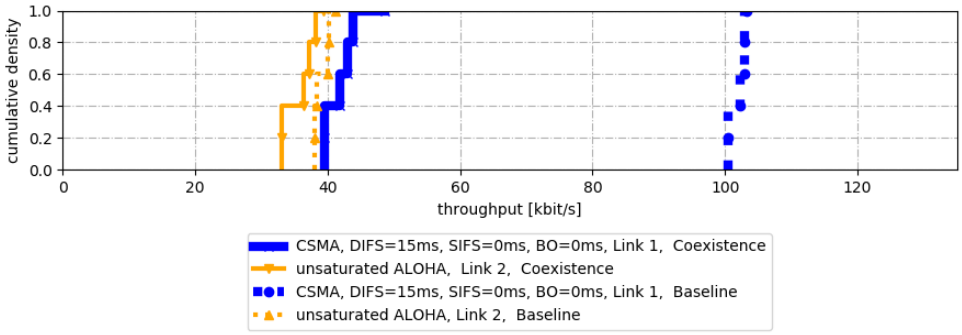


FIGURE 5.32: Throughput for one link with 1-persistent CSMA and one link with unsaturated ALOHA.

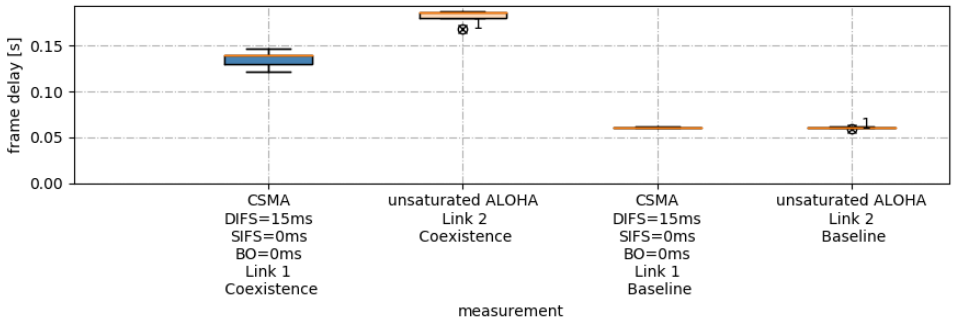


FIGURE 5.33: Frame delay for one link with 1-persistent CSMA and one link with unsaturated ALOHA.

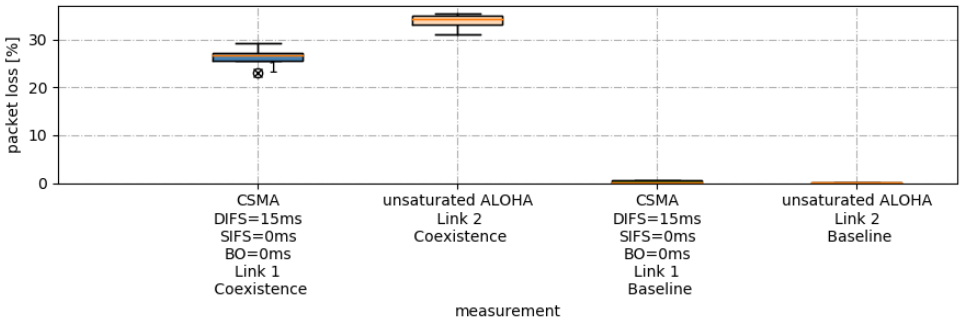
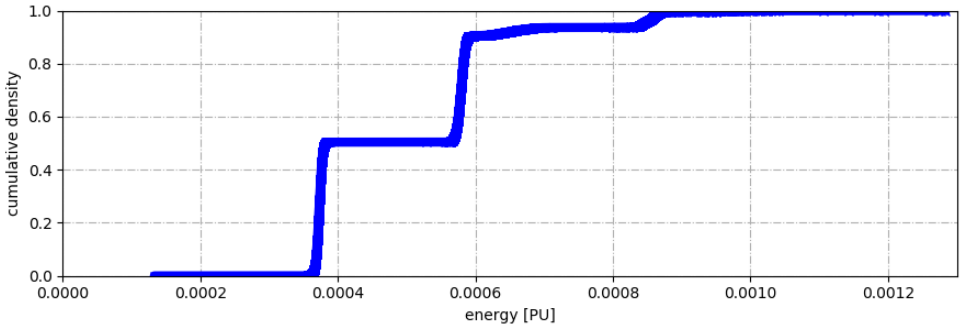


FIGURE 5.34: Packet loss for one link with 1-persistent CSMA and one link with unsaturated ALOHA.

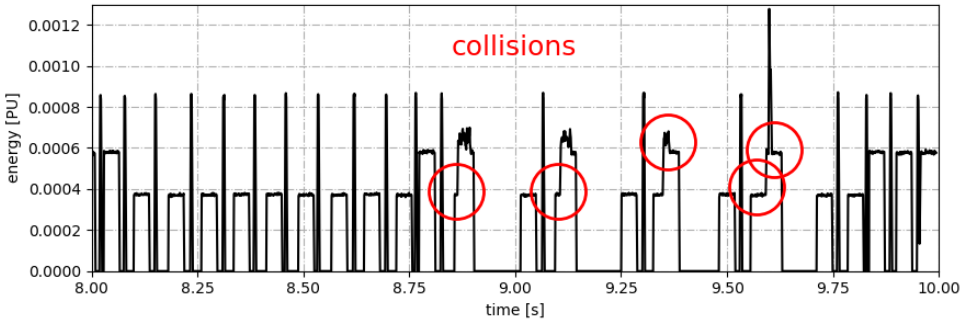
5.2.4 1-persistent CSMA and unsaturated ALOHA

If there is a link that sends saturated ALOHA traffic through the channel another link will have zero throughput as shown in Sections 5.1.1 and 5.2.1, which is why we do not consider scenarios with saturated ALOHA traffic anymore. We now discuss a scenario where link 1 uses 1-persistent CSMA and link 2 *unsaturated* ALOHA. In the

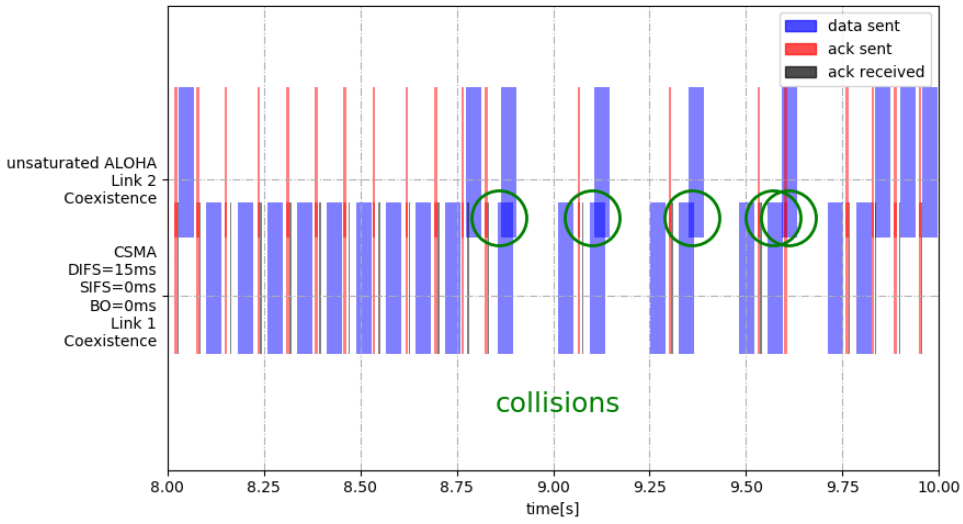
given scenario it does not make much sense for the CSMA/CA transmitter to back off when the channel is sensed busy either, due to fact that the ALOHA node does not use the LBT mechanism, thus "giving it the chance to transmit" is waste of time as it transmits whenever it wants anyway. Thus, removing the backoff in CSMA/CA promises higher throughput. For this reason we now compare this experiment with the one in Section 5.2.2. The assumption of increased throughput is correct as the comparison between Figures 5.32 and 5.24 confirms that removing backoff more than doubles CSMA throughput ($\approx$ 18 kbps compared to 40kbps). It would however, make sense to back off after the reception of an ACK to give the ALOHA node a chance to send a packet. The lack of this backoff explains the increased ALOHA mean packet loss ($\approx 34\%$ total, $\approx 300\%$ increase; Figure 5.34), which is still comparatively low due to the high SINR of the ALOHA node. A representative excerpt of the logical channel occupation is given in Figure 5.35(c), where from seconds 8.2 to 9.7 only one of five ALOHA frames around does not collide with a CSMA packet (around second 8.8). The energy plot in Figure 5.35(b) offers a more detailed physical view the channel, where the peak energy level of colliding data packets is not much higher than the energy level of a successful ALOHA transmission. In conjunction with the energy CDF in Figure 5.35(a), which is taking the whole measurement duration into account and has only a small bend around 0.0006 PU we conclude that if we decrease the TX power of the ALOHA node or use a less robust MCS, such as 64-QAM, ALOHA packet loss would drastically increase.



(a) channel energy CDF



(b) channel energy



(c) logical channel occupation

FIGURE 5.35: Observed channel energy and logical occupation for one link with 1-persistent CSMA and one link with unsaturated ALOHA.

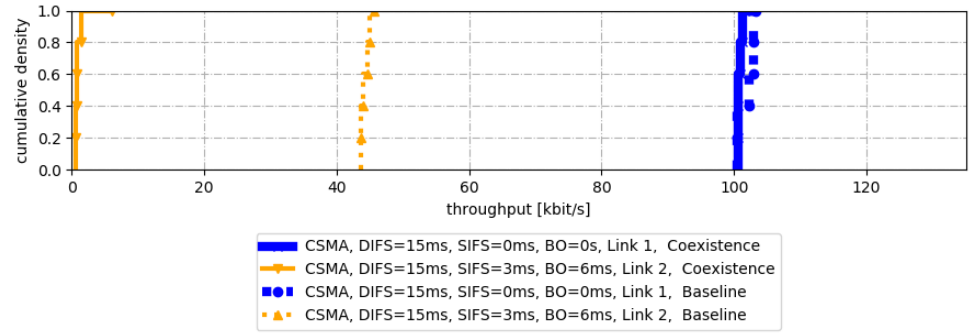


FIGURE 5.36: Throughput for one link with 1-persistent CSMA and one link with the high parameter CSMA/CA variant.

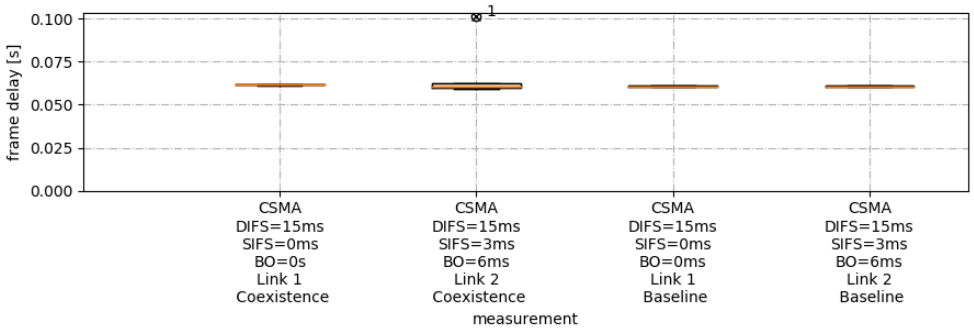
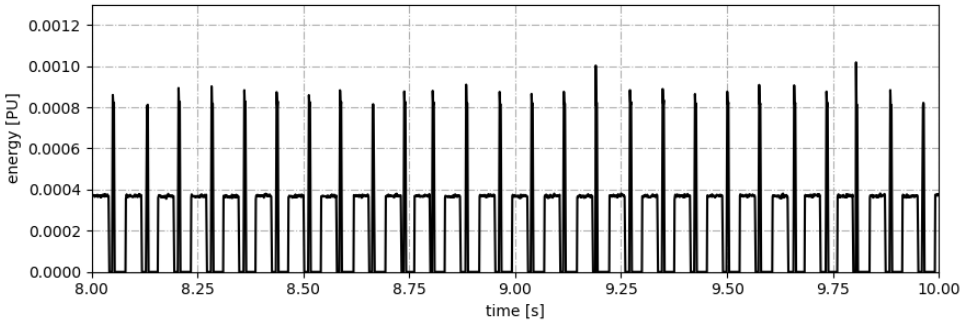


FIGURE 5.37: Frame delay for one link with 1-persistent CSMA and one link with the high parameter CSMA/CA variant.

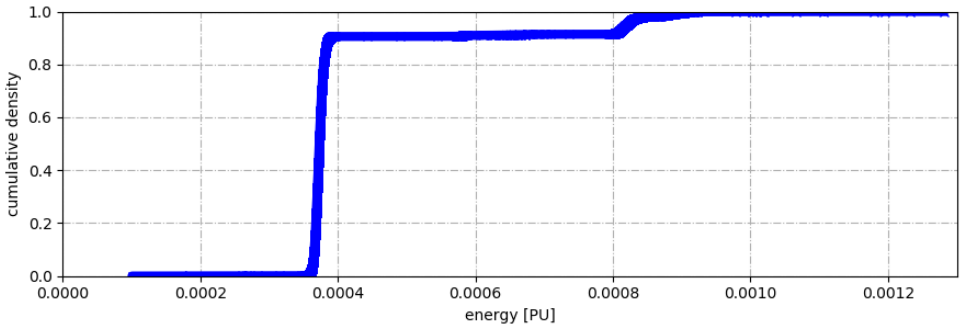
5.2.5 1-persistent CSMA and CSMA/CA

In our last experiment we show that the greedy 1-persistent CSMA approach starves other CSMA/CA links. From the perspective of a CSMA/CA transmitter it does not matter whether it shares a channel with a saturated 1-persistent CSMA transmitter or saturated ALOHA transmitter (Section 5.2.1). We can virtually identify no difference in the metrics (Figures 5.37 and 5.35(a)(b) compared to the corresponding Figures in Section 5.2.1) between those combinations except for the reduced throughput 5.38(b) due to the addition of channel sensing with the duration of DIFS compared to ALOHA. Only if we decreased the offered load of 1-persistent CSMA we could see any and better coexistence[6] with CSMA/CA at all.

[6] compared to ALOHA due to the LBT mechanism that stops the 1-persistent CSMA transmitter from interfering with ongoing transmissions



(a) channel energy



(b) channel energy CDF

FIGURE 5.38: Observed channel energy and logical occupation for one link with 1-persistent CSMA and one link with the high parameter CSMA/CA variant.

6

Conclusions and Future Work

In this chapter, we summarize the most important conclusions from the measurement results and give an outlook on what we think should be subject of future endeavors.

We showed that two nodes in symmetrical measurement setups, i.e. using the same MAC protocols with the same parameters in a shared channel, perform similarly in terms of throughput, frame delay and backoff times, where applicable. We showed that two backlogged pure ALOHA nodes recklessly push their packets into the channel resulting in zero throughput. Next, we showed that two CSMA/CA transmitters with DIFS, SIFS and BO values scaled up from the IEEE 802.11g standard coexist very well with almost no collisions and similar throughput. In an effort to increase throughput, we scaled the values of DIFS, SIFS and BO down and found out that we cannot arbitrarily reduce them without collisions other than those resulting from the two nodes by chance transmitting at the same time, due to the limited system time granularity caused by hardware delays. Further optimizing the parameter values we found that DIFS $= 9\,\text{ms}$, SIFS $= 1\,\text{ms}$, BO $= 2\,\text{ms}$ is still a stable combination. In the last experiment with the same MAC protocols for both links we proved that removing the backoff mechanism leads to the typical 1-persistent CSMA behavior of multiple transmitters starting to transmit at the same time, leading to collision and thus high frame delays and high packet loss.

Furthermore, we examined the coexistence of transmitters employing different MAC protocols. Pro forma we have shown that a single saturated ALOHA transmitter is sufficient to lock out other transmitters from the channel. Consecutively, we showed that even low loads of ALOHA traffic are quite detrimental to the performance of a CSMA/CA transmitter. The next experiment showed that in a scenario with two CSMA/CA transmitters the backoff slot duration, provided DIFS is rather small compared to the mean backoff, which is the case in our experiments, is the decisive factor for throughput distribution among links. Subsequently, we tried to improve on the throughput of CSMA in the unsaturated ALOHA combined with CSMA/CA scenario by replacing the CSMA/CA transmitter with a 1-persistent CSMA transmitter, i.e. as a main measure remove the backoff. As a result, the 1-persistent CSMA throughput doubled, whereas the packet loss of ALOHA trippled to about 35%. The last experiment shows that from the point of view of a CSMA/CA transmitter it does not make a difference whether there is another saturated 1-persistent CSMA or saturated ALOHA transmitter occupying the shared channel.

Eventually, we want to give an outlook on future work proceeding from questions left undiscussed in this thesis. Going from our last experiment as a starting point it would be interesting to see how much better 1-persistent CSMA fares compared to ALOHA in lower load situations concerning coexistence with CSMA/CA. More important and probably more practical questions are how the backoff mechanism can be

enhanced to adaptively accommodate different traffic situations without introducing uneconomical complexity. One idea is to modify the CW growth by taking the average CW of a limited number of previous transmissions into account or to use a different growth scheme, e.g. linear instead of binary exponential. Other ideas are using different and/or adaptive backoff slot durations for different nodes in the network or changing minimum contention windows all in order to further reduce the chance that multiple nodes start sending at exact same time, while simultaneously keeping unnecessary waiting times at bay.

Moreover, mechanisms of other MAC approaches, such as duty cycles, preamble sampling, piggybacking, as well as combinations of different mechanisms should be taken into consideration. Duty cycling as used in LTE-U or various MAC protocols for WSN seems particularly promising.

A completely different approach is to study the effect of transmission power, which may adaptively be varied to limit the range of a transmitter in order to allow a greater number of transmissions in the environment. On the other hand TX power could be adaptively increased if a node has priority traffic, allowing collisions to send traffic through a channel that is otherwise saturated.

A

Bash and Python Scripts

This appendix aims at giving insight in selected scripts used in the three phases of the measuring, data processing and plotting process. The basic principle, however, is depicted in section 4.5. Minor edits were made for format and aesthetic reasons.

```
1  echo "remote_measurement is set to "$remote_measurement"."
2
3  function setup_remote_connection
4  {
5    reset
6    sshpass -p "inets" ssh -$remote_flags $remote_user@$remote_ip
       "bash -s" < remote_measurement_$link.sh
7  }
8
9  function prepare_measurement
10 {
11     reset
12     measurement_counter=0
13     ## let's make sure all the directories exist
14     printf "\nchecking if paths exists...\n"
15
16     #let's first make absolutely sure the raw data source path
       exists
17     if [ ! -d $raw_data_source_path ];
18       then
19         mkdir -p $raw_data_source_path
20         echo $raw_data_source_path" created."
21       else
22         rm -r $raw_data_source_path/*
23     fi
24
25     if [ -d $plot_directory_path ];
26       then
27         echo $plot_directory_path" already existed!"
28         cd $plot_directory_path
29         # create measurement directory
30         while [ -d $measurement_counter ]; do
31             measurement_counter=$(($measurement_counter+1))
32         done
33         export measurement_counter;
34     fi
35
36     if [ -d $log_path ];
```

```
37        then
38          echo $log_path" already existed!"
39        else
40          mkdir -p $log_path
41          echo $log_path" directory created."
42      fi
43
44      mkdir -p $plot_directory_path/$measurement_counter
45      echo $plot_directory_path/$measurement_counter" directory
        created."
46
47      mkdir -p $data_source_path/$measurement_counter
48      echo $data_source_path/$measurement_counter" directory created."
49
50      mkdir -p $jobs_open_path
51      mkdir -p $jobs_done_path
52
53      ## let's check if measurement script is defined
54      # if $measurement_scripts undefined:
55      # go through directory and list all python files
56      if [ -z ${measurement_scripts+x} ];
57        then
58          echo  "no measurement scripts set,
59                going through files inside of $locate_base_path."
60          echo "please add a the full path of one of the files to
        \$scritps."
61          locate -r "$locate_base_path" | grep "\.py$"
62          echo "terminated. ding dong"
63          exit -1
64      fi
65
66      printf "\n"
67  }
68
69  function measure
70  {
71    local prematurely_aborted=0
72
73    for ((x = 1 ; x <= $measurement_repetitions ; x += 1)); do
74
75      # get pid to later kill it
76      for i in "${measurement_scripts[@]}"
77      do
78        python $measurement_script_path/$i &
79      done
80
81      for ((y = $timer ; y > 0 ; y -= 1)); do
82        echo "measurement $x/$measurement_repetitions complete in $y
        second(s)."
83        if [ $check_if_prematurely_aborted -eq 1 ];
84          then
85            if $(ps -p ${measurement_scripts_pid[*]}) | grep
        ${measurement_scripts_pid[*]};
```

```
86            then
87              :
88            else
89              prematurely_aborted=1
90              echo  "Scripts were killed prematurely. Measurement
    may be incomplete."
91              break
92          fi
93      fi
94      sleep 1
95    done
96
97    kill $(jobs -p)
98
99    # save this measurement's data to special folder
100   mkdir -p $data_source_path/$measurement_counter/$x
101   echo  "measurement $x raw data directory created
    $data_source_path/$measurement_counter/$x/."
102
103   echo $raw_data_source_path
104   echo $(ls $raw_data_source_path | egrep "*_$link.txt")
105
106   cd $raw_data_source_path
107   mv -v $(ls | egrep "*_$link.txt")
    $data_source_path/$measurement_counter/$x/
108   cp -v $(ls | egrep "sniffer")
    $data_source_path/$measurement_counter/$x/
109   if [ "$receiver_mode" == "single" ];
110     then
111       cp -v $(ls | egrep "receiver")
    $data_source_path/$measurement_counter/$x/
112   fi
113   echo  "measurement $x raw data moved to
    $data_source_path/$measurement_counter/$x/."
114   printf "\n"
115   if [ $prematurely_aborted -eq 1 ];
116     then
117       if [ $plot_if_prematurely_aborted -eq 0 ];
118         then
119           echo "plotting if measurement prematurely aborted set
    to false."
120           echo "terminated."
121           exit -1
122       fi
123   fi
124
125 done
126
127 #exit remote connection
128 if [ $remote_measurement -eq 1 ]; then
129   echo "remote_measurement is set to "$remote_measurement"."
130   exit
131 fi
```

```
132 }
133
134 function plot
135 {
136   ## plot the results
137   echo "now processing results..."
138
139   # call the plotting scripts as data
140   #echo "starting to generate plots..."
141   echo "plotting python should be: "$plot_py" ("$os")."
142
143   for i in ${plot_scripts[@]}; do
144     bash -c "$plot_py $plot_py_path/$i"
145   done
146
147   echo "+------------------+"
148   echo "|plotting completed|"
149   echo "+------------------+"
150 }
151
152 function cleanup
153 {
154     ##cleaning up the mess you created!
155     #kill all child proceesses
156     echo "staring cleanup..."
157     echo "killing all lingering child processes..."
158     killall -9 -g $0
159     cd $this_path
160     exit
161 }
162 trap cleanup sighup sigint sigkill;
163 trap "cd $this_path" exit;
164
165 function main
166 {
167   # clear up console
168   #reset
169   # check if jobs_open directory is empty
170   if [ ! "$(ls -a $jobs_open_path)" ]; then
171     echo "there seem to be no open jobs. measuring with default
      parameters."
172     prepare_measurement
173     #take measurements
174     measure | tee -a $log_path/default_$measurement_counter.log
175     # create plot if desired
176     if [ $plot_enabled -eq 1 ]; then
177       plot | tee -a $log_path/default_$measurement_counter.log; fi
178   else
179     prepare_measurement
180     echo "open jobs detected! let's get to work..."
181     jobs=$jobs_open_path/*
182     for job in $jobs; do
183       source $job;
```

```
184       job_name=$(echo $job | rev | cut -d"/" -f1 | rev )
185       log=$log_path/$job_name"_"$measurement_counter.log
186       #echo $job_name
187       cat $job | tee -a $log
188       cat measurement_$link.conf | tee -a $log
189       measure | tee -a $log
190       if [ $plot_enabled -eq 1 ]; then
191         plot | tee -a $log
192       fi
193       if [ $move_after_job_done -eq 1 ]; then
194         cp $job $plot_directory_path/$measurement_counter/
195         mv $job $jobs_done_path/
196       fi
197       export measurement_counter=$((measurement_counter++))
198     done
199   fi
200 }
201
202 if [ $debug_mode -eq 1 ]; then
203   echo "+-----------------+"
204   echo "|debug mode active|"
205   echo "+-----------------+"
206 fi
207
208 if [ $remote_measurement -eq 1 ]; then
209   # call to main included here
210     setup_remote_connection
211   else
212     main
213 fi
```

LISTING A.1: measure.sh

```
1  import numpy as np
2  import myplot
3  import os
4
5  import rtt
6  import throughput as tp
7  import channel_occupation
8  import backoff
9  import sniffer
10
11 # From Bash
12 measurement = [int(os.environ["measurement_counter"])]
13 links = [int(os.environ["link"])]
14 repetitions = int(os.environ["measurement_repetitions"])
15 data_source_path = os.environ["data_source_path"]
16 plot_path =
       os.environ["plot_directory_path"]+"/"+os.environ["measurement_counter"]+"/"
17 plot_type = ["cdf", "boxplot"]
```

```
throughput_data_files =
    os.environ["throughput_data_files"].split(",")
rtt_data_files = os.environ["rtt_data_files"].split(",")
co_data_files = os.environ["co_data_files"].split(",")
sniffer_data_files = os.environ["sniffer_data_files"].split(",")
retxs_data_files = os.environ["retxs_data_files"].split(",")
show_plot = int(os.environ["show_plot_after_measurement"])
rtt_mode = os.environ["rtt_mode"]
max_retxs = 6
eval_mode = "live"
timer = int(os.environ["timer"])
receiver_mode = os.environ["receiver_mode"]

#From Python
plot_pdf = False
boxplot_xticks = [ "measurement "+str(index) for index in
    measurement ]
legend_labels = [ tick.replace("\n", ", ") for tick in
    boxplot_xticks ]

custom_legend_coordinates = {
    "rtt":                  [0.24,0.85,"upper left"],
    "packet_loss":          [1,0,"lower right"],
    "retxs":                [1,0,"lower right"],
    "throughput":           [1,0,"lower right"],
    "diagnosis_sender":     [1,0,"lower right"],
    "diagnosis_receiver":   [1,0,"lower right"],
    "backoff":              [1,0,"lower right"],
    "channel_occupation":   [1,0,"lower right"],
    "sniffer":              [1,0,"lower right"]
}

create_plots = {
    "rtt":                   False,
    "packet_loss":           False,
    "retxs":                 False,
    "throughput":            True,
    "diagnostic":            True,
    "backoff":               True,
    "channel_occupation":    True,
    "sniffer":               True
}

channel_occupation_mode = {
    "occupation_mode":   ["overview", "zoom"],
    "zoom":              [5,7],
    "zoom_mode":         "interval",
    "zoom_interval":     2
}

sniffer_settings =  {
    "sniffer_mode":             ["physical", "smoothed"],
    "link":                     1,
```

```
    "zoom":                     [0.0,timer*repetitions],
    "zoom_mode":                "interval",
    "zoom_interval":            2,
    "smoothing_difference":     0.0001,
    "smoothing_derivative":     0.01,
    "smoothing_range":          [0.0010,0.0013]
}

#Unimplemented, use later
annotations_below   = []
annotations_other   = []

eval_dict = {
    "measurement":              measurement,
    "repetitions":              repetitions,
    "data_source_path":         data_source_path,
    "xticks":                   boxplot_xticks,
    "legend":                   legend_labels,
    "annotations_below":        annotations_below,
    "annotations_other":        annotations_other,
    "throughput_data_files":    throughput_data_files,
    "retxs_data_files":         retxs_data_files,
    "rtt_data_files":           rtt_data_files,
    "show_plot":                show_plot,
    "legend_coordinates":       custom_legend_coordinates,
    "create_plots":             create_plots,
    "links":                    links,
    "rtt_mode":                 rtt_mode,
    "channel_occupation_mode":  channel_occupation_mode,
    "co_data_files":            co_data_files,
    "sniffer_data_files":       sniffer_data_files,
    "sniffer_settings":         sniffer_settings,
    "timer":                    timer,
    "plot_pdf":                 plot_pdf
}

for index,a_plot_type in enumerate(plot_type):
    if plot_type[index] == "cdf":
        grid                = True
    else:
        grid                = True

    eval_dict["plot_type"]  = [plot_type[index]]
    eval_dict["plot_path"]  = plot_path
    eval_dict["grid"]       = grid

    if create_plots["backoff"] == True:
        print("Creating backoff plot!")
        backoff.backoff(**eval_dict).plot()
    if create_plots["rtt"] == True:
        print("Creating rtt plot!")
        rtt.rtt(**eval_dict).plot()
    if create_plots["throughput"] == True:
```

```
121         print("Creating throughput plot!")
122         tp.tp(**eval_dict).plot()
123 
124 # The plots with only one plot type!
125 if create_plots["channel_occupation"] == True:
126     print("Creating channel occupation plot!")
127     channel_occupation.channel_occupation(**eval_dict)
128 if create_plots["sniffer"] == True:
129     print("Creating sniffer energy plot!")
130     sniffer.sniffer(**eval_dict)
131 
132 print("Done.")
```

LISTING A.2: evaluation.py

```
1  import numpy as np
2  import subprocess
3  import myplot
4  import os.path as pt
5  import lines
6  
7  print("Hello from throughput.py!")
8  
9  class tp:
10     def __init__(self,**kwargs):
11         # variables left out for brevity's sake.
12         # pdb.set_trace()
13 
14     def calc(self):
15         print("I'll create a numpy array!")
16         self.data =
       np.zeros(shape=(len(self.measurement),self.repetitions))
17         self.sender_diagnosis =
       np.zeros(shape=(len(self.measurement),self.repetitions))
18         self.receiver_diagnosis =
       np.zeros(shape=(len(self.measurement),self.repetitions))
19         print("Done.")
20 
21         if self.receiver_mode == "single":
22             link = ""
23         else:
24             link = "_"+str(self.links[index])
25 
26         for index,single_measurement in enumerate(self.measurement):
27             for i in range(self.repetitions):
28                 print("I'll open files from iteration "+str(i+1)+"
       for you!")
29                 print("file_path:")
30                 file_path =
       self.data_source_path+'/'+str(self.measurement[index])+'/'+str(i+1)+'/'
```

```
31                  data_sent_file_path =
    file_path+self.throughput_data_files[0]+link+".txt"
32                  ack_received_file_path =
    file_path+self.throughput_data_files[1]+"_"+str(self.links[index])+".txt"
33                  print(data_sent_file_path)
34                  print(ack_received_file_path)
35
36                  data_received_file_path =
    file_path+self.diagnosis_files[0]+link+".txt"
37                  ack_sent_file_path =
    file_path+self.diagnosis_files[1]+link+".txt"
38
39                  if pt.isfile(ack_received_file_path):
40                      print("Let's count some ACKS!")
41                      ackcount =
    lines.linecount(ack_received_file_path)
42                  else:
43                      # no acks found
44                      print("ACK file not found at
    "+ack_received_file_path+".")
45                      ackcount = 0
46                      self.data[index,i] = 0
47                  if pt.isfile(data_sent_file_path):
48                      print("Let's count some data!")
49                      datacount = lines.linecount(data_sent_file_path)
50                      print("ackcount: "+str(ackcount))
51                      print("datacount: "+str(datacount))
52                      # *8 => bytes --> bits ||| /1000 => 1 --> kilo
53                      self.data[index,i] =
    min(datacount,ackcount)*self.packet_size/self.timer*8/1000
54                  else:
55                      # no data sent off
56                      print("Data file not found at
    "+data_sent_file_path+".")
57                      datacount = 0
58                      self.data[index,i] = 0
59
60                  if not self.receiver_mode == "single":
61                      if pt.isfile(data_received_file_path) and
    pt.isfile(ack_sent_file_path) and datacount > 0 and ackcount > 0:
62                          print("Hoorray, diagnosis files found!")
63                          receiver_datacount =
    lines.linecount(data_received_file_path)
64                          receiver_ackcount =
    lines.linecount(ack_sent_file_path)
65                          self.sender_diagnosis[index,i] = 100 -
    (receiver_ackcount - ackcount)/receiver_ackcount*100
66                          self.receiver_diagnosis[index,i] = 100 -
    (datacount - receiver_datacount)/datacount*100
67                      else:
68                          print("Diagnosis files incomplete or
    missing :( \
```

```
69                          (or no sender-side received acks/
    receiver-side received data)!")
70                  else:
71                      print("Diagnosis based on receiver is not
    possible in a single receiver, multiple sender scenario.")
72
73
74          print(self.data)
75
76      def plot(self):
77          self.calc()
78          all_data = []
79
80          for val in self.data:
81              all_data.extend(val)
82
83          #print("all_data:")
84          #print(all_data)
85
86          print("Let's plot! ;-)")
87          if self.create_plots == True or
    self.create_plots["throughput"] == True:
88              myplot.myplot(  data=self.data,
89                  plottype=self.plot_type,
90                  title="Throughput",
91                  xlabel="throughput [kbit/s]",
92                  ylabel="throughput [kbit/s]",
93                  savepath=self.plot_path+"/",
94                  show=self.show_plot,
95                  grid=self.grid,
96                  xticks=self.xticks,
97                  legend=self.legend,
98                  legend_loc=self.legend_loc,
99                  annotations_below=self.annotations_below,
100                 annotations_other=self.annotations_other,
101
    legend_coordinates=self.legend_coordinates["throughput"],
102                 eval_mode=self.eval_mode,
103                 xlims=[0,135])
104
105         if not self.receiver_mode == "single":
106             if self.create_plots == True or
    self.create_plots["diagnostic"] == True:
107                 myplot.myplot(  data=self.receiver_diagnosis,
108                     plottype=self.plot_type,
109                     title="Receiver Receiving Score (Inverse = Data
    Loss)",
110                     xlabel="score [%]",
111                     ylabel="score [%]",
112                     savepath=self.plot_path+"/",
113                     show=self.show_plot,
114                     grid=self.grid,
115                     xticks=self.xticks,
```

```
116                     legend=self.legend,
117                     legend_loc=self.legend_loc,
118                     annotations_below=self.annotations_below,
119                     annotations_other=self.annotations_other,
120
    legend_coordinates=self.legend_coordinates["diagnosis_receiver"],
121                     eval_mode=self.eval_mode)
122
123             myplot.myplot(  data=self.sender_diagnosis,
124                 plottype=self.plot_type,
125                 title="Sender Receiving Score (Inverse = ACK Loss)",
126                 xlabel="score [%]",
127                 ylabel="score [%]",
128                 savepath=self.plot_path+"/",
129                 show=self.show_plot,
130                 grid=self.grid,
131                 xticks=self.xticks,
132                 legend=self.legend,
133                 legend_loc=self.legend_loc,
134                 annotations_below=self.annotations_below,
135                 annotations_other=self.annotations_other,
136
    legend_coordinates=self.legend_coordinates["diagnosis_sender"],
137                 eval_mode=self.eval_mode)
```

LISTING A.3: sniffer.py

```
1  import matplotlib.patches as mpatches
2  import matplotlib.pyplot as plt
3  import numpy as np
4  import os
5  import math
6
7  class myplot:
8      def __init__(   self, plottype, data, bins="",
9                      title="", xlabel="", ylabel="",
10                     show=False, savepath=False, data_x=None,
    **kwargs
11                 ):
12
13         print("Hello from myplot.py!")
14
15         self.data               = np.asarray(data).transpose()
16         self.data_x             = data_x
17
18         print("Title is '"+title+"'.")
19
20         if  (title == "Retransmissions per Frame"
21             or len(data) == 1
22             or "bar"  in plottype
23             or "hist" in plottype
```

```
24              or "broken_barh" in plottype):
25              print("Keeping original data format instead of
    transposing.")
26              self.data               = data
27
28          self.bins               = bins
29          self.plottype           = plottype
30          self.xlabel             = xlabel
31          self.ylabel             = ylabel
32          # Let's have reasonable figure dimensions
33          self.fig, self.ax       = plt.subplots(figsize=(9,6))
34          self.kwargs             = kwargs
35          self.grid               = kwargs.get("grid", False)
36          self.legend             = kwargs.get("legend", [])
37          self.xticks             = kwargs.get("xticks", [])
38          self.legend_loc         = kwargs.get("legend_loc", "best")
39          self.annotations_below  = kwargs.get("annotations_below",
    [])
40          self.annotations_other  = kwargs.get("annotations_other",
    [])
41          self.legend_coordinates = kwargs.get("legend_coordinates",
    False)
42          self.timer              = kwargs.get("timer", 300)
43          self.repetitions        = kwargs.get("repetitions", 5)
44          self.eval_mode          = kwargs.get("eval_mode", "belated")
45          self.xlims              = kwargs.get("xlims", False)
46          self.savepath           = savepath,
47          self.plot_pdf           = kwargs.get("plot_pdf", False)
48
49          plottypes = {
50              "hist":         lambda: self.hist(),
51              "line":         lambda: self.line(),
52              "cdf":          lambda: self.cdf(),
53              "pdf":          lambda: self.pdf(),
54              "boxplot":      lambda: self.boxplot(),
55              "debug":        lambda: self.debug(),
56              "bar":          lambda: self.bar(),
57              "broken_barh":  lambda: self.broken_barh(),
58              "line_xy":      lambda: self.line_xy()
59          }
60
61          titles = {
62              "cdf":          "CDF",
63              "line":         "Line Chart",
64              "line_xy":      "Line Chart",
65              "hist":         "Histogram",
66              "pdf":          "PDF",
67              "boxplot":      "Boxplot",
68              "debug":        "Debug",
69              "bar":          "Bar Chart",
70              "broken_barh":  "Gantt Chart"
71          }
72
```

```
73          for aplot in plottype:
74              #print("single plot in plottype array is:"+str(aplot))
75              self.title = title+" "+titles[aplot]
76              self.savename = self.title
77              self.title = ""
78              plottypes[aplot]()
79              #set axis limits
80              self.ax.set_ylim(ymin=0)
81              if self.xlims != False and not (aplot in ["boxplot",
    "bar"]):
82                  self.ax.set_xlim(self.xlims[0], self.xlims[1])
83              else:
84                  self.ax.set_xlim(xmin=0)
85              plt.tight_layout()
86              self.ax.xaxis.grid(self.grid, linestyle="dashdot")
87              self.ax.yaxis.grid(self.grid, linestyle="dashdot")
88              if not aplot == "boxplot" and not aplot ==
    "broken_barh":
89                  if len(np.asarray(self.data).transpose()) ==
    len(self.legend):
90                      if self.legend_coordinates == False:
91                          box = self.ax.get_position()
92                          self.ax.set_position([
93                              box.x0,
94                              box.y0+box.height*0.3,
95                              box.width,
96                              box.height*0.7
97                          ])
98                          self.ax.legend(fancybox=True,
99                                      loc='upper center',
100                                     bbox_to_anchor=(0.5, -0.15))
101                     else:
102                         if self.legend_coordinates[2] != "best":
103                             self.ax.legend(fancybox=True,
104
    loc=self.legend_coordinates[2],
105
    bbox_to_anchor=(self.legend_coordinates[0],
106
    self.legend_coordinates[1]))
107                         else:
108                             self.ax.legend(fancybox=True,loc="best")
109             else:
110                 print ( "len(self.data) = "
111                         +
    str(len(np.asarray(self.data).transpose()))
112                         + " and len(self.legend) = "
113                         + str(len(self.legend))
114                         +" don't match!")
115         # Add anotations:
116         self.annotate(annotation)
117         # Save and show plot
118         if(savepath):
```

```
119                 print("***savepath***")
120                 print(savepath)
121                 self.save(savepath, aplot)
122             if(show):
123                 self.show()
124 
125             #plt.close(self.fig)
126             self.fig.clear()
127             #print(self.data)
128 
129     def bar(self):
130 
131         colors = ['steelblue', 'orange', 'green', 'red', 'purple',
    'brown', 'pink']
132         color_repetitions = math.ceil(len(self.data)/len(colors))
133         colors = color_repetitions * colors
134 
135         print(self.data)
136         for idx,val in enumerate(self.data):
137             data_points=len(self.data[idx])
138             width=5/data_points
139             index=np.arange( data_points )
140 
141             self.ax.bar(left=idx*width+width/2,
142                 height=val,
143                 color=colors[idx],
144                 alpha=0.5
145              )
146 
147             self.setLabels(ylabel=self.ylabel,
148                 xlabel="measurement",
149                 title=self.title
150             )
151 
152     def broken_barh(self):
153         plot_data = []
154         debug_data = []
155         for index,item in
    enumerate(self.data["occupation_starting"]):
156 
    plot_data.append(list(zip(self.data["occupation_starting"][index],
    self.data["occupation_durations"][index])))
157 
158         for index,item in enumerate(self.data["acks_received"]):
159 
    debug_data.append(list(zip(self.data["acks_received"][index],
    self.data["acks_received_bar_width"][index])))
160 
161         print("plot_data:")
162         #print(plot_data)
163         print("debug_data:")
164         #print(debug_data)
165         print("data_len:")
```

```
166         data_len = len(plot_data)
167         print(data_len)
168         debug_len = len(debug_data)
169         print("debug_len:")
170         print(debug_len)
171 
172         print("data and ack lengths:")
173         for index,item in enumerate(plot_data):
174             if index % 2 == 0:
175                 print("data index "+str(index)+":")
176             else:
177                 print("ack index "+str(index)+":")
178             print(len(item))
179 
180         for index, item in enumerate(debug_data):
181             print ("debug data index "+str(index)+":")
182             print(len(item))
183             #print(item)
184 
185         self.ax.set_ylim(10, 5*data_len+20)
186         if self.xlims == False:
187             self.ax.set_xlim(0, self.timer*self.repetitions)
188         self.ax.xaxis.grid(self.grid, linestyle="dashdot")
189         self.ax.yaxis.grid(self.grid, linestyle="dashdot")
190 
191         self.ax.set_yticks([x*10+15 for x in
    range(int(data_len/2))])
192         self.xticks = [tick.replace(",", ",\n") for tick in
    self.xticks]
193         self.ax.set_yticklabels([self.xticks[index] for index in
    range(int(data_len/2))])
194 
195         self.setLabels(
196             xlabel="time[s]",
197             title=self.title
198         )
199 
200         colors = ['blue','red','black']
201         transparency=0.7
202 
203         patch_a  = mpatches.Patch(color=colors[0],
    alpha=transparency, label="data sent")
204         patch_b  = mpatches.Patch(color=colors[1],
    alpha=transparency, label='ack sent')
205         patch_c  = mpatches.Patch(color=colors[2],
    alpha=transparency, label="ack received")
206 
207         if self.legend_coordinates[2] != "best":
208             self.ax.legend( handles=[patch_a,patch_b,patch_c],
209                             fancybox=True,
210                             loc=self.legend_coordinates[2],
211 
    bbox_to_anchor=(self.legend_coordinates[0],
```

```
212
    self.legend_coordinates[1]))
213         else:
214             self.ax.legend( handles=[patch_a,patch_b,patch_c],
215                             fancybox=True,
216                             loc="best")
217
218         for index,item in enumerate(plot_data):
219             print("Added (data) set with index "+str(index)+" to
    plot.")
220             if index % 2 == 0:
221                 self.ax.broken_barh(item,((index+1)*5+5,13),
    facecolors=colors[0], alpha=0.5)
222             elif (index-1) % 2 == 0: # well else should be enough
    here :)
223                 self.ax.broken_barh(item,((index)*5+5,13),
    facecolors=colors[1], alpha=0.5)
224
225         for index,item in enumerate(debug_data):
226             print("Added (debug) set with index "+str(index)+" to
    plot.")
227             if index % 2 == 0:
228                 self.ax.broken_barh(item,((index+1)*5+5,13),
    facecolors=colors[2], alpha=0.5)
229             elif (index-1) % 2 == 0:
230                 self.ax.broken_barh(item,((index)*5+5,13),
    facecolors=colors[2], alpha=0.5)
231
232         plt.tight_layout()
233
234     def line(self):
235         from scipy.interpolate import interp1d
236         x = np.arange(1, len(self.data)+1, 1)
237         y = self.data
238         f = interp1d(x,y)
239         plt.plot(x, f(x), 'k')
240
241         self.setLabels( xlabel="measurement",
242                         ylabel=self.ylabel,
243                         title=self.title
244         )
245
246     def line_xy(self):
247         from scipy.interpolate import interp1d
248         x = self.data_x
249         y = self.data
250         f = interp1d(x,y)
251         plt.plot(x, f(x), 'k')
252         self.setLabels( xlabel=self.xlabel,
253                         ylabel=self.ylabel,
254                         title=self.title
255         )
256
```

```
257     def hist(self):
258         self.n, self.bins, self.patches = self.ax.hist(x=self.data,
259                         bins=self.bins,
260                         normed=1,
261                         histtype='step',
262                         cumulative=True,
263                         label=self.legend)
264         self.setLabels( xlabel=self.xlabel,
265                         ylabel="cumulative density",
266                         title=self.title)
267         print(self.patches)
268 
269     def cdf(self):
270         #print(self.data)
271         print(self.legend)
272         markers = ["x","v","o","^","8","s","p","+","D","*"]
273         linestyles = ["-", "--", "-.", ":","-", "--", "-.",
    ":","-", "--"]
274         linewidths = [1.8,1.65,1.5,1.35,1.2,1.05,1,0.9,0.8,0.75]
275 
276         if self.eval_mode == "belated":
277             cdf_data = np.asarray(self.data).transpose()
278         if self.eval_mode == "live":
279             cdf_data = np.asarray(self.data).transpose()
280             if self.title == "Retransmissions per Frame":
281                 cdf_data = [cdf_data]
282         if len(self.data) == 1:
283             print("Hooray, my title is "+self.title+".")
284             cdf_data = self.data
285 
286         if len(cdf_data) > 1:
287             for index,item in enumerate(cdf_data):
288                 print("index:"+str(index))
289                 print("___markers____")
290                 print(markers[index])
291                 x = np.sort(item)
292                 y = np.arange(1,len(x)+1) / len(x)
293                 x = np.insert(x,0,x[0])
294                 y = np.insert(y,0,0)
295                 self.plot = plt.step(x,
296                     y,
297                     marker=markers[index],
298                     linestyle=linestyles[index],
299                     linewidth=linewidths[index],
300                     markevery=range(1,len(x)),
301                     label=self.legend[index])
302         else:
303             x = np.sort(cdf_data[0])
304             y = np.arange(1,len(x)+1) / len(x)
305             x = np.insert(x,0,x[0])
306             y = np.insert(y,0,0)
307             print(x)
308             print(y)
```

```
309             self.plot = plt.step(x,
310                     y,
311                     marker=markers[0],
312                     linestyle=linestyles[0],
313                     linewidth=linewidths[0],
314                     markevery=range(1,len(x)),
315                     label=self.legend[0])
316 
317         self.setLabels( xlabel=self.xlabel,
318                         ylabel="cumulative density",
319                         title=self.title)
320         self.ax.set_ylim(ymax=1)
321 
322     def pdf(self):
323         self.n, self.bins, self.patches = self.ax.hist(x=self.data,
324                         bins=self.bins,
325                         align='left',
326                         fill='true',
327                         normed=1,
328                         cumulative=False,
329                         label=self.legend)
330         self.setLabels( xlabel=self.xlabel,
331                         ylabel="probability density",
332                         title=self.title)
333 
334         cm = plt.cm.get_cmap('jet')
335         for index, patch in enumerate(self.patches):
336             plt.setp(patch, 'facecolor',
    cm(float(index/len(self.patches))))
337 
338     def boxplot(self):
339         print(self.data)
340 
341         self.plot = plt.boxplot(self.data,
342                                 #notch=True,
343                                 patch_artist=True,
344                                 flierprops=dict(marker='x'))
345 
346         colors = ['steelblue', 'peachpuff', 'green', 'red',
    'purple', 'brown', 'pink']
347         color_repetitions = math.ceil(len(self.data)/len(colors))
348         colors = color_repetitions * colors
349 
350         for patch, color in zip(self.plot['boxes'], colors):
351             patch.set_facecolor(color)
352 
353         self.setLabels( xlabel="measurement",
354                         ylabel=self.ylabel,
355                         title=self.title)
356 
357         boxdict = self.ax.boxplot(self.data)
358         fliers  = boxdict["fliers"]
359 
```

```
360         for j in range(len(fliers)):
361             yfliers = boxdict['fliers'][j].get_ydata()
362             xfliers = boxdict['fliers'][j].get_xdata()
363             ufliers = set(yfliers)
364             for i, uf in enumerate(ufliers):
365                 self.ax.text(xfliers[i] + 0.05, uf,
    list(yfliers).count(uf))
366
367         if len(np.asarray(self.data).transpose()) ==
    len(self.xticks):
368             print(self.xticks)
369             plt.xticks([x+1 for x in
    range(len(self.xticks))],self.xticks)
370             #self.ax.set_xticklabels(self.xticks);
371         else:
372             print ( "len(self.data) = "
373                     + str(len(self.data))
374                     + " and len(self.xticks) = "
375                     + str(len(self.xticks))
376                     +" don't match!")
377
378     def setLabels(self, xlabel="", ylabel="", title=""):
379         self.ax.set_xlabel(xlabel)
380         self.ax.set_ylabel(ylabel)
381         self.ax.set_title(title)
382
383     def save(self, savepath, plot_type):
384         #savename = self.title
385         savename = self.savename
386         savename = savename.lower()
387         savename = savename.replace(" ", "_")
388         self.fig.savefig(savepath+savename+".png")
389         if self.plot_pdf:
390             self.fig.savefig(savepath+savename+".pdf")
391
392     def show(self):
393         plt.show()
394
395     def annotate(self, annotations=False):
396         if annotations:
397             self.ax.annotate(annotations)
398         # Add custom annotation code here
399             ...
400
401     def debug(self):
402         print("data: "+str(self.data))
403         print("bins: "+str(self.bins))
404         print("plottype: "+self.plottype)
```

LISTING A.4: myplot.py

B

ABBREVIATIONS

AM amplitude modulation

BEB binary exponential backoff

BMAC Berkeley MAC

CA carrier aggregation

CCA clear channel assessment

CDMA code division multiple access

CDF cumulative distribution function

CLI command line interface

CSMA carrier sense multiple access

CSMA/CA CSMA with collision avoidance

CSMA/CD CSMA with collision detection

CSAT carrier sense adaptive transmission

CTS clear to send

CW contention window

DC duty cycle

DCF distributed coordination function

DIFS DCF interframe spacing

DIY do it yourself

DLL Data Link Layer

DOS denial of service

DSP digitial speech processor

EIFS extended interframe spacing

FDMA Frequency Division Multiple Access

FM frequency modulation

FPGA field programmable gate array

GNU GNU is not Unix

GR GNU Radio

GRC GNU Radio Companion

GUI graphical user interface

IEEE Institute for Electrical and Electronics Engineers

LAA Licensed Assisted Access

LabVIEW Laboratory Virtual Instrumentation Engineering Workbench

LAN local area network

LLC logical link control

LBT listen before talk

LTE Long Term Evolution

MAC medium access control

MATLAB Matrix Laboratory

MCS modulation and coding scheme

MU medium utilization

NAV network allocation vector

NIC network interface card

OFDMA orthogonal FDMA

OOT out-of-tree

OSI Open Systems Interconnection

PCF point coordination function

PDU packet data unit

PHY physical (layer)

PIFS PCF interframe spacing

PU power units

PMT polymorphic type

QoS quality of service

QPSK quadrature phase-shift keying

RF radio frequency

RTS request to send

RTT round-trip time

RX receiving/reception

SDK software development kit

SMAC Sensor MAC

SDL supplemental downlink

SDR software defined radio

SDU service data unit

SIFS short interframe spacing

SWIG simplified wrapper and interface generator

TDD time division duplex

TDMA time division multiple access

TMAC Timeout MAC

TX transmitting/transmission

UE user equipment

UHD USRP hardware driver

USRP universal software radio peripheral

WLAN wireless LAN

WSN wireless sensor networks

List of Figures

List of Tables

BIBLIOGRAPHY

[1] T. Nihtilä, V. Tykhomyrov, O. Alanen, M. A. Uusitalo, A. Sorri, M. Moisio, S. Iraji, R. Ratasuk, and N. Mangalvedhe, "System performance of lte and ieee 802.11 coexisting on a shared frequency band," in *2013 IEEE Wireless Communications and Networking Conference (WCNC)*, April 2013, pp. 1038–1043.

[2] "Lte-u technology and coexistence," http://www.lteuforum.org/uploads/3/5/6/8/3568127/lte-u_coexistence_mechansim_qualcomm_may_28_2015.pdf, accessed: 2017-10-11.

[3] A. Ghosh, R. Ratasuk, B. Mondal, N. Mangalvedhe, and T. Thomas, "Lte-advanced: next-generation wireless broadband technology [invited paper]," *IEEE Wireless Communications*, vol. 17, no. 3, pp. 10–22, June 2010.

[4] J. S. Lee, Y. W. Su, and C. C. Shen, "A comparative study of wireless protocols: Bluetooth, uwb, zigbee, and wi-fi," in *IECON 2007 - 33rd Annual Conference of the IEEE Industrial Electronics Society*, Nov 2007, pp. 46–51.

[5] N. Bhushan, J. Li, D. Malladi, R. Gilmore, D. Brenner, A. Damnjanovic, R. T. Sukhavasi, C. Patel, and S. Geirhofer, "Network densification: the dominant theme for wireless evolution into 5g," *IEEE Communications Magazine*, vol. 52, no. 2, pp. 82–89, February 2014.

[6] I. Gomez-Miguelez, A. Garcia-Saavedra, P. D. Sutton, P. Serrano, C. Cano, and D. J. Leith, "srslte: an open-source platform for lte evolution and experimentation," pp. 25–32, 10 2016.

[7] C. Capretti, F. Gringoli, N. Facchi, and P. Patras, "Lte/wi-fi co-existence under scrutiny: An empirical study," in *Proceedings of the Tenth ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation, and Characterization*, ser. WiNTECH '16. New York, NY, USA: ACM, 2016, pp. 33–40. [Online]. Available: http://doi.acm.org/10.1145/2980159.2980164

[8] X. Zhang and K. G. Shin, "Enabling coexistence of heterogeneous wireless systems: Case for zigbee and wifi," in *Proceedings of the Twelfth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, ser. MobiHoc '11. New York, NY, USA: ACM, 2011, pp. 6:1–6:11. [Online]. Available: http://doi.acm.org/10.1145/2107502.2107510

[9] P. Yi, A. Iwayemi, and C. Zhou, "Developing zigbee deployment guideline under wifi interference for smart grid applications," *IEEE Transactions on Smart Grid*, vol. 2, no. 1, pp. 110–120, March 2011.

[10] C. F. Chiasserini and R. R. Rao, "Coexistence mechanisms for interference mitigation between ieee 802.11 wlans and bluetooth," in *Proceedings.Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 2, 2002, pp. 590–598 vol.2.

[11] "Ieee standard for local and metropolitan area networks: Overview and architecture - redline," *IEEE Std 802-2014 (Revision to IEEE Std 802-2001) - Redline*, pp. 1–166, June 2014.

[12] J. D. Day and H. Zimmermann, "The osi reference model," *Proceedings of the IEEE*, vol. 71, no. 12, pp. 1334–1340, Dec 1983.

[13] A. Tanenbaum, *Computer Networks*, 4th ed. Prentice Hall Professional Technical Reference, 2002.

[14] M. S. Gast, *802.11 Wireless Networks: The Definitive Guide, Second Edition*. O'Reilly Media, Inc., 2005.

[15] D. R. Raymond, R. C. Marchany, M. I. Brownfield, and S. F. Midkiff, "Effects of denial-of-sleep attacks on wireless sensor network mac protocols," *IEEE transactions on vehicular technology*, vol. 58, no. 1, pp. 367–380, 2009.

[16] V. Garg, *Wireless Communications & Networking*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007.

[17] A. Bachir, M. Dohler, T. Watteyne, and K. K. Leung, "Mac essentials for wireless sensor networks," *IEEE Communications Surveys Tutorials*, vol. 12, no. 2, pp. 222–248, Second 2010.

[18] H. J. Kwon, J. Jeon, A. Bhorkar, Q. Ye, H. Harada, Y. Jiang, L. Liu, S. Nagata, B. L. Ng, T. Novlan, J. Oh, and W. Yi, "Licensed-assisted access to unlicensed spectrum in lte release 13," *IEEE Communications Magazine*, vol. 55, no. 2, pp. 201–207, February 2017.

[19] I. Demirkol, C. Ersoy, and F. Alagoz, "Mac protocols for wireless sensor networks: a survey," *IEEE Communications Magazine*, vol. 44, no. 4, pp. 115–121, April 2006.

[20] J. Polastre, J. Hill, and D. Culler, "Versatile low power media access for wireless sensor networks," in *Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems*, ser. SenSys '04. New York, NY, USA: ACM, 2004, pp. 95–107. [Online]. Available: http://doi.acm.org/10.1145/1031495.1031508

[21] "Simple fm receiver," http://electronicsforu.com/electronics-projects/simple-fm-receiver, accessed: 2017-10-03.

[22] "Am receiver circuit," http://www.electroschematics.com/9043/am-receiver-circuit/, accessed: 2017-10-03.

[23] "About gnu radio," https://www.gnuradio.org/about/, accessed: 2017-10-05.

[24] "What is labview?" http://www.ni.com/en-us/shop/labview.html, accessed: 2017-10-05.

[25] "Python integration toolkit for labview by enthought," http://sine.ni.com/nips/cds/view/p/lang/en/nid/213990, accessed: 2017-10-05.

[26] "Usrp support package from communications system toolbox," https://de.mathworks.com/hardware-support/usrp.html, accessed: 2017-10-06.

[27] "Tutorials core concepts," https://wiki.gnuradio.org/index.php/TutorialsCoreConcepts, accessed: 2017-10-03.

[28] "Gnu radio manual and c++ api reference 3.7.10.1," https://gnuradio.org/doc/-doxygen, accessed: 2017-10-04.

[29] N. Rupasinghe and Güvenç, "Licensed-assisted access for wifi-lte coexistence in the unlicensed spectrum," in *2014 IEEE Globecom Workshops (GC Wkshps)*, Dec 2014, pp. 894–899.

[30] J. Jeon, H. Niu, Q. C. Li, A. Papathanassiou, and G. Wu, "Lte in the unlicensed spectrum: Evaluating coexistence mechanisms," in *2014 IEEE Globecom Workshops (GC Wkshps)*, Dec 2014, pp. 740–745.

[31] A. M. Cavalcante, E. Almeida, R. D. Vieira, S. Choudhury, E. Tuomaala, K. Doppler, F. Chaves, R. C. D. Paiva, and F. Abinader, "Performance evaluation of lte and wi-fi coexistence in unlicensed bands," in *2013 IEEE 77th Vehicular Technology Conference (VTC Spring)*, June 2013, pp. 1–6.

[32] G. Thonet, P. Allard-Jacquin, and P. Colle, "Zigbee-wifi coexistence," *Schneider Electric White Paper and Test Report*, vol. 1, pp. 1–38, 2008.

[33] R. Gummadi, H. Balakrishnan, and S. Seshan, "Metronome: Coordinating spectrum sharing in heterogeneous wireless networks," in *2009 First International Communication Systems and Networks and Workshops*, Jan 2009, pp. 1–10.

[34] S. Pollin, I. Tan, B. Hodge, C. Chun, and A. Bahai, "Harmful coexistence between 802.15.4 and 802.11: A measurement-based study," in *2008 3rd International Conference on Cognitive Radio Oriented Wireless Networks and Communications (CrownCom 2008)*, May 2008, pp. 1–6.

[35] "Ieee standard for information technology–telecommunications and information exchange between systems local and metropolitan area networks–specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications," *IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012)*, pp. 1–3534, Dec 2016.

[36] "Gnu radio guided tutorial in python," https://wiki.gnuradio.org/index.php/Guided_Tutorial_GNU_Radio_in_Python, accessed: 2017-10-08.

Eidesstattliche Versicherung

Name, Vorname

Matrikelnummer (freiwillige Angabe)

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Arbeit/Bachelorarbeit/ Masterarbeit* mit dem Titel

selbständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Ort, Datum

Unterschrift

*Nichtzutreffendes bitte streichen

Belehrung:

§ 156 StGB: Falsche Versicherung an Eides Statt

Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt

(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.

(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:

Ort, Datum

Unterschrift