# 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

We thank...

# Contents

# ABSTRACT

The demand for higher wireless transmission rates and capacities is growing rapidly. As a consequence, options to more efficiently make use of the limited frequency resources are being explored. One possibility

# 1

# Introduction

# 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 layers' responsibilities. We will 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 [1]. 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 $C$

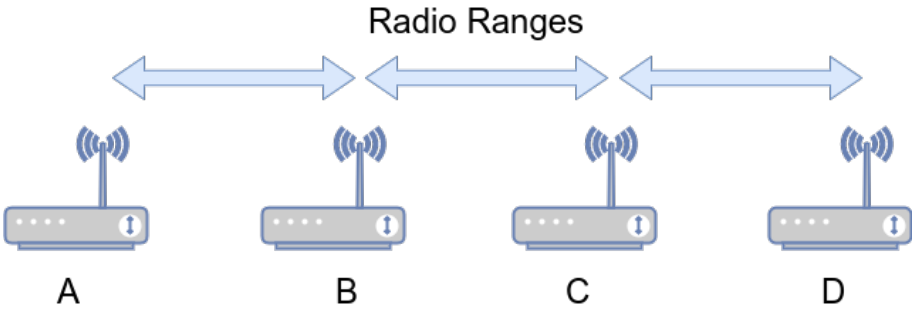



FIGURE 2.1: Setup to explain the hidden and exposed node problem. Each node can only reach its direct neighbors.

and falsely assume that the channel is idle and start transmitting, which leads to a collision. This is the hidden node problem [3] [4].

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 [3] [4] .

#### 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

| 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 [2]

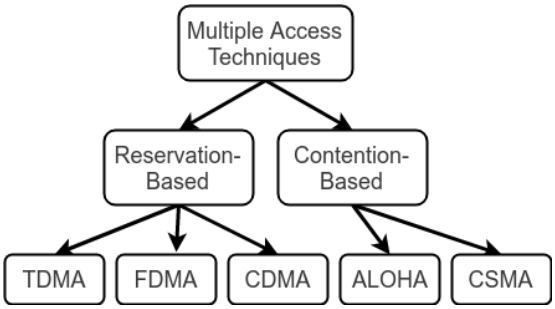



FIGURE 2.2: Classification of MAC techniques

(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 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 [5].

### *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 [6]. 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 [7] 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 will proceed with discussing representative protocols of the two categories. Thereafter, we will 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. Owing to the fact that we do not incorporate reservation-based protocols in our experiments we will 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 [7].

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 [6].

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 [6].

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 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 [3]:

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:

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

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

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 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 $t_{\text{slot}}$ 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*

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 will be 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.

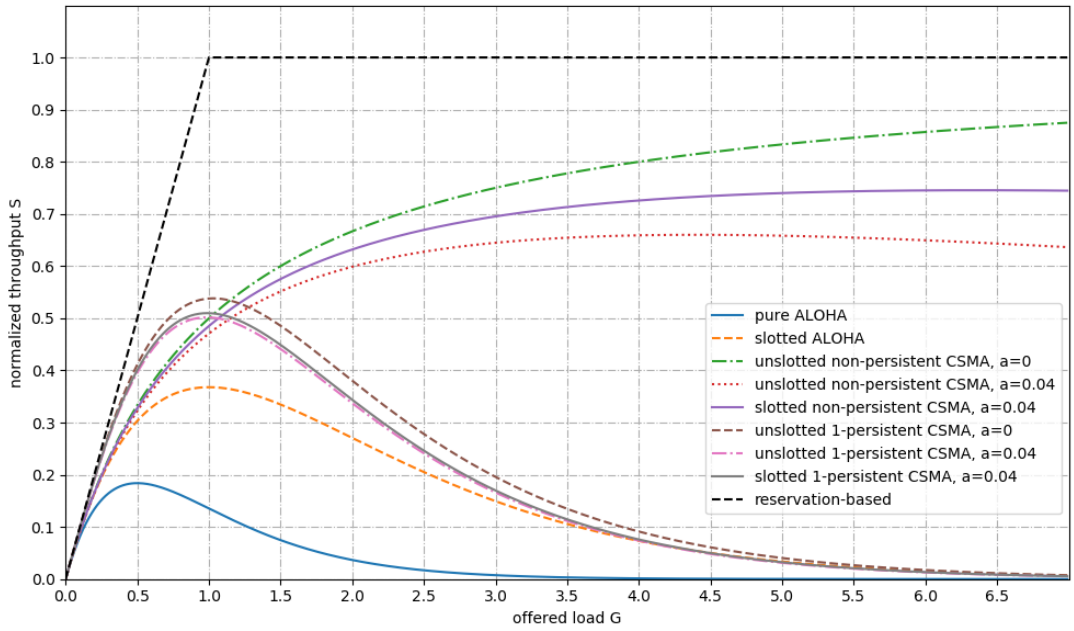


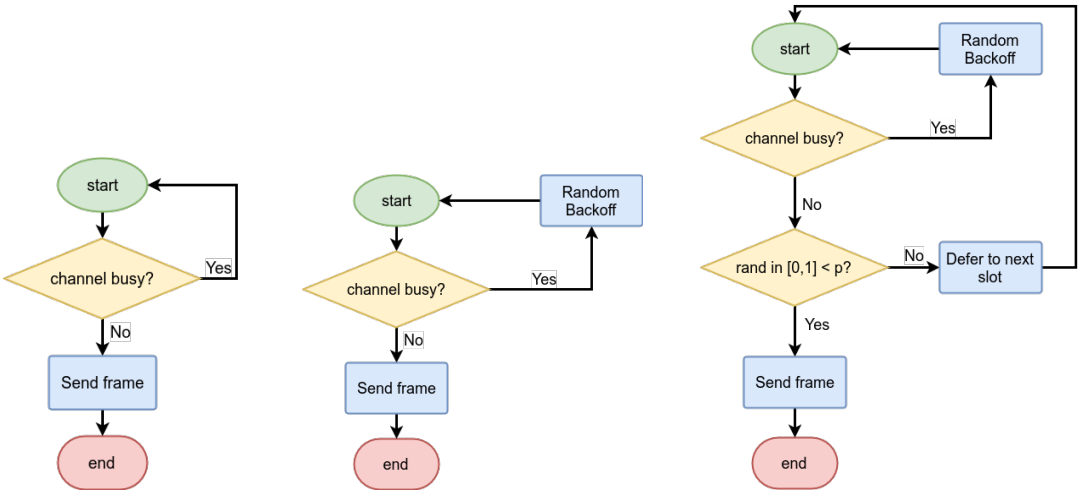

FIGURE 2.3: Normalized throughput over offered load according to formulae in [3], [6], [7], , with $a = \tau/T_p$, where $\tau$ is the maximum propagation delay and $T_p$ the packet transmission time and under the assumptions made in Section 2.1.5.1.

Despite being a substantial improvement over ALOHA, this protocol has at least two problems [3]:

- 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 $B$ and $C$ are waiting for their turn. Once $A$ finished transmission $B$ and $C$ will 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. $B$ will assume 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




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

FIGURE 2.4: The three basic flavors of CSMA

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 [3].

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 received and transmitted energy or pulse width of the signal, 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

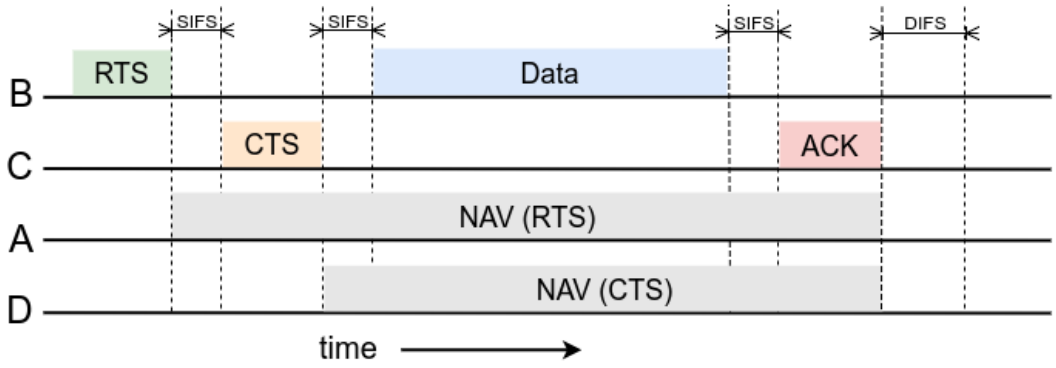



FIGURE 2.5: Virtual carrier sensing in CSMA/CA, as described in [3] and [4]

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.

#### *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 will discuss next in accordance to [4] and [6].

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 is 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 [4]. With BEB a node picks a slot in the so-called contention window (CW). This slot is just a random integer and the contention window is a range lower-bounded by zero and upper-bounded by $2^{n+m}-1$, whereed busy again the whole BEB procedure is repeated with $n$ incremented, thus the CW doubled, hence **binary** exponential backoff. The

motivation for $CW_{\text{min}}$ is to reduce the cha $m$ is a fixed integer, $2^m$ called minimum CW ($CW_{\text{min}}$) and $n = 0$ for the moment. After picking the slot the node waits for $t_w =$ #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 sensed busy again the whole BEB procedure is repeated with $n$ incremented, thus the CW doubled, hence **binary exponential** backoff. The motivation for $CW_{\text{min}}$ 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.

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 the so-called network allocation vector (NAV), which are node-internal timers reminding $A$ and $D$ that the channel is still in use. This mechanism is called virtual carrier sensing due to the fact that the nodes defer their transmission based on the information received through other frames.

#### *2.1.5.5 Licensed Assisted Access*

Generally[2], 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 licensor and is thus generally more reliable in terms of quality of service [9]. One principal approach to ensure harmonious coexistence of LTE and WiFi in the unlicensed band is License Assisted Access (LAA), relying on LBT. Since the LBT mechanism of LAA largely resembles CSMA/CA [3] it seems quite natural to assume it will coexist better with WiFi [8]. The other one is LTE-U which uses DCs and is discussed in Section 2.1.6.1.

### *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] for both license assisted access (LAA) as well as LTE-U

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

#### 2.1.6.1 LTE-U

LTE-U uses Carrier Sense Adaptive Transmission (CSAT), which tries to avoid primary channels of WiFi transmissions and other LTE-U operators. If that is not possible, duty-cycles are dynamically adapted depending on WiFi medium utilization (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 [9].

#### 2.1.6.2 Sensor MAC

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 [7] [10].

#### 2.1.6.3 Timeout MAC

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 [7].

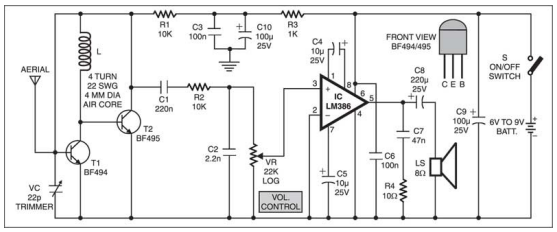
#### 2.1.6.4 Berekeley MAC

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. In BMAC CCA is based on outlier detection, instead of thresholding like in CSMA further reducing energy use [11].

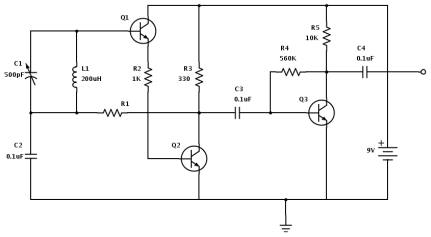
## 2.2 SOFTWARE-DEFINED RADIO

### 2.2.1 Purpose of 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.




(a) FM Receiver [12]




(b) AM Receiver [13]

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

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).

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.

### *2.2.2 What is 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 [14].

A proprietary, well-documented alternative to GNU Radio is LabVIEW developed by National Instruments [15]. 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 [16].

Mathworks MATLAB/Simulink also provides a communication systems toolbox. However, the devices we used are not on the list of officially supported devices [17].

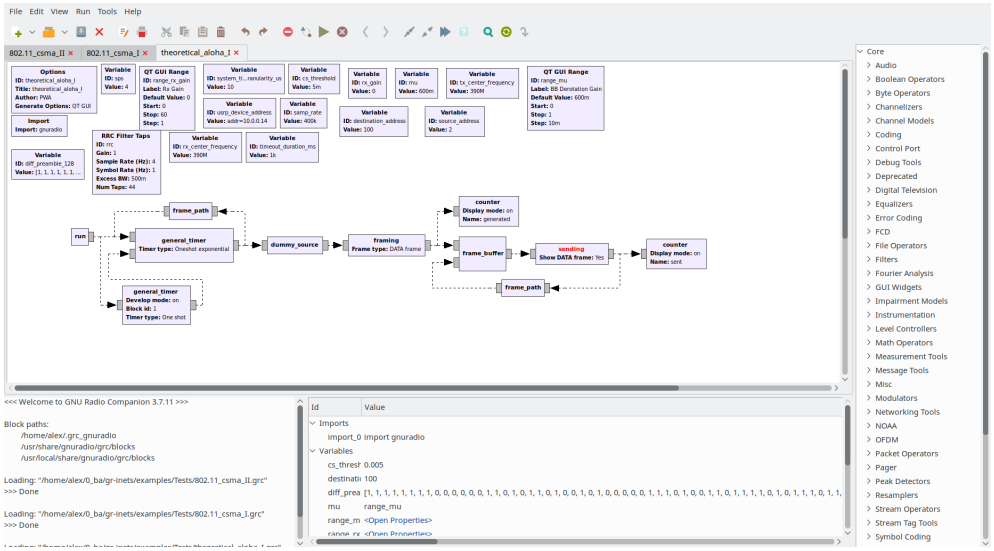



FIGURE 2.7: GNU Radio Companion GUI

### 2.2.3 *Core Concepts of GNU Radio*

#### 2.2.3.1 *Flowgraphs and Blocks*

The two most basic concepts of GNU Radio are flowgraphs and blocks. As mentioned in 2.2.2 flowgraphs are directed graphs, whose nodes are functional blocks and whose vertices determine the direction of data flow [18].

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 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.

#### 2.2.3.2 *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 [18] [19].

#### *2.2.3.3 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 [19].

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.

# 3

# Related Work

## 3.1 LTE-U/WiFi Coexistence Studies

Gomez-Miguelez et al. propose srsLTE, an open-source SDR library for the PHY layer of LTE release 8 [20]. In contrast to many earlier simulation-based studies they have actually set up physical devices, better reflecting system level details of both technologies and providing insight for real-world deployments [21] [22] [23] [24] . Their testbed comprises of several WiFi and LTE links, for which they used Ettus USRP B210 boards (LTE) and low-power single-board computers from Soekris (WiFi). 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, WiFi and LTE TX power, LTE bandwidth, LTE central frequency (i.e. LTE and WiFi spectrum overlap).

Their main results can be summarized as follows:

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

Another empirical study [25] conducted by Capretti et al. also evaluates LTE Unlicensed/WiFi coexistence based on LTE-U. Their testbed consists of one eNB and one UE, one WiFi access point and five WiFi nodes. Their WiFi 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. The software used includes srsLTE to build up a LTE release 8 compliant LTE stack, as well as GNU Radio.

The following parameters were subject of interest: duty cycle, WiFi power settings WiFi MCS (modulation and coding scheme) and packet size. The metrics measured were satisfied load in percent, total WiFi throughput, WiFi jitter and LTE packet loss.

Their main findings can be summarized as follows:

- The duty cycle patterns are a main influence on achievable WiFi 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 WiFi transmissions if the TX power levels are comparable and no duty cycling is employed.
- If WiFi TX power is increased, WiFi load negatively impacts LTE throughput.
- There is no panacea strategy ensuring maximum WiFi throughput operating under different MCSs and packet sizes.
- LTE performance is unaffected by WiFi contention levels.

## 3.2 CONNECTION TO OUR WORK

Both studies aim at understanding under which circumstances duty-cycle-based LTE-U can coexist with WiFi in the unlicensed band by conducting experimental research with SDR (USRP) LTE and commodity WiFi devices.

We, too, carry out experimental research and use USRPs to understand the influence of parameters used in CSMA/CA in a setup with two links each either employing ALOHA or a CSMA/CA on the overall performance. While the studies focus on the variation of power and bandwidth parameters of the transmission, we put emphasis on timing aspects.

# 4

# MEASUREMENT METHODOLOGY

This chapter is dedicated to answering the questions of what was measured and how results were obtained. Firstly, the measurement setup is discussed. Secondly, the GNU Radio flowgraphs are explained. Subsequently, with reference to the flowgraphs measurement metrics are formally defined in view of the necessity to verify the recorded data. 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 will discuss the quality standards of the measurements.

## 4.1 MEASUREMENT SETUP

The setup consists of two USRP2s from Ettus Research and two USRP 2920s from National Instruments. The former pair is programmed as receiver and sniffer, whereas the latter as senders as depicted in 4.1. Each USRP was connected to a gigabit switch through a LAN cabel. The scripts running on the devices were launched from a local computer with the IP 134.130.223.151, which was remote controlled from my private laptop. Hereafter, we will 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. Table 4.1 contains other necessary information to reproduce the measurement results.

Other important parameters were the timeout[1] that we set to 100ms, where the round-trip time (RTT) as defined in 4.3.2 was 68ms at most. We used the quadrature phase-shift keying (QPSK) MCS.

## 4.2 GNU RADIO FLOWGRAPHS

A GNU Radio flowgraph is a directed graph, whose each vertex called block implements a certain functionality of the MAC protocol and the edge determine the direction of data flow as discussed in Section 2.2.3.1. The blocks used are part of a out-of-tree (OOT) [2] module programmed by Peng Wang.

### *4.2.1 Receiver and Sniffer*

Figure 4.2.1 shows the two-way handshake receiving logic. After frame integrity is checked by the `frame_check` block and type is confirmed to be data frame an acknowledgment by the `frame_type_check` is generated. The `frame_probe`

[1] the period of time for each data packet within which an ACK must be received or otherwise declared lost

[2] an OOT module is an external module not provided in the standard setup of GR

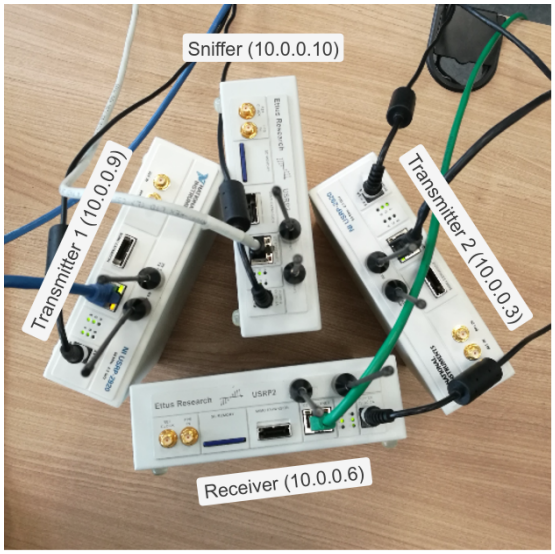



FIGURE 4.1: Photo of the measurement setup.

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

TABLE 4.1: Setup parameters

blocks record the times when the frames reach certain positions in the flowgraph 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 sender.

The sniffer (flowgraph in Figure 4.2.1) 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 senders can be made according to the energy levels, since the sniffer is located between the senders 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 4.3.

### 4.2.2 Pure ALOHA Transmitter

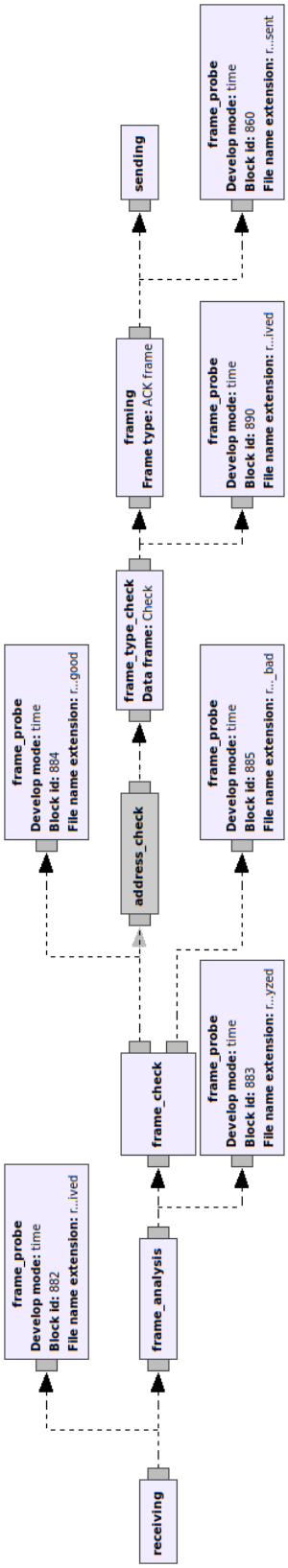
The flowgraph, whose discussion follows, is depicted in Figure 4.2.2. The `run` block enables us to start several senders exactly at the same time, which is useful if we execute the flowgraphs manually without the automated measurement scripts. Payload is generated in the `dummy_source` block, packed into a frame in the `framing` block and buffered in the `frame_buffer` block. The interval between generated frames is determined 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.

### 4.2.3 CSMA Transmitter

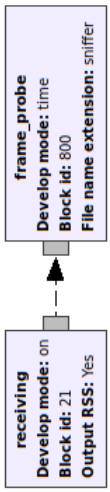
The CSMA transmitter (Figure 4.2.3) is based on the ALOHA transmitter, but features extra mechanisms as described in section 2.1.5.2, which will now be discussed. The flowgraph aims at resembling IEEE 802.11 DCF and features CCA through thresholding in the `carrier_sensing` block. Despite the fact that this block has the feature of adaptively determining an appropriate carrier sensing threshold we chose a fixed value of 0.002 power units (PU) [3]. 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.

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




(a) Receiver




(b) Sniffer

FIGURE 4.2: GRC Receiver Flowgraphs

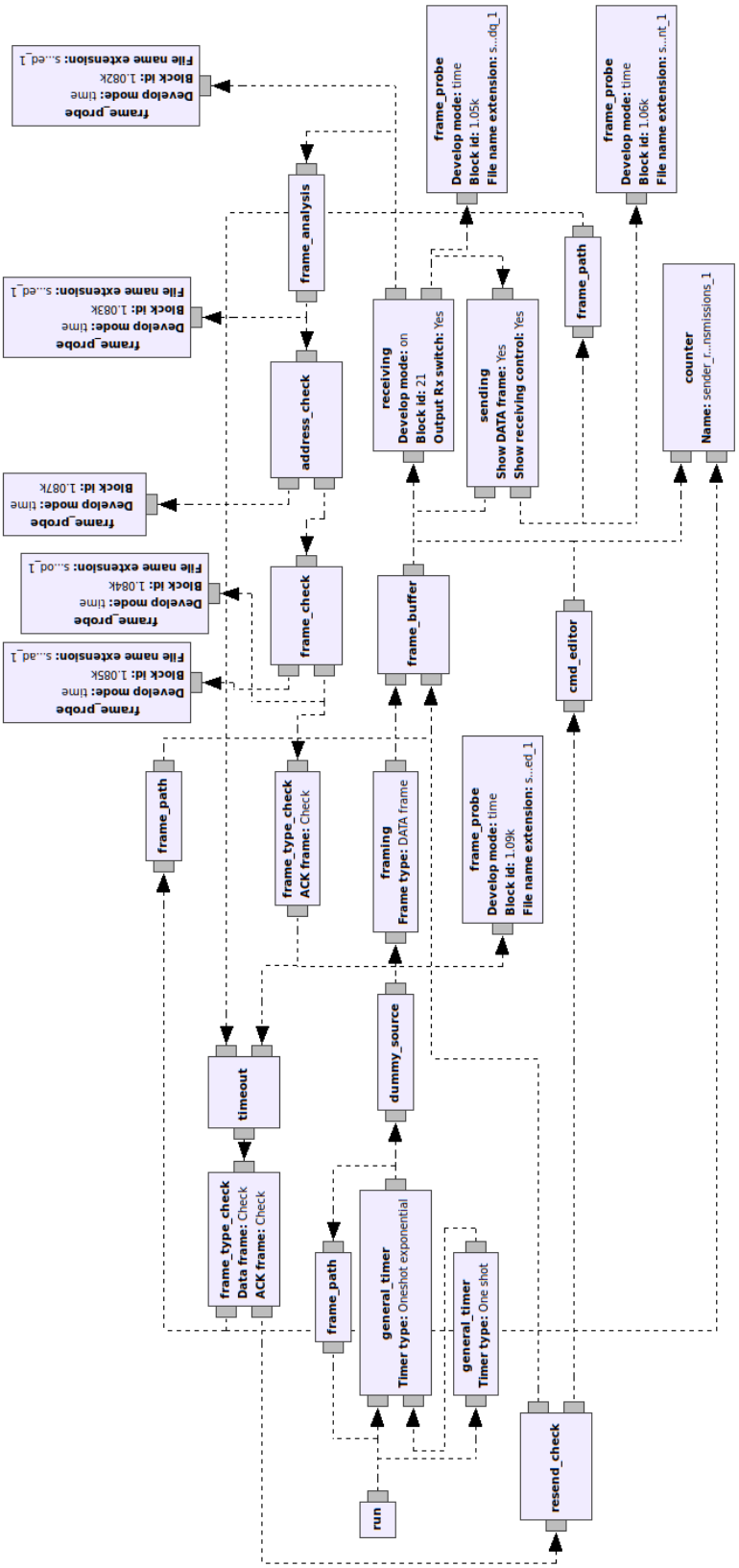



FIGURE 4.3: GRC Pure ALOHA Transmitter Flowgraph

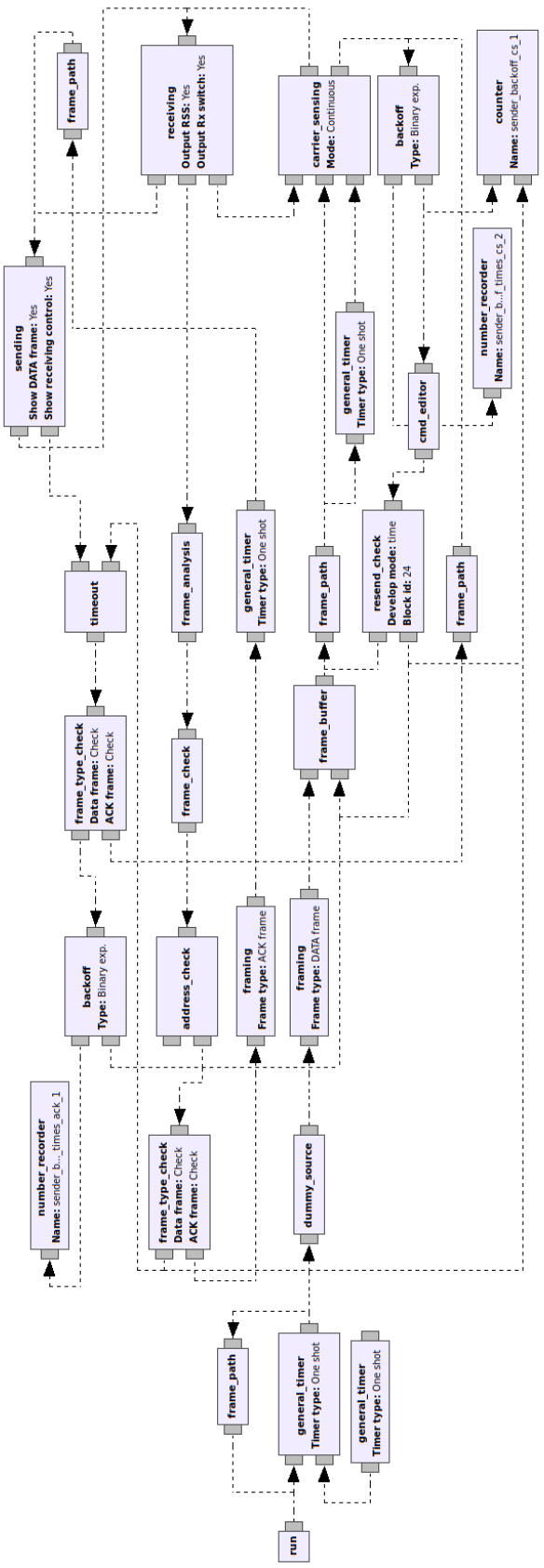



FIGURE 4.4: GRC CSMA Transmitter Flowgraph

## 4.3 MEASUREMENT METRICS

All recorded metrics will be defined in this section. Furthermore, we will describe the way how they were obtained and how we verified them. 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 4.2.1, 4.2.2 and 4.2.3. 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 sender. 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.

### *4.3.1 Throughput*

We define throughput as the mean useful data[4] transmission rate in the unit kbit/s. We obtain this metric simply by counting the number of ACKs received at the sender, 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. Joint 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.

### *4.3.2 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 this 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
```

[4] payload and headers disregarding retransmissions

```
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 will 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.

### 4.3.3 *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 $n_{\text{ACKs}}$ and $n_{\text{data}}$ 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.

### 4.3.4 *Backoff Times*

The script `backoff.py` sums up three different backoff times: Firstly, the backoff due to failed [5] carrier sensing. Secondly, we capture the backoff times due to successful transmission 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 in our view reflects the efficiency of the CSMA protocols in dependency on the parameters DIFS, SIFS and backoff slot length [6]. We chose a minimum contention window of $2^5 = 32$ backoff slots with binary exponential increase.

### 4.3.5 *Packet Durations & Channel Occupation*

The channel occupation chart as in Figure **??** 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 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 40ms

[5] by failed we mean a negative CCA, i.e. a busy channel.

[6] 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.

to transmit and an ACK frame took 7ms. The variation of frame length 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 will be limited to a range of a few seconds at most.

### *4.3.6 Channel Energy Level*

The energy levels (denoted in a linear-scale, non-negative power unit) in 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[7]. Even collisions are clearly visible and which sender caused them. Throughput ratio among senders can easily be verified by opting in data representation 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 senders have sent an equal number of data packets.

## 4.4 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 literally 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 @ 450MHz",
    r"unsaturated ALOHA ~ Poisson($1/\lambda=200ms$), Link 2 @ 450MHz",
    "CSMA\nDIFS=15ms\nSIFS=0ms\nBO=0ms\nLink 1 @ 450MHz\n Baseline",
    "unsaturated ALOHA\nLink 2 @ 450MHz\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).

[7] Frame durations can be verified by reading the time values from the x-axis. RTT and frame delay as per definition in Section 4.3.2 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.

We will 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` directory after completion. In the job files important variables 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`.

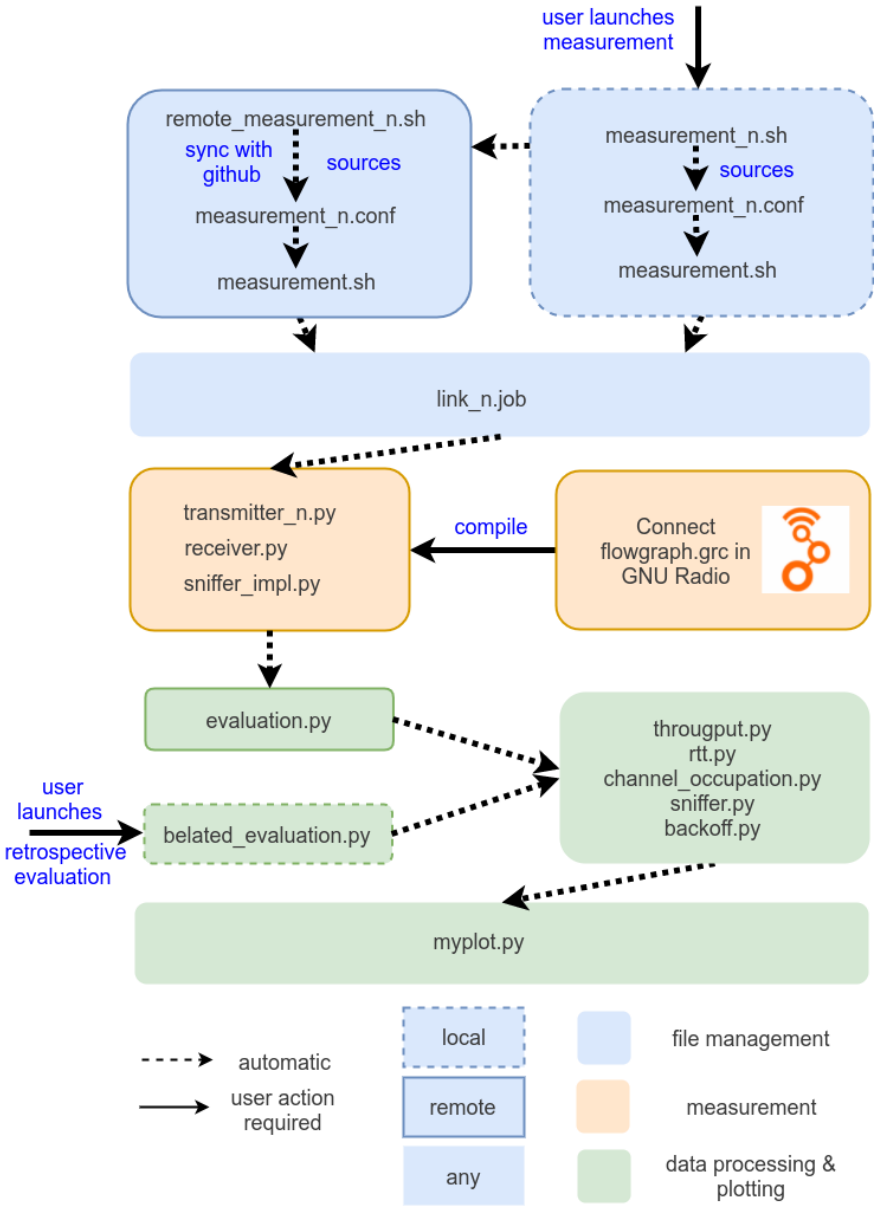
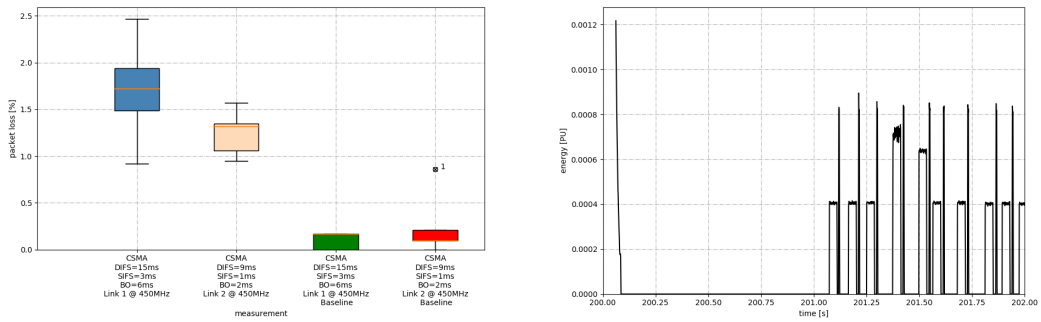



FIGURE 4.5: The three-phase measurement script system




(a) packet loss

(b) channel energy

FIGURE 4.6: Plots depicting the quality standards of our measurement

## 4.5 QUALITY STANDARDS

**Statistical Reliability** Each measurement features five repetitions of the same à 100 seconds. Compared to a single measurement of 500 seconds this has the downside that script and hardware initialization[8] (about 1.1 seconds per repetition as can be seen in 4.6 (b)[9]) 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 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 to and processed by the script and compared with manually calculated results in a similar fashion as GNU Radio QA tests discussed in [26]. 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 (a). This problem of packet loss is generally worse for link 1, because it is further away from the receiver as is shown in Figure 4.1.

[8] In order to repeat the measurement we simply kill the GR flowgraph Python script and restart it.

[9] Of the 1.1 seonds less than 100ms are script initialization overhead since as soon as the the energy is nonzero the devices started working.

# 5

# MEASUREMENT RESULTS

In this chapter we discuss different combinations of MAC protocols employed on the two links. Firstly, we briefly discuss how we configured the flowgraphs to model the MAC protocols. Thereafter, we assess the measurement results where both senders employ the same MAC protocols. Eventually, we do the same where the senders employ different MAC protocols.

## 5.1 FLOWGRAPH PROTOCOL CONFIGURATION

**Pure ALOHA** For pure ALOHA senders we use the flowgraph as described in Section 4.2.2 and shown in Figure 4.2.2.

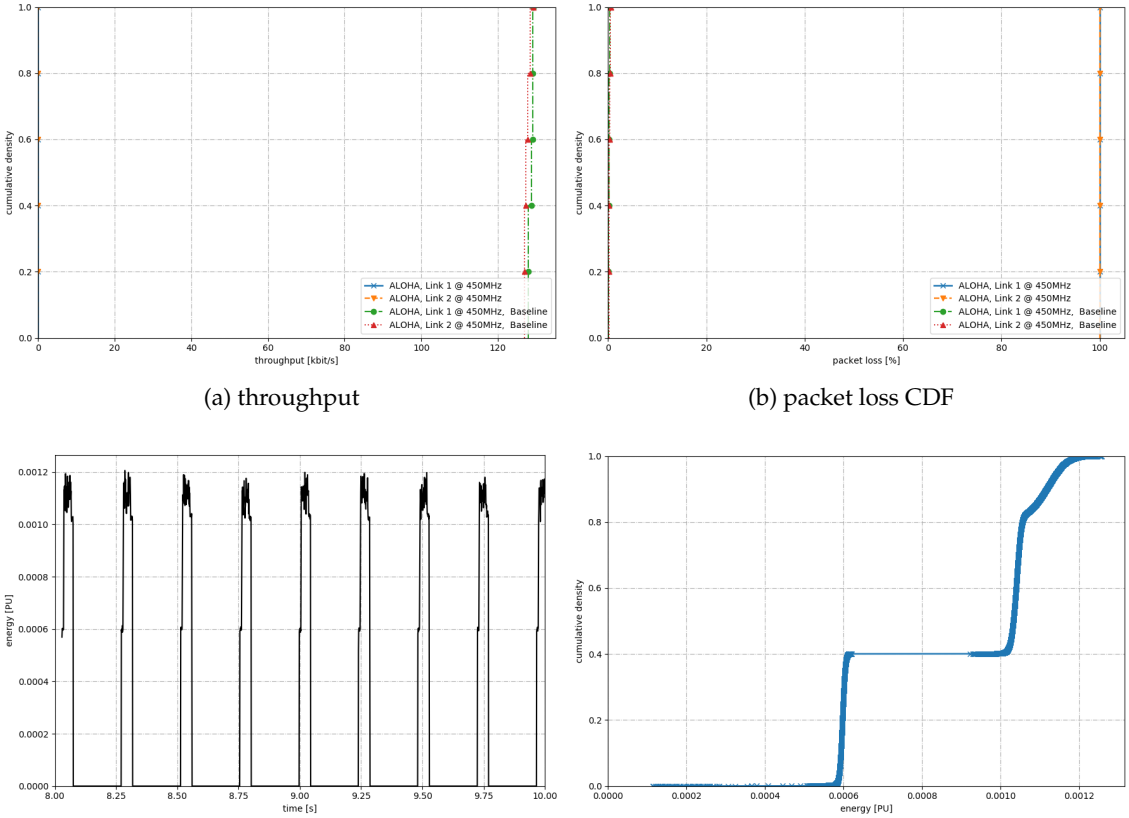
**CSMA/CA** As mentioned in Section 4.2.3 we are not featuring the optional IEEE 802.11 RTS/CTS exchange, realize DIFS and SIFS with `general_timers` and the backoff with the `backoff` blocks.

**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 will refer to it as 1-persistent CSMA.

**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.

## 5.2 SAME PROTOCOL COMBINATIONS

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




(a) throughput

(b) packet loss CDF

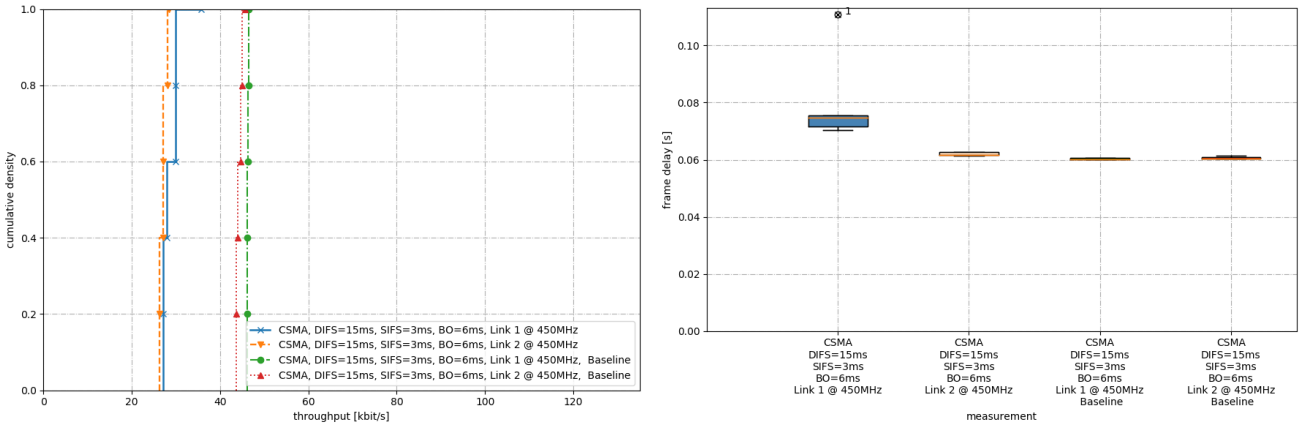
(c) channel energy

(d) channel energy CDF

FIGURE 5.1: Measurement results for two ALOHA senders in the same channel

### 5.2.1 *Pure ALOHA*

For two links with saturated ALOHA traffic we should have zero throughput, since each and every package collides. Figure 5.1 (a) confirms this assumption. The corresponding packet loss of 100% is depicted in Subfigure (b). A reference that can be read off Subfigure (a) value is the throughput of a single 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 joint throughput to this value. Furthermore, a physical view on the channel from the sniffer's perspective is provided in Subfigure (c). Due to the fact that the two transmissions of the senders are not completely overlapping we can see that we don't have a single sender with observed transmission energy level around 0.0011 PU, but instead two senders, where the observed energy level of the first transmitter is around 0.0006 PU, which the channel energy CDF in Subfigure (d) 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.



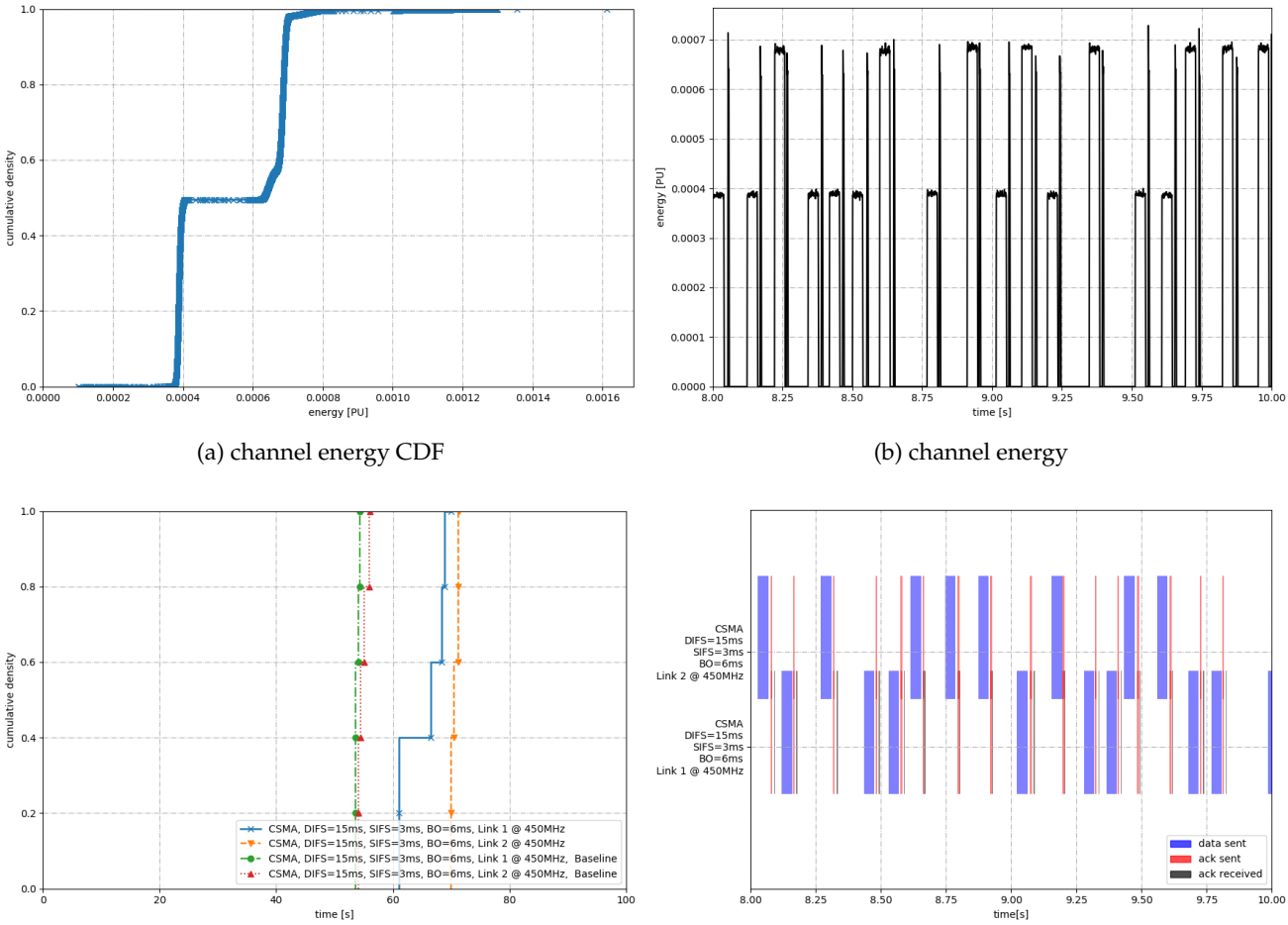

(a) throughput

(b) frame delay

FIGURE 5.2: Measurement results for the CSMA/CA with high parameter set

### *5.2.2 CSMA/CA With High Parameter Set*

First of all with "high parameter set" we mean we chose high values for DIFS, SIFS and backoff slot time (BO) from which expected that they lead to good coexistence of the two links. In particular, we chose DIFS=15ms, SIFS=3ms and BO=6ms. Figure 5.2 (a) shows that the throughput roughly halves when two links are active at the same time. Subfigure (a) 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 $\mathrm{sec}\,label$ :




(a) channel energy CDF

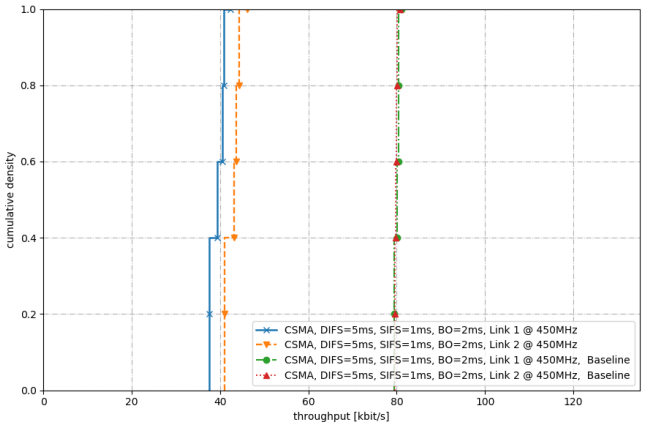
(b) channel energy

(c) total backoff

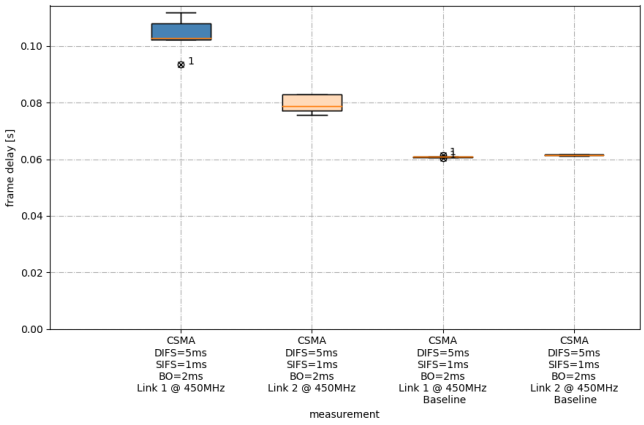
(d) logical channel occupation

FIGURE 5.3: Measurement results for the CSMA/CA with high parameter set

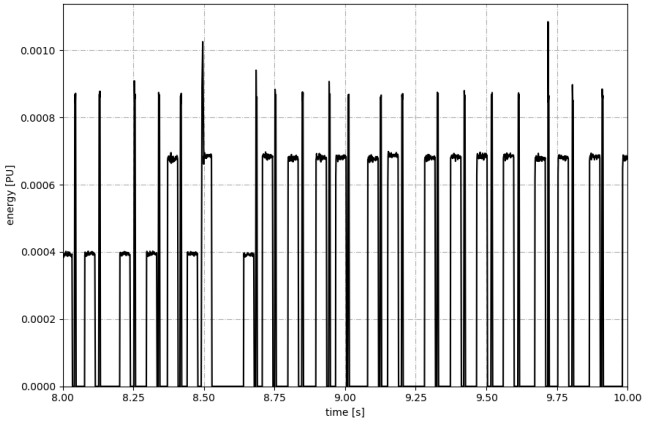
### *5.2.3 CSMA/CA With Low Parameter Set*



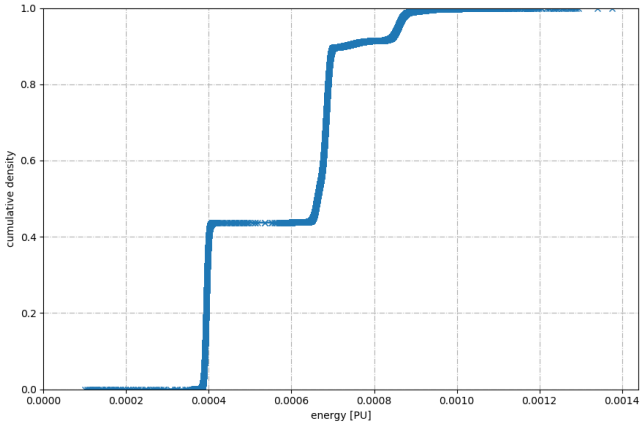

(a) throughput



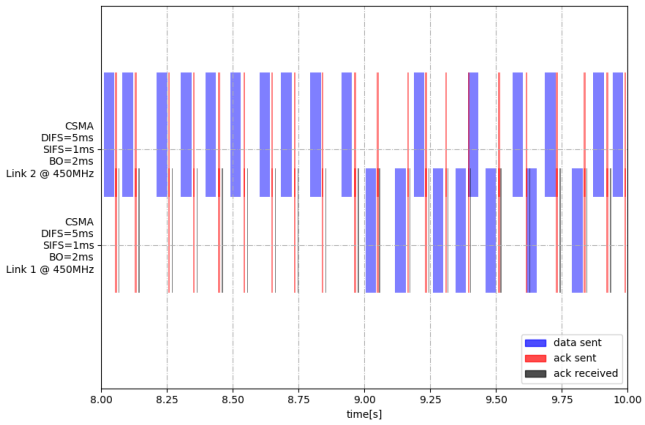

(b) frame delay



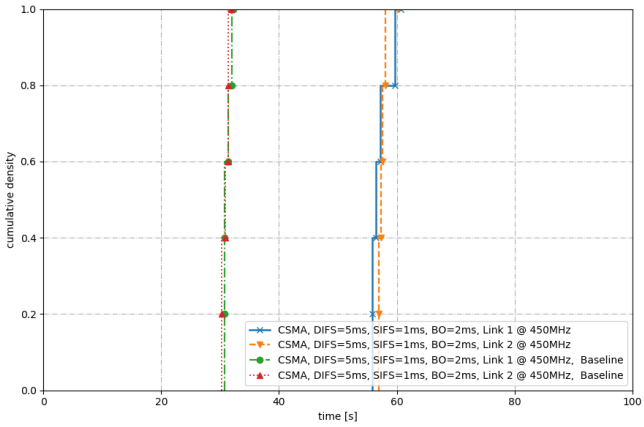

(c) channel energy




(d) channel energy CDF



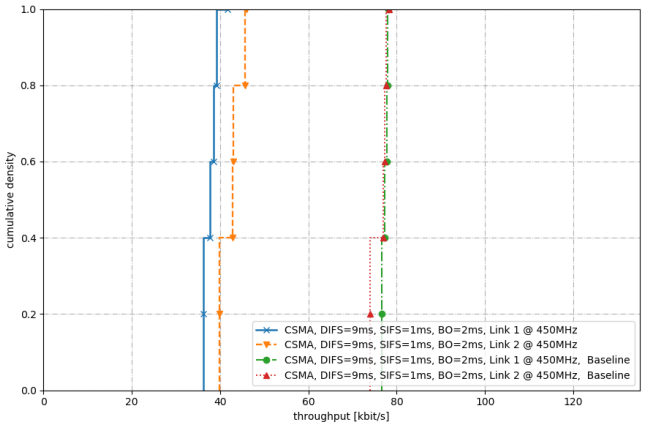

(e) logical channel occupation



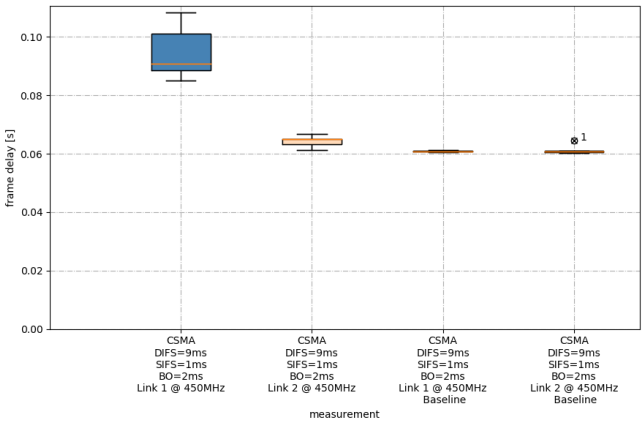

(f) total backoff

FIGURE 5.4: Measurement results for the CSMA/CA with low parameter set

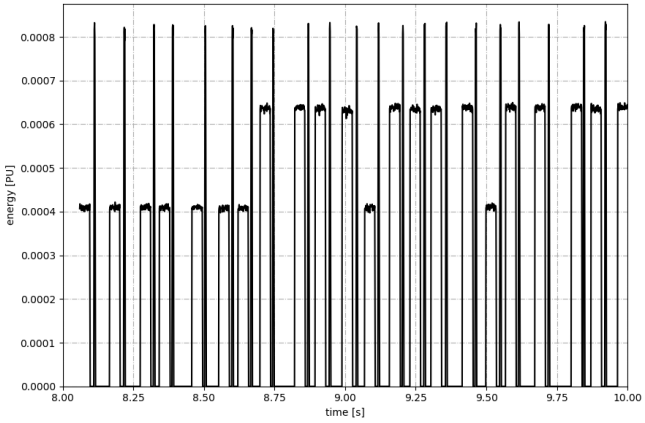
### *5.2.4 CSMA/CA With Medium Parameter Set*



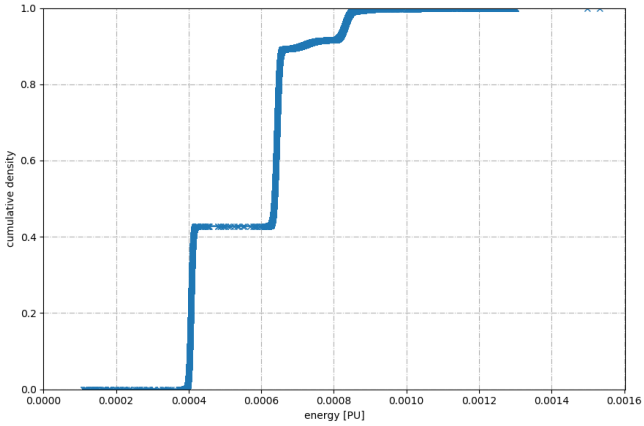

(a) throughput



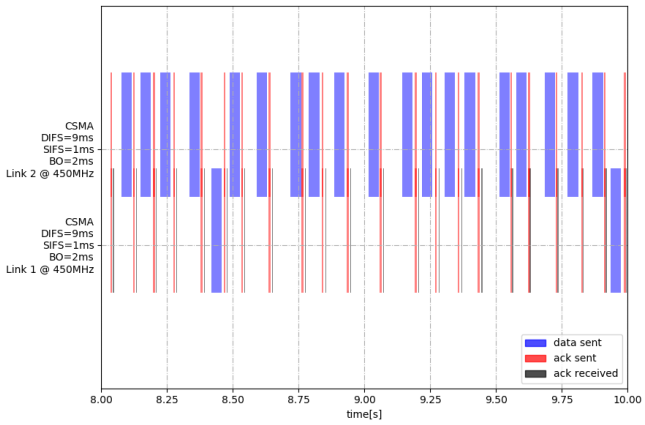

(b) frame delay



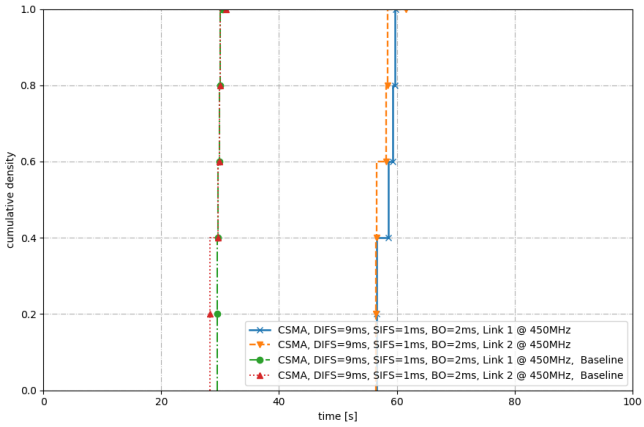

(c) channel energy




(d) channel energy CDF



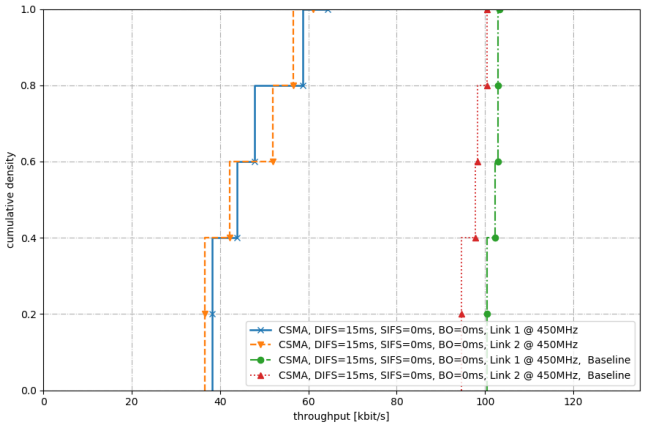

(e) logical channel occupation



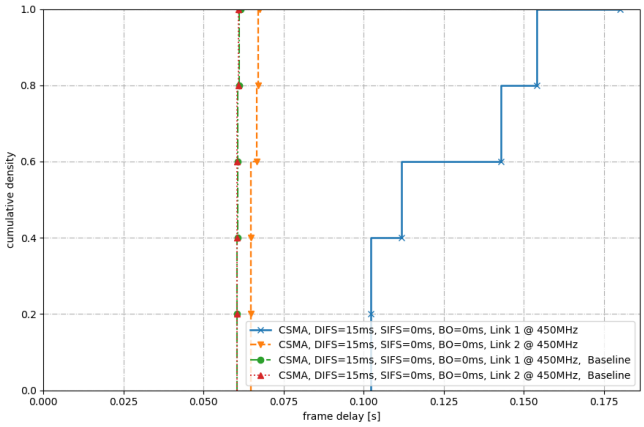

(f) total backoff

FIGURE 5.5: Measurement results for the CSMA/CA with medium parameter set

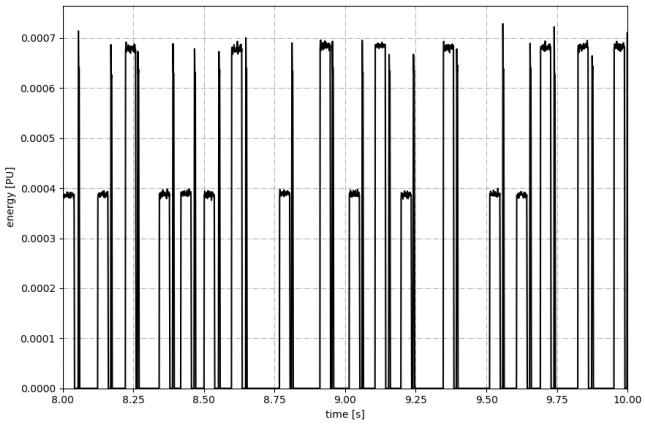
### *5.2.5 1-persistent CSMA*



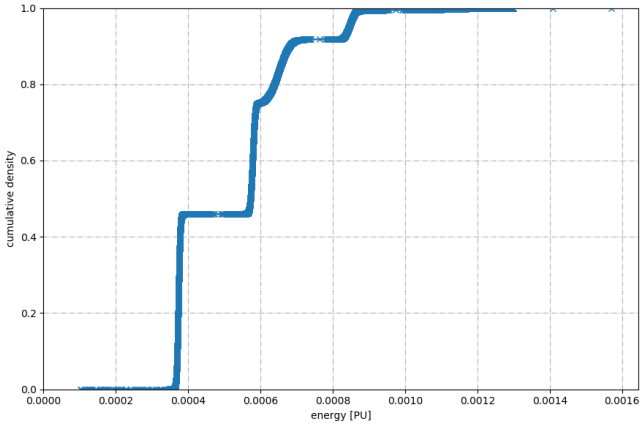

(a) throughput



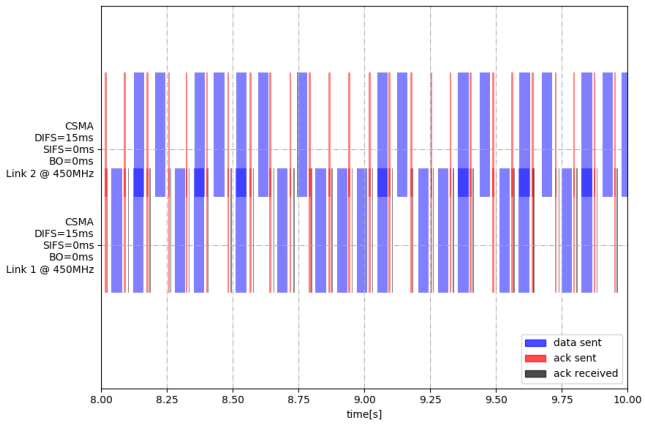

(b) frame delay



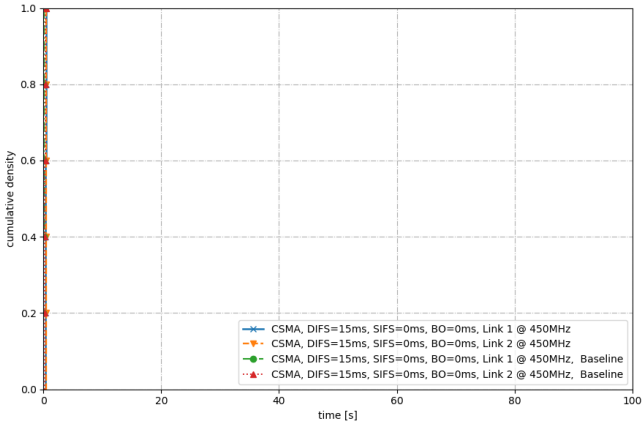

(c) Channel Energy




(d) Channel Energy CDF




(e) Logical Channel Occupation

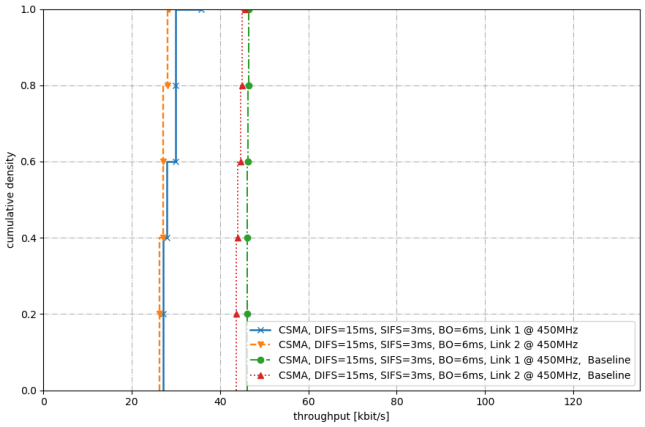



(f) Total Backoff

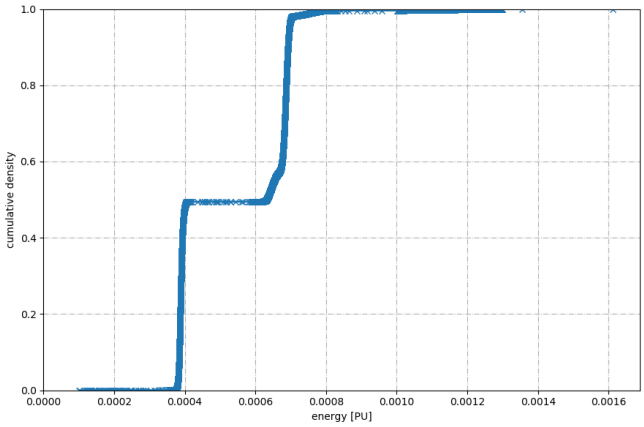
FIGURE 5.6: Measurement results for 1-persistent CSMA/CA

## 5.3 DIFFERENT PROTOCOL COMBINATIONS

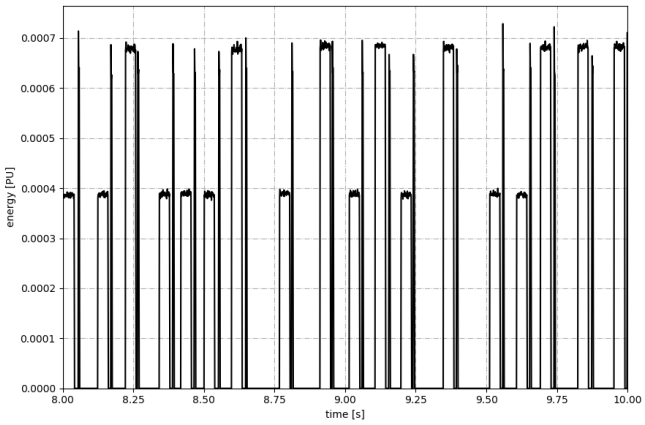
### *5.3.1 ALOHA and CSMA/CA*



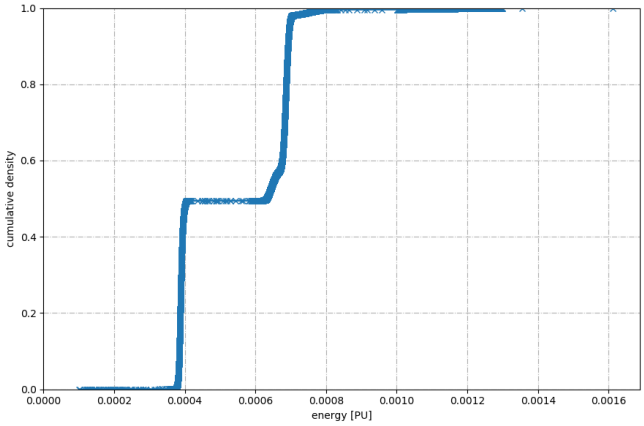

(a) Throughput



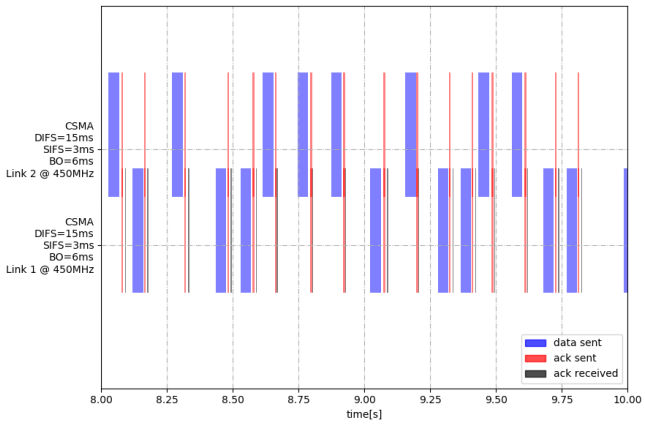

(b) RTT



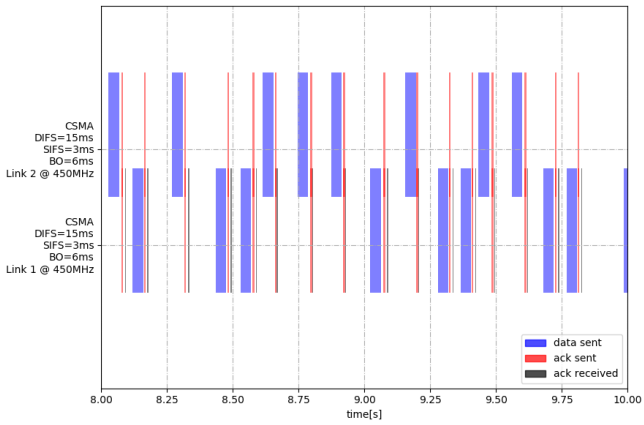

(c) Channel Energy




(d) Channel Energy CDF



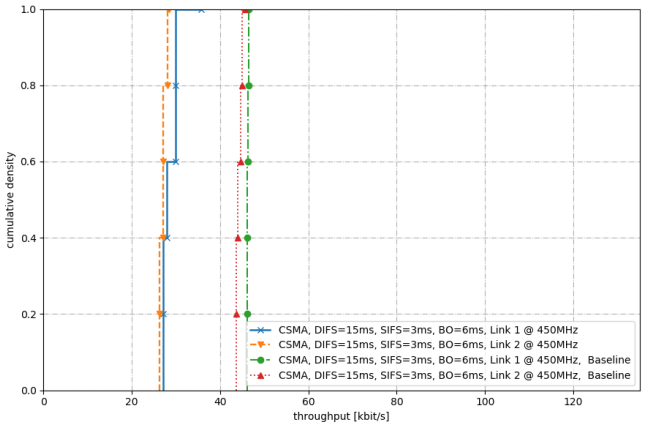

(e) Logical Channel Occupation



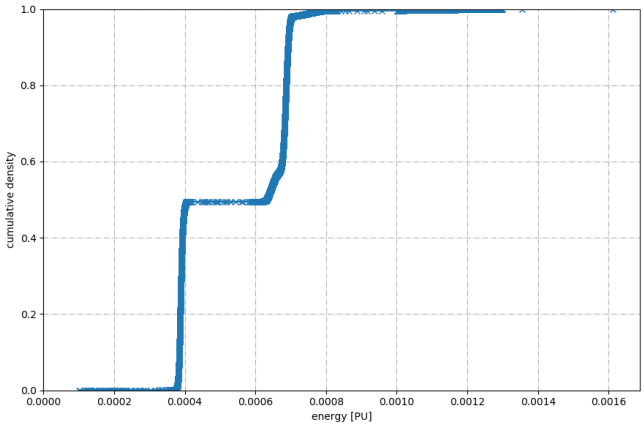

(f) Total Backoff

FIGURE 5.7: Measurement results for the CSMA/CA with large parameter set

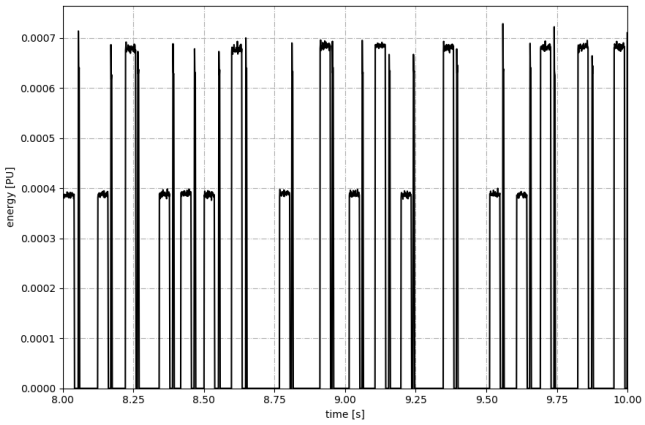
### 5.3.2 *Unsaturated ALOHA and CSMA/CA*



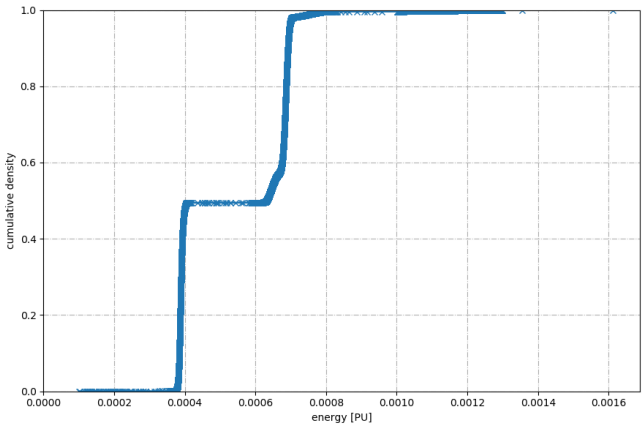

(a) Throughput



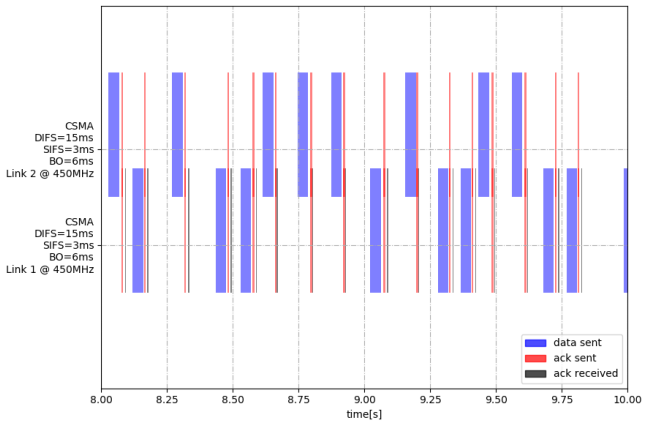

(b) RTT



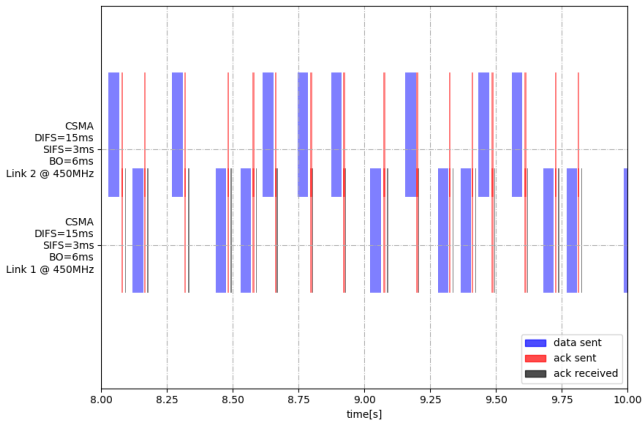

(c) Channel Energy




(d) Channel Energy CDF

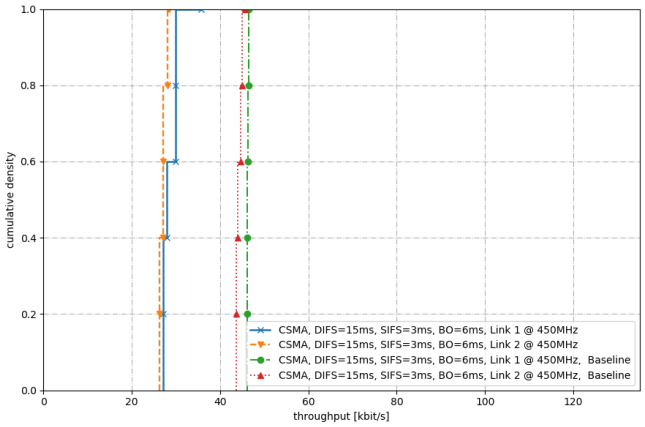



(e) Logical Channel Occupation

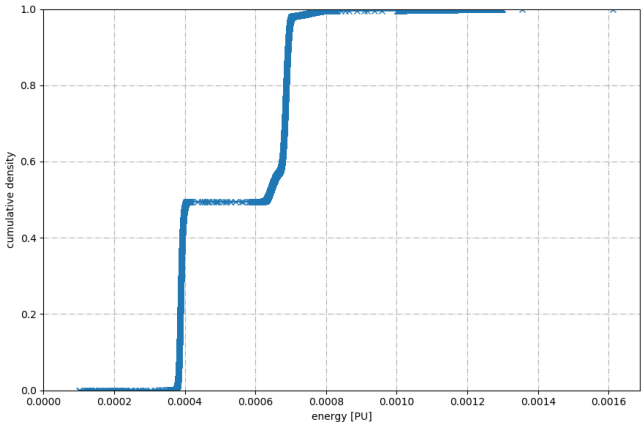



(f) Total Backoff

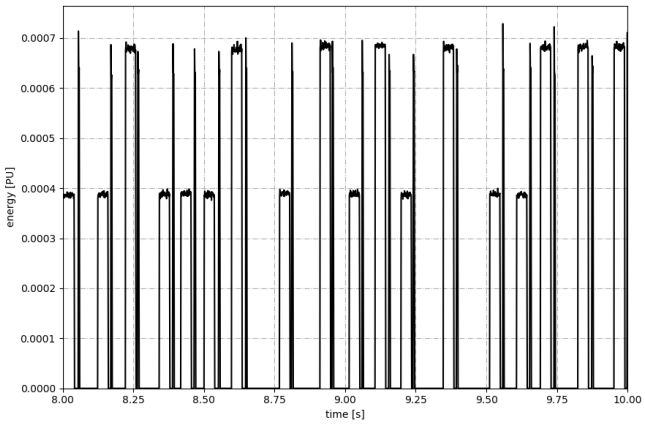
FIGURE 5.8: Measurement results for the CSMA/CA with large parameter set



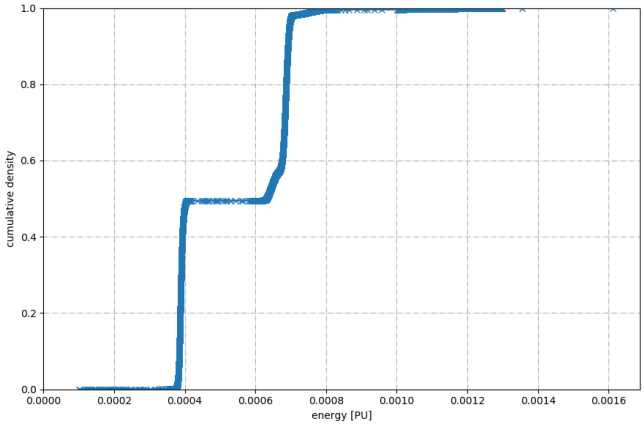

(a) Throughput



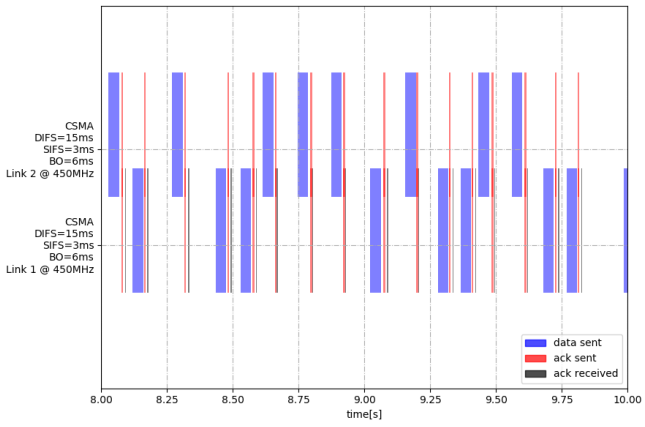

(b) RTT



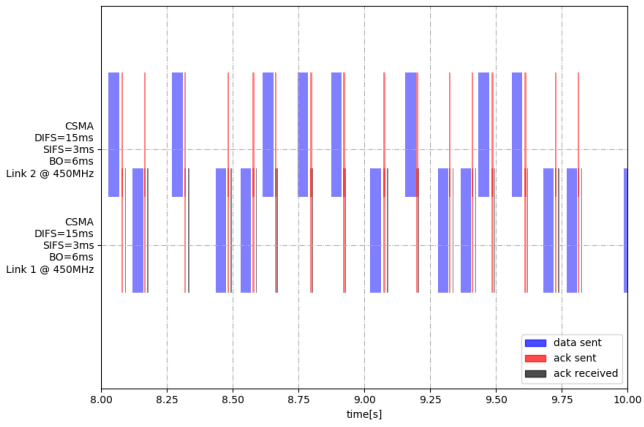

(c) Channel Energy




(d) Channel Energy CDF



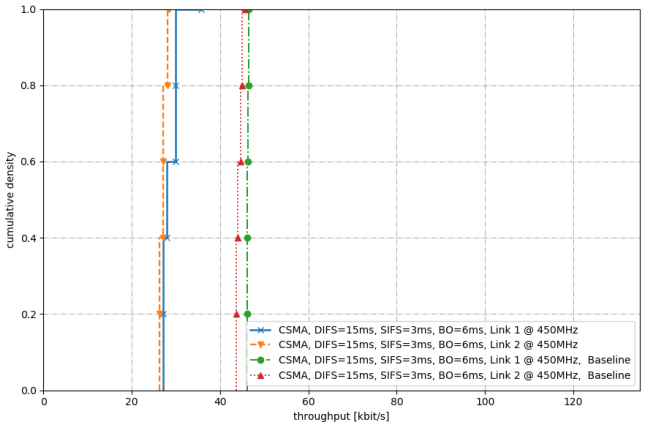

(e) Logical Channel Occupation



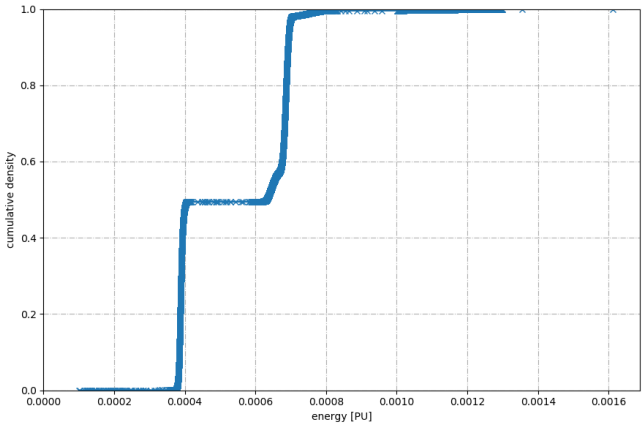

(f) Total Backoff

FIGURE 5.9: Measurement results for the CSMA/CA with large parameter set

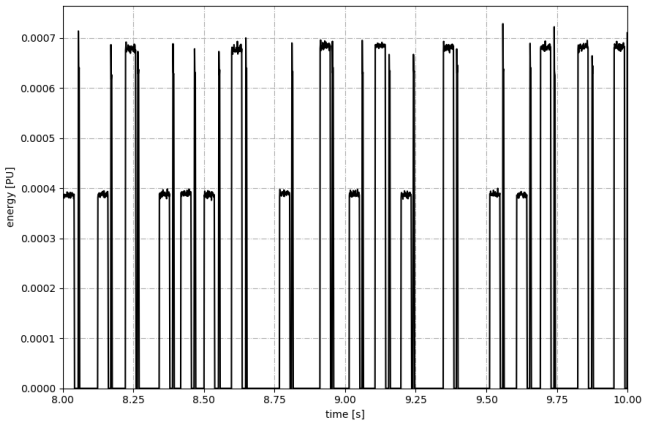
### *5.3.5 1-persistent CSMA and CSMA/CA*



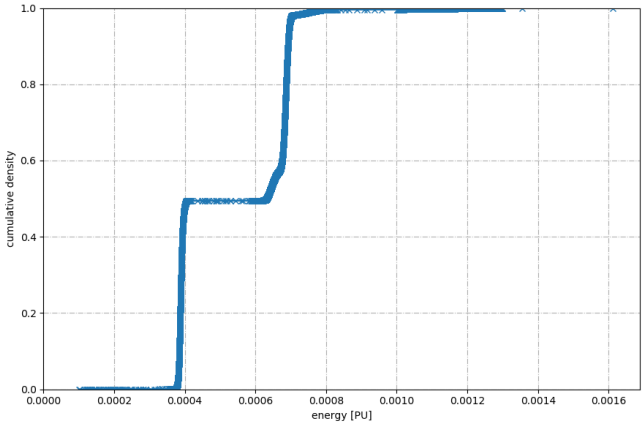

(a) Throughput



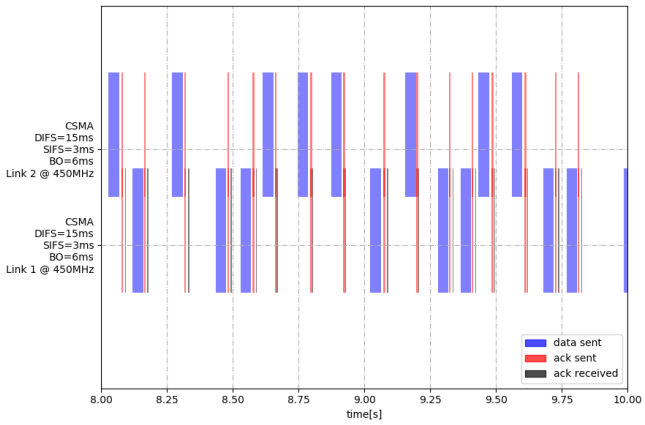

(b) RTT



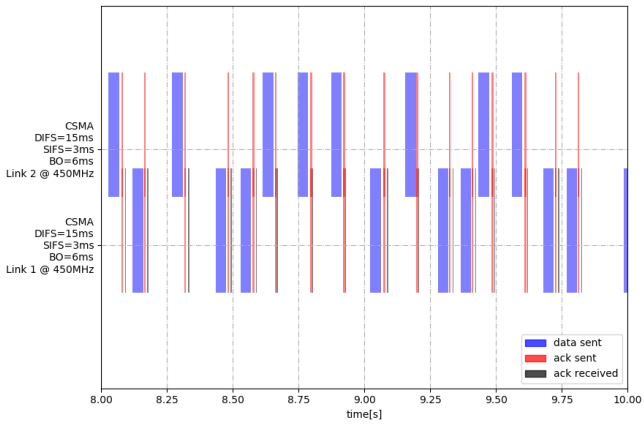

(c) Channel Energy




(d) Channel Energy CDF




(e) Logical Channel Occupation




(f) Total Backoff

FIGURE 5.10: Measurement results for the CSMA/CA with large parameter set

# 6

# Conclusions and Future Work

Conclusions and Future Work here.

# 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 4.4. 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 matplotlib.patches as mpatches
2  import matplotlib.pyplot as plt
3  import numpy as np
4  import os
5  import math
6  import pdb
7  
8  class myplot:
9      def __init__(   self, plottype, data, bins="",
10                     title="", xlabel="", ylabel="",
11                     show=False, savepath=False, data_x=None,
       **kwargs
12                 ):
13 
14         print("Hello from myplot_belated.py!")
15 
16         self.data               = np.asarray(data).transpose()
17         self.data_x             = data_x
18 
19         print("Title is '"+title+"'.")
20         # FIXME: Very hackish, but who cares!
21         if  (title == "Retransmissions per Frame"
22             or len(data) == 1
23             or "bar"  in plottype
24             or "hist" in plottype
25             or "broken_barh" in plottype):
26             print("Keeping original data format instead of
       transposing.")
27             self.data               = data
28 
29         self.bins               = bins
30         self.plottype           = plottype
31         self.xlabel             = xlabel
32         self.ylabel             = ylabel
33         # Let's have reasonable figure dimensions
34         self.fig, self.ax       = plt.subplots(figsize=(9,6))
```

```
35          self.kwargs             = kwargs
36          self.grid               = kwargs.get("grid", False)
37          self.legend             = kwargs.get("legend", [])
38          self.xticks             = kwargs.get("xticks", [])
39          self.legend_loc         = kwargs.get("legend_loc", "best")
40          self.annotations_below  = kwargs.get("annotations_below",
    [])
41          self.annotations_other  = kwargs.get("annotations_other",
    [])
42          self.legend_coordinates = kwargs.get("legend_coordinates",
    False)
43          self.timer              = kwargs.get("timer", 300)
44          self.repetitions        = kwargs.get("repetitions", 5)
45          self.eval_mode          = kwargs.get("eval_mode", "belated")
46          self.xlims              = kwargs.get("xlims", False)
47          self.savepath           = savepath,
48          self.plot_pdf           = kwargs.get("plot_pdf", False)
49
50          plottypes = {
51              "hist":         lambda: self.hist(),
52              "line":         lambda: self.line(),
53              "cdf":          lambda: self.cdf(),
54              "pdf":          lambda: self.pdf(),
55              "boxplot":      lambda: self.boxplot(),
56              "debug":        lambda: self.debug(),
57              "bar":          lambda: self.bar(),
58              "broken_barh":  lambda: self.broken_barh(),
59              "line_xy":      lambda: self.line_xy()
60          }
61
62          titles = {
63              "cdf":          "CDF",
64              "line":         "Line Chart",
65              "line_xy":      "Line Chart",
66              "hist":         "Histogram",
67              "pdf":          "PDF",
68              "boxplot":      "Boxplot",
69              "debug":        "Debug",
70              "bar":          "Bar Chart",
71              "broken_barh":  "Gantt Chart"
72          }
73
74          for aplot in plottype:
75              #print("single plot in plottype array is:"+str(aplot))
76              self.title = title+" "+titles[aplot]
77              self.savename = self.title
78              self.title = ""
79              plottypes[aplot]()
80              #set axis limits
81              self.ax.set_ylim(ymin=0)
82              if self.xlims != False and not (aplot in ["boxplot",
    "bar"]):
83                  self.ax.set_xlim(self.xlims[0], self.xlims[1])
```

```
84              else:
85                  self.ax.set_xlim(xmin=0)
86              plt.tight_layout()
87              self.ax.xaxis.grid(self.grid, linestyle="dashdot")
88              self.ax.yaxis.grid(self.grid, linestyle="dashdot")
89              if not aplot == "boxplot" and not aplot ==
    "broken_barh":
90                  if len(np.asarray(self.data).transpose()) ==
    len(self.legend):
91                      if self.legend_coordinates == False:
92                          box = self.ax.get_position()
93                          self.ax.set_position([
94                              box.x0,
95                              box.y0+box.height*0.3,
96                              box.width,
97                              box.height*0.7
98                          ])
99                          self.ax.legend(fancybox=True,
100                                         loc='upper center',
101                                         bbox_to_anchor=(0.5, -0.15))
102                     else:
103                         if self.legend_coordinates[2] != "best":
104                             self.ax.legend(fancybox=True,
105
    loc=self.legend_coordinates[2],
106
    bbox_to_anchor=(self.legend_coordinates[0],
107
    self.legend_coordinates[1]))
108                         else:
109                             self.ax.legend(fancybox=True,loc="best")
110                 else:
111                     print ( "len(self.data) = "
112                             +
    str(len(np.asarray(self.data).transpose()))
113                             + " and len(self.legend) = "
114                             + str(len(self.legend))
115                             +" don't match!")
116             # Add anotations:
117             for annotation in self.annotations_other:
118                 self.ax.annotate(annotation)
119             # Save and show plot
120             if(savepath):
121                 print("***savepath***")
122                 print(savepath)
123                 self.save(savepath, aplot)
124             if(show):
125                 self.show()
126
127             #plt.close(self.fig)
128             self.fig.clear()
129             #print(self.data)
130
```

```
131             self.annotate()
132
133     def bar(self):
134
135         colors = ['steelblue', 'orange', 'green', 'red', 'purple',
    'brown', 'pink']
136         color_repetitions = math.ceil(len(self.data)/len(colors))
137         colors = color_repetitions * colors
138
139         print(self.data)
140         for idx,val in enumerate(self.data):
141             data_points=len(self.data[idx])
142             width=5/data_points
143             index=np.arange( data_points )
144
145             self.ax.bar(left=idx*width+width/2,
146                 height=val,
147                 color=colors[idx],
148                 alpha=0.5
149              )
150
151             self.setLabels(ylabel=self.ylabel,
152                 xlabel="measurement",
153                 title=self.title
154             )
155
156     def broken_barh(self):
157         plot_data = []
158         debug_data = []
159         for index,item in
    enumerate(self.data["occupation_starting"]):
160
    plot_data.append(list(zip(self.data["occupation_starting"][index],
    self.data["occupation_durations"][index])))
161
162         for index,item in enumerate(self.data["acks_received"]):
163
    debug_data.append(list(zip(self.data["acks_received"][index],
    self.data["acks_received_bar_width"][index])))
164
165         print("plot_data:")
166         #print(plot_data)
167         print("debug_data:")
168         #print(debug_data)
169         print("data_len:")
170         data_len = len(plot_data)
171         print(data_len)
172         debug_len = len(debug_data)
173         print("debug_len:")
174         print(debug_len)
175
176         print("data and ack lengths:")
177         for index,item in enumerate(plot_data):
```

```
178             if index % 2 == 0:
179                 print("data index "+str(index)+":")
180             else:
181                 print("ack index "+str(index)+":")
182             print(len(item))
183 
184         for index, item in enumerate(debug_data):
185             print ("debug data index "+str(index)+":")
186             print(len(item))
187             #print(item)
188 
189         self.ax.set_ylim(10, 5*data_len+20)
190         if self.xlims == False:
191             self.ax.set_xlim(0, self.timer*self.repetitions)
192         self.ax.xaxis.grid(self.grid, linestyle="dashdot")
193         self.ax.yaxis.grid(self.grid, linestyle="dashdot")
194 
195         self.ax.set_yticks([x*10+15 for x in
    range(int(data_len/2))])
196         self.xticks = [tick.replace(",", ",\n") for tick in
    self.xticks]
197         self.ax.set_yticklabels([self.xticks[index] for index in
    range(int(data_len/2))])
198 
199         self.setLabels(
200             xlabel="time[s]",
201             title=self.title
202         )
203 
204         colors = ['blue','red','black']
205         transparency=0.7
206 
207         patch_a  = mpatches.Patch(color=colors[0],
    alpha=transparency, label="data sent")
208         patch_b  = mpatches.Patch(color=colors[1],
    alpha=transparency, label='ack sent')
209         patch_c  = mpatches.Patch(color=colors[2],
    alpha=transparency, label="ack received")
210 
211         if self.legend_coordinates[2] != "best":
212             self.ax.legend( handles=[patch_a,patch_b,patch_c],
213                             fancybox=True,
214                             loc=self.legend_coordinates[2],
215 
    bbox_to_anchor=(self.legend_coordinates[0],
216 
    self.legend_coordinates[1]))
217         else:
218             self.ax.legend( handles=[patch_a,patch_b,patch_c],
219                             fancybox=True,
220                             loc="best")
221 
222         for index,item in enumerate(plot_data):
```

```
223             print("Added (data) set with index "+str(index)+" to
    plot.")
224             if index % 2 == 0:
225                 self.ax.broken_barh(item,((index+1)*5+5,13),
    facecolors=colors[0], alpha=0.5)
226             elif (index-1) % 2 == 0: # well else should be enough
    here :)
227                 self.ax.broken_barh(item,((index)*5+5,13),
    facecolors=colors[1], alpha=0.5)
228
229         for index,item in enumerate(debug_data):
230             print("Added (debug) set with index "+str(index)+" to
    plot.")
231             if index % 2 == 0:
232                 self.ax.broken_barh(item,((index+1)*5+5,13),
    facecolors=colors[2], alpha=0.5)
233             elif (index-1) % 2 == 0:
234                 self.ax.broken_barh(item,((index)*5+5,13),
    facecolors=colors[2], alpha=0.5)
235
236         plt.tight_layout()
237
238     def line(self):
239         from scipy.interpolate import interp1d
240         x = np.arange(1, len(self.data)+1, 1)
241         y = self.data
242         f = interp1d(x,y)
243         plt.plot(x, f(x), 'k')
244
245         self.setLabels( xlabel="measurement",
246                         ylabel=self.ylabel,
247                         title=self.title
248         )
249
250     def line_xy(self):
251         from scipy.interpolate import interp1d
252         x = self.data_x
253         y = self.data
254         f = interp1d(x,y)
255         plt.plot(x, f(x), 'k')
256         self.setLabels( xlabel=self.xlabel,
257                         ylabel=self.ylabel,
258                         title=self.title
259         )
260
261     def hist(self):
262         self.n, self.bins, self.patches = self.ax.hist(x=self.data,
263                         bins=self.bins,
264                         normed=1,
265                         histtype='step',
266                         cumulative=True,
267                         label=self.legend)
268         self.setLabels( xlabel=self.xlabel,
```

```
269                         ylabel="cumulative density",
270                         title=self.title)
271         print(self.patches)
272 
273     def cdf(self):
274         #print(self.data)
275         print(self.legend)
276         markers = ["x","v","o","^","8","s","p","+","D","*"]
277         linestyles = ["-", "--", "-.", ":","-", "--", "-.",
    ":","-", "--"]
278         linewidths = [1.8,1.65,1.5,1.35,1.2,1.05,1,0.9,0.8,0.75]
279 
280         if self.eval_mode == "belated":
281             cdf_data = np.asarray(self.data).transpose()
282         if self.eval_mode == "live":
283             cdf_data = np.asarray(self.data).transpose()
284             if self.title == "Retransmissions per Frame":
285                 cdf_data = [cdf_data]
286         if len(self.data) == 1:
287             print("Hooray, my title is "+self.title+".")
288             cdf_data = self.data
289 
290         if len(cdf_data) > 1:
291             for index,item in enumerate(cdf_data):
292                 print("index:"+str(index))
293                 print("___markers____")
294                 print(markers[index])
295                 x = np.sort(item)
296                 y = np.arange(1,len(x)+1) / len(x)
297                 x = np.insert(x,0,x[0])
298                 y = np.insert(y,0,0)
299                 self.plot = plt.step(x,
300                     y,
301                     marker=markers[index],
302                     linestyle=linestyles[index],
303                     linewidth=linewidths[index],
304                     markevery=range(1,len(x)),
305                     label=self.legend[index])
306         else:
307             x = np.sort(cdf_data[0])
308             y = np.arange(1,len(x)+1) / len(x)
309             x = np.insert(x,0,x[0])
310             y = np.insert(y,0,0)
311             print(x)
312             print(y)
313             self.plot = plt.step(x,
314                     y,
315                     marker=markers[0],
316                     linestyle=linestyles[0],
317                     linewidth=linewidths[0],
318                     markevery=range(1,len(x)),
319                     label=self.legend[0])
320 
```

```
321
322         self.setLabels( xlabel=self.xlabel,
323                         ylabel="cumulative density",
324                         title=self.title)
325         self.ax.set_ylim(ymax=1)
326
327     def pdf(self):
328         self.n, self.bins, self.patches = self.ax.hist(x=self.data,
329                         bins=self.bins,
330                         align='left',
331                         fill='true',
332                         normed=1,
333                         cumulative=False,
334                         label=self.legend)
335         self.setLabels( xlabel=self.xlabel,
336                         ylabel="probability density",
337                         title=self.title)
338
339         cm = plt.cm.get_cmap('jet')
340         for index, patch in enumerate(self.patches):
341             plt.setp(patch, 'facecolor',
    cm(float(index/len(self.patches))))
342
343     def boxplot(self):
344         print(self.data)
345
346         self.plot = plt.boxplot(self.data,
347                                 #notch=True,
348                                 patch_artist=True,
349                                 flierprops=dict(marker='x'))
350
351         colors = ['steelblue', 'peachpuff', 'green', 'red',
    'purple', 'brown', 'pink']
352         color_repetitions = math.ceil(len(self.data)/len(colors))
353         colors = color_repetitions * colors
354
355         for patch, color in zip(self.plot['boxes'], colors):
356             patch.set_facecolor(color)
357
358         self.setLabels( xlabel="measurement",
359                         ylabel=self.ylabel,
360                         title=self.title)
361
362         boxdict = self.ax.boxplot(self.data)
363         fliers  = boxdict["fliers"]
364
365         for j in range(len(fliers)):
366             yfliers = boxdict['fliers'][j].get_ydata()
367             xfliers = boxdict['fliers'][j].get_xdata()
368             ufliers = set(yfliers)
369             for i, uf in enumerate(ufliers):
370                 self.ax.text(xfliers[i] + 0.05, uf,
    list(yfliers).count(uf))
```

```
371
372         if len(np.asarray(self.data).transpose()) ==
    len(self.xticks):
373             print(self.xticks)
374             plt.xticks([x+1 for x in
    range(len(self.xticks))],self.xticks)
375             #self.ax.set_xticklabels(self.xticks);
376         else:
377             print ( "len(self.data) = "
378                     + str(len(self.data))
379                     + " and len(self.xticks) = "
380                     + str(len(self.xticks))
381                     +" don't match!")
382
383     def setLabels(self, xlabel="", ylabel="", title=""):
384         self.ax.set_xlabel(xlabel)
385         self.ax.set_ylabel(ylabel)
386         self.ax.set_title(title)
387
388     def save(self, savepath, plot_type):
389         #savename = self.title
390         savename = self.savename
391         savename = savename.lower()
392         savename = savename.replace(" ", "_")
393         self.fig.savefig(savepath+savename+".png")
394         if self.plot_pdf:
395             self.fig.savefig(savepath+savename+".pdf")
396
397     def show(self):
398         plt.show()
399
400     def annotate(self):
401         pass
402
403     def debug(self):
404         print("data: "+str(self.data))
405         print("bins: "+str(self.bins))
406         print("plottype: "+self.plottype)
```

LISTING A.3: 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] "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.

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

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

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

[5] 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.

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

[7] 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.

[8] 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.

[9] "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.

[10] 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.

[11] 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

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

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

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

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

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

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

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

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

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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

[26] "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