

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

ACKNOWLEDGEMENTS	II
CONTENTS	III
ABSTRACT	V
1 INTRODUCTION	1
2 BACKGROUND	2
2.1 MAC PROTOCOLS	2
2.1.1 MAC LAYER IN THE OSI MODEL	2
2.1.2 CHALLENGES FOR WIRELESS MAC PROTOCOLS	2
2.1.3 CLASSIFICATION OF MAC PROTOCOLS	4
2.1.4 RESERVATION-BASED MAC PROTOCOLS	4
2.1.5 CONTENTION-BASED MAC PROTOCOLS	5
2.1.6 DUTY-CYCLE MAC PROTOCOLS	10
2.2 SOFTWARE-DEFINED RADIO	11
2.2.1 PURPOSE OF SOFTWARE-DEFINED RADIO	11
2.2.2 WHAT IS GNU RADIO?	12
2.2.3 CORE CONCEPTS OF GNU RADIO	13
3 RELATED WORK	15
3.1 LTE-U/WiFi COEXISTENCE STUDIES	15
3.2 CONNECTION TO OUR WORK	16
4 MEASUREMENT METHODOLOGY	17
4.1 MEASUREMENT SETUP	17
4.2 GNU RADIO FLOWGRAPHS	17
4.2.1 RECEIVER AND SNIFFER	17
4.2.2 PURE ALOHA TRANSMITTER	19
4.2.3 CSMA TRANSMITTER	19
4.3 MEASUREMENT METRICS	23
4.3.1 THROUGHPUT	23
4.3.2 ROUND-TRIP TIME	23
4.3.3 PACKET LOSS AND RETRANSMISSIONS PER FRAME	24
4.3.4 BACKOFF TIMES	24

4.3.5	PACKET DURATIONS & CHANNEL OCCUPATION	24
4.3.6	CHANNEL ENERGY LEVEL	25
4.4	MEASUREMENT SCRIPT SYSTEM	25
4.5	QUALITY STANDARDS	27
5	MEASUREMENT RESULTS	28
5.1	MAC PROTOCOLS	28
5.2	SAME PROTOCOL COMBINATIONS	28
5.2.1	ALOHA	29
5.2.2	CSMA/CA WITH HIGH PARAMETER SET	30
5.2.3	CSMA/CA WITH LOW PARAMETER SET	31
5.2.4	CSMA/CA WITH MEDIUM PARAMETER SET	32
5.2.5	1-PERSISTENT CSMA	34
5.3	DIFFERENT PROTOCOL COMBINATIONS	36
5.3.1	ALOHA AND CSMA/CA	36
5.3.2	UNSATURATED ALOHA AND CSMA/CA	37
5.3.3	INHOMOGENEOUS CSMA/CA	39
5.3.4	1-PERSISTENT CSMA AND UNSATURATED ALOHA	41
5.3.5	1-PERSISTENT CSMA AND CSMA/CA	43
6	CONCLUSIONS AND FUTURE WORK	44
A	BASH AND PYTHON SCRIPTS	45
B	ABBREVIATIONS	61
	LIST OF FIGURES	64
	LIST OF TABLES	66
	BIBLIOGRAPHY	67
	DECLARATION	68

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

1

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

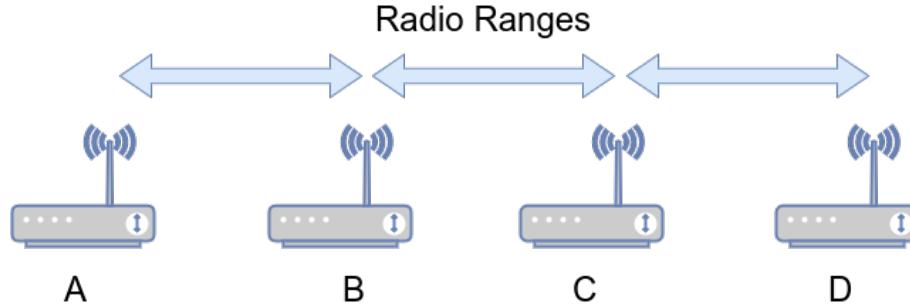


FIGURE 2.1: Setup to explain the hidden and exposed node problem. Each node can only reach its direct neighbors.

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]

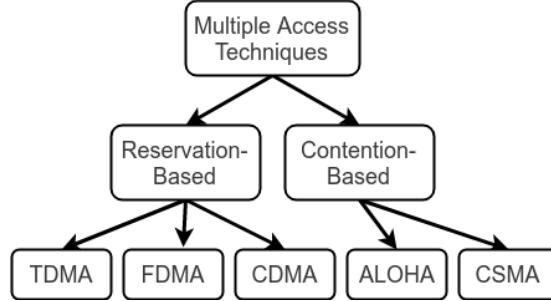


FIGURE 2.2: Classification of MAC techniques

(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 special¹ 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:

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

¹by 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 $\frac{\partial S}{\partial G} \stackrel{!}{=} 0$:

$$\frac{\partial S}{\partial G} = \frac{\partial}{\partial G} Ge^{-2G} \quad (2.2)$$

$$= e^{-2G}(1 - 2G) \quad (2.3)$$

$$\stackrel{!}{=} 0 \quad (2.4)$$

$$\Leftrightarrow G = 0.5 \quad (2.5)$$

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

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

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

2.1.5.2 CSMA

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.

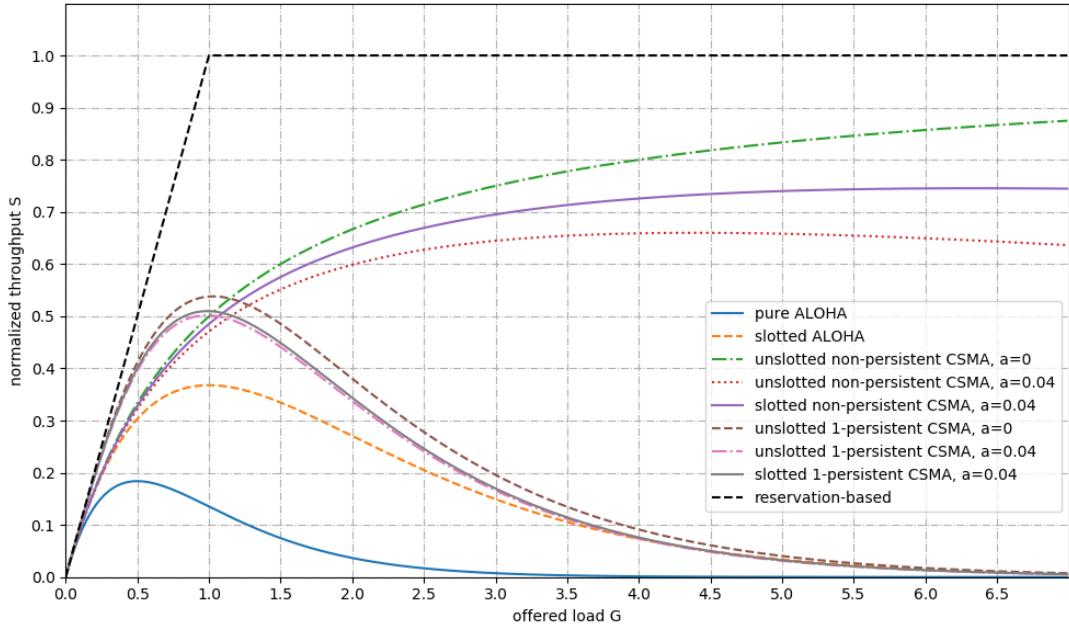


FIGURE 2.3: Normalized throughput over offered load according to formulae in [3], [6], [7], , with $a = \tau/T_p$, where τ is the maximum propagation delay and T_p the packet transmission time and under the assumptions made in Section 2.1.5.1.

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-persistent 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

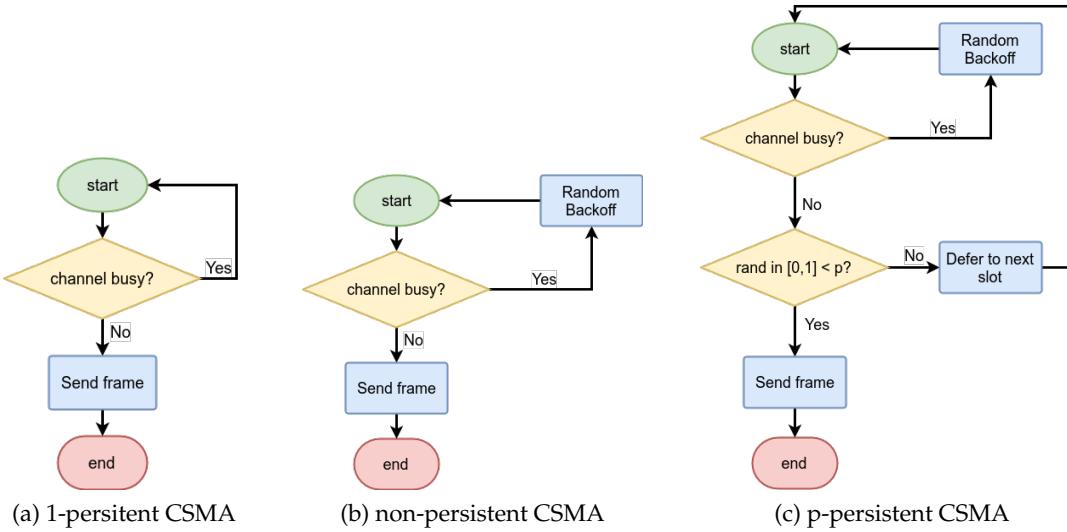


FIGURE 2.4: The three basic flavors of CSMA

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

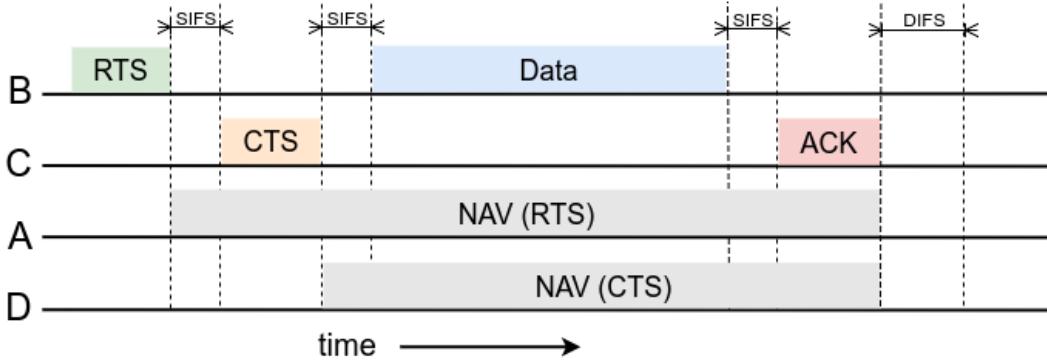


FIGURE 2.5: Virtual carrier sensing in CSMA/CA, as described in [3] and [4]

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 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 $2^{n+m} - 1$, where the whole BEB procedure is repeated with n incremented, thus the CW doubled, hence **binary** exponential backoff. The

motivation for CW_{\min} is to 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

Generally², 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/CA³ 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.

²for both license assisted access (LAA) as well as LTE-U

³actually 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 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.

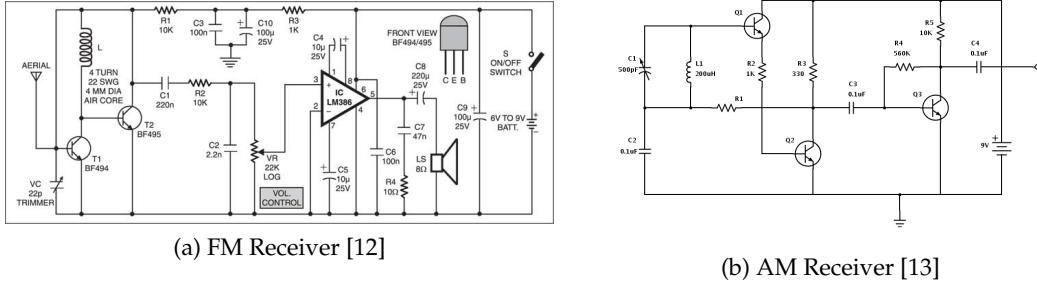


FIGURE 2.6: Simple do-it-yourself (DIY) radio receiver circuit diagrams

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].

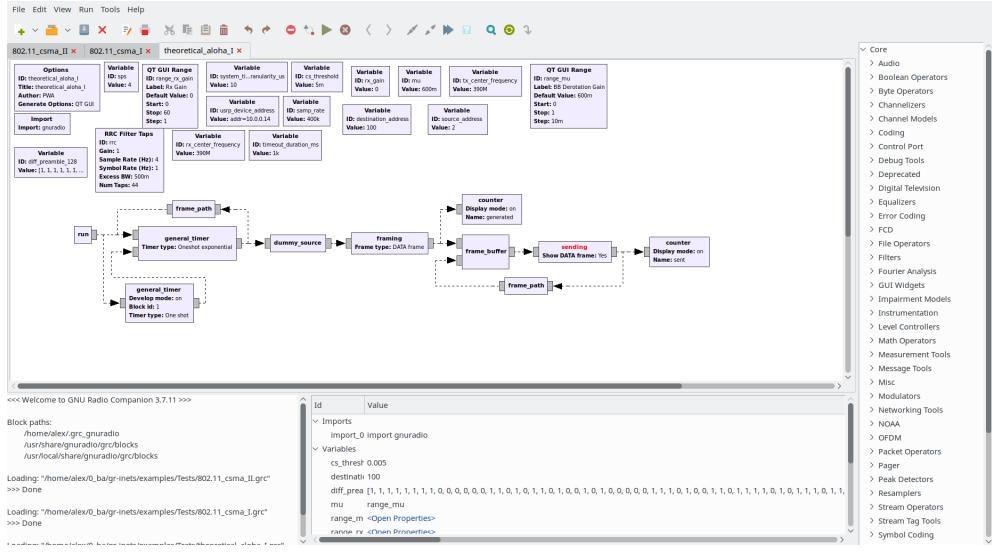


FIGURE 2.7: GNU Radio Companion GUI

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-Miguel 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 Broadcom.

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 cable. 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 timeout¹ 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`

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

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

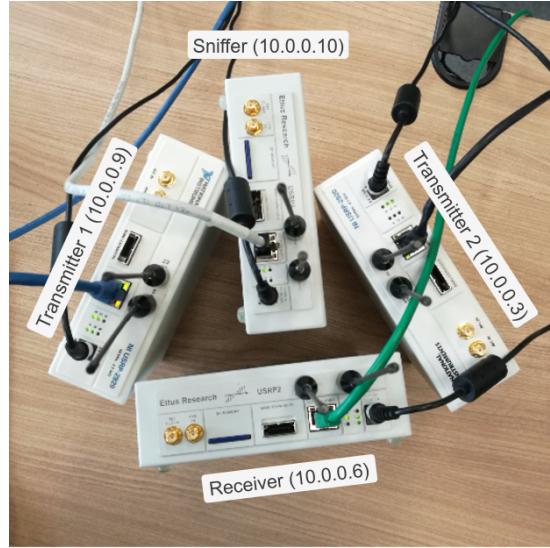


FIGURE 4.1: Photo of the measurement setup.

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

TABLE 4.1: 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 received 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 exchange.

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

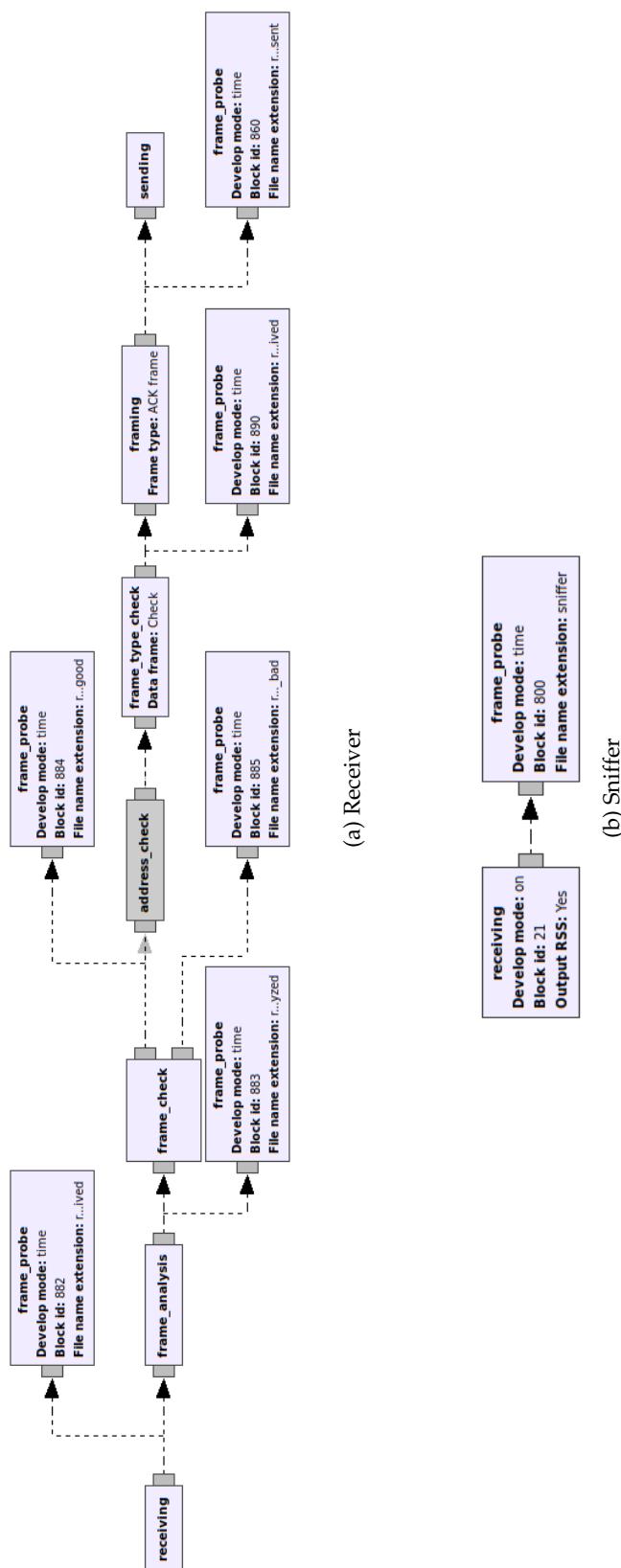


FIGURE 4.2: GRC Receiver Flowgraphs

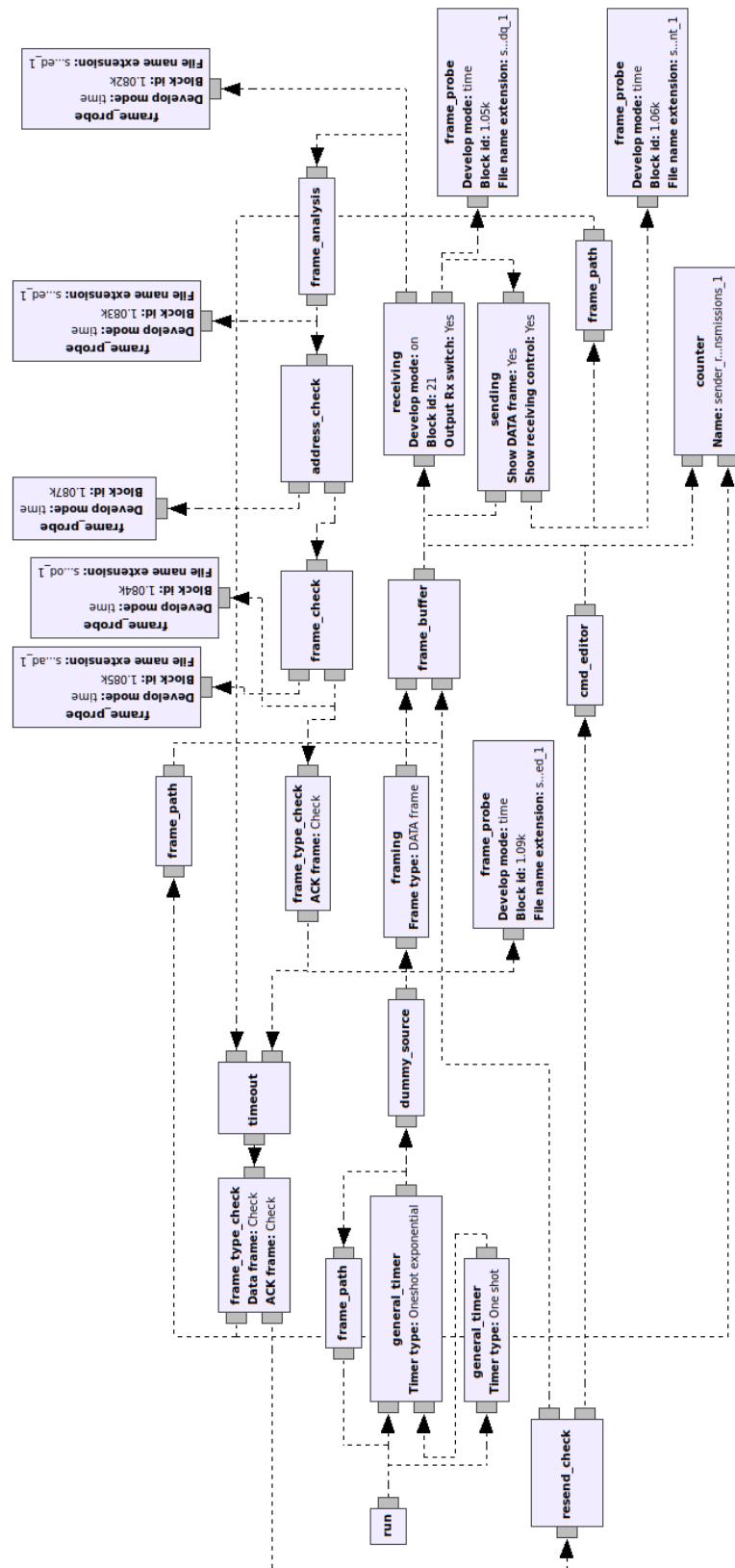


FIGURE 4.3: GRC Pure ALOHA Transmitter Flowgraph

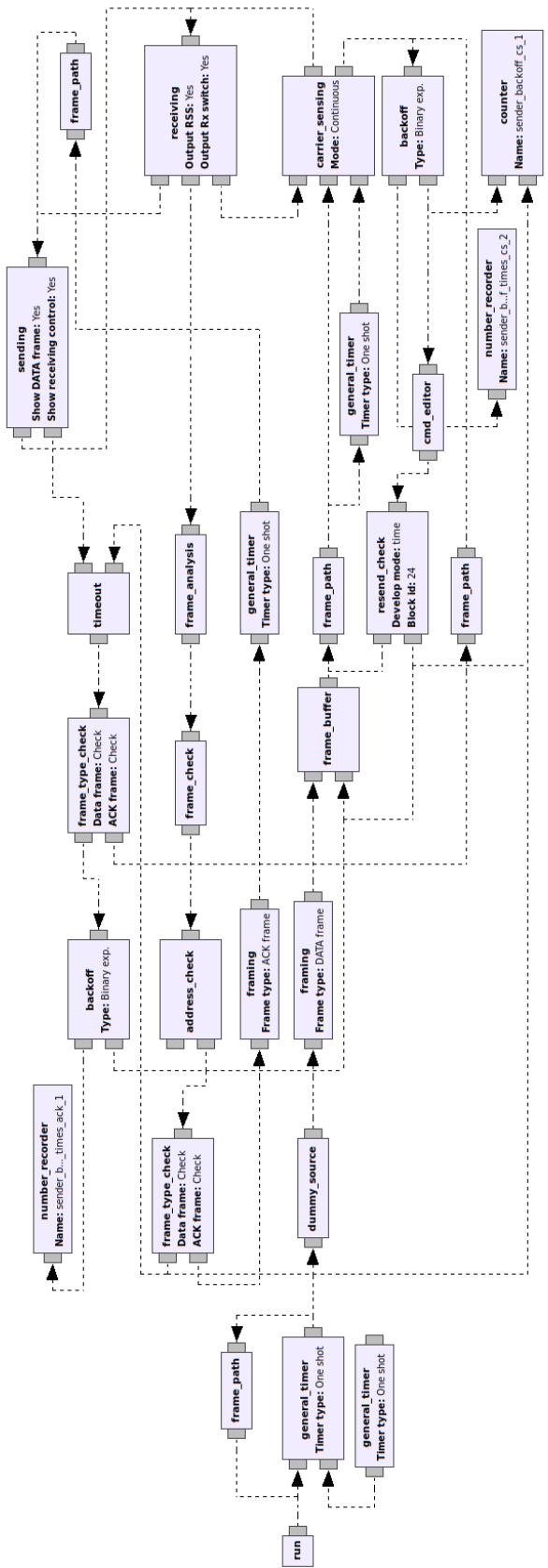


FIGURE 4.4: GRC CSMA Transmitter Flowgraph

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 data⁴ 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

```

⁴payload 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 = 1
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 $1 - \frac{n_{ACKs}}{n_{data}}$, where n_{ACKs} and n_{data} are the number of ACK packets received and data packets sent by the transmitter. Retransmissions per frame are obtained simply by recording them with a `counter` block, which is incremented whenever the `timeout` runs out and reset when an ACK is received.

4.3.4 *Backoff Times*

The script `backoff.py` sums up three different backoff times: Firstly, the backoff due to failed⁵ 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 length⁶. 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

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

⁶E.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 occupation⁷. 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.

```

1 measurement      = [714, 715, 728, 646]
2 links           = [1, 2, 1, 2]
3 boxplot_xticks = [
4     "CSMA\nDIFS=15ms\nSIFS=0ms\nBO=0ms\nLink 1 @ 450MHz",
5     r"unsaturated ALOHA ~ Poisson($1/\lambda=200ms$), Link 2 @
6     450MHz",
7     "CSMA\nDIFS=15ms\nSIFS=0ms\nBO=0ms\nLink 1 @ 450MHz\n Baseline",
8     "unsaturated ALOHA\nLink 2 @ 450MHz\n Baseline",
8 ]

```

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).

⁷Frame 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.

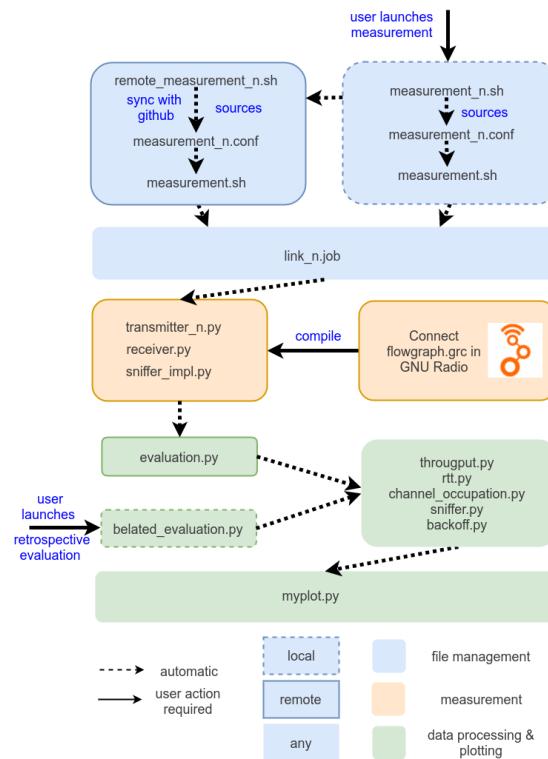


FIGURE 4.5: The three-phase measurement script system

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`.

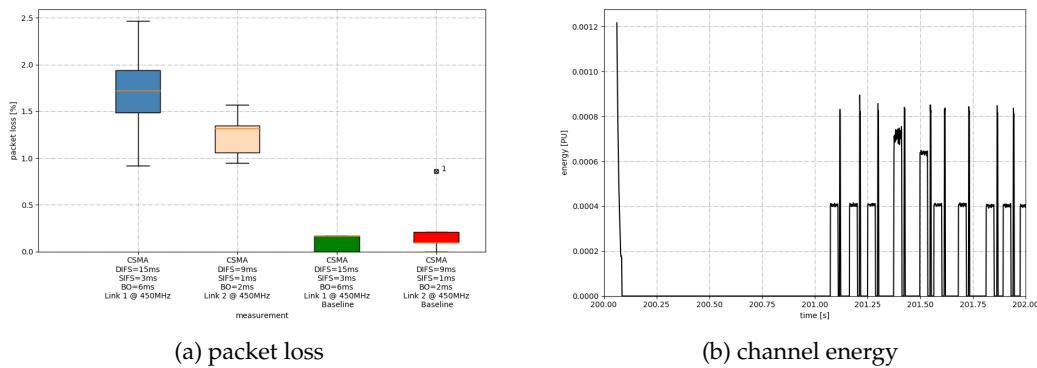


FIGURE 4.6: Plots depicting the quality standards of our measurement

4.5 QUALITY STANDARDS

Statistical Reliability Each measurement features five repetitions⁸ of the same à 100 seconds. Compared to a single measurement of 500 seconds this has the downside that script and initialization (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 quality assurance 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.

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

⁹Of the 1.1 seconds less than 100ms are script initialization overhead since as soon as 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. We first assess the measurement results where both senders employ the same MAC protocols. Subsequently, we will do the same for different combinations of MAC protocols.

5.1 MAC PROTOCOLS

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. We set the durations of SIFS, DIFS and BO to a scaled versions of the

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 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.

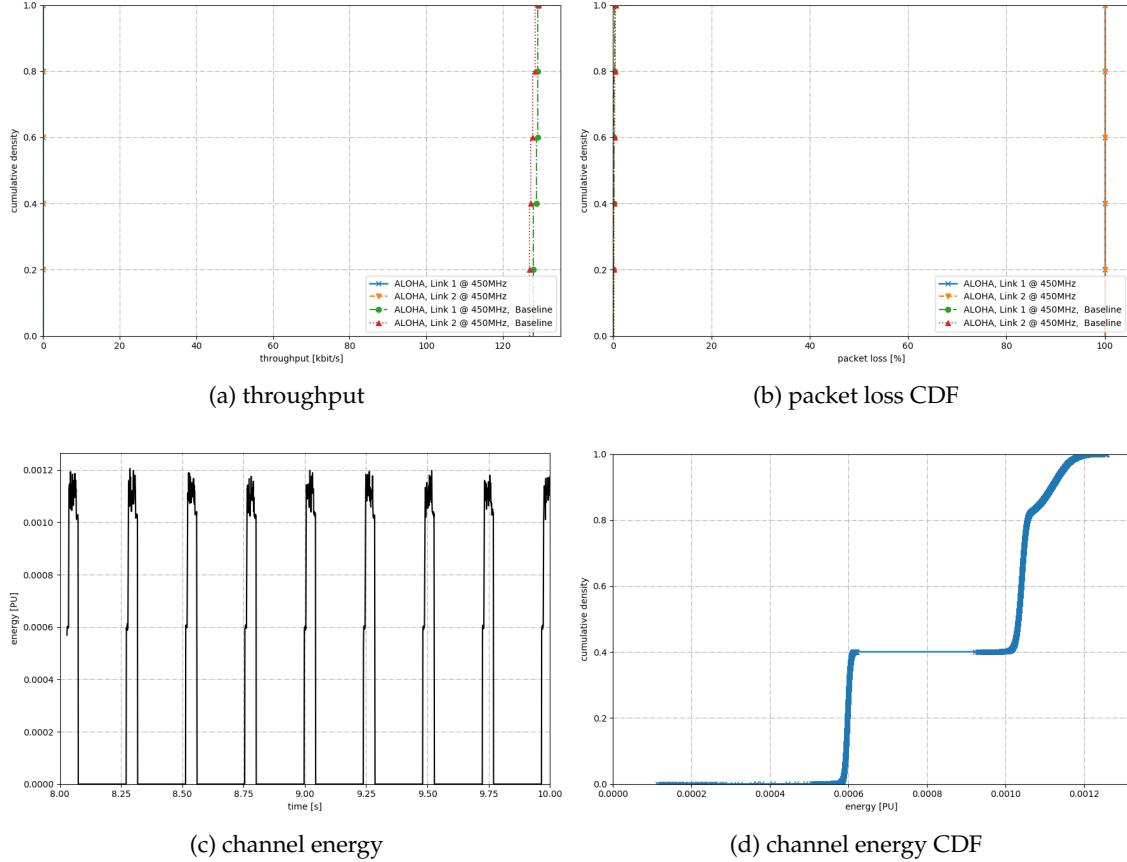


FIGURE 5.1: Measurement results for two ALOHA senders in the same channel

5.2.1 ALOHA

For two links with saturated ALOHA traffic we expect zero aggregate 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 value that can be read off Subfigure (a) 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 do not 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.

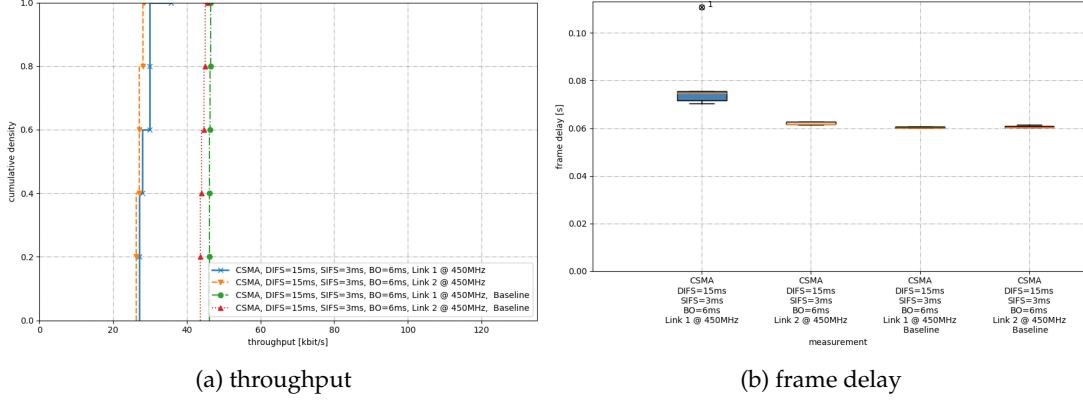


FIGURE 5.2: Measurement results for the CSMA/CA with high parameter set

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 we 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 (b) shows that the frame delay roughly stays the same, where the deviation of the first link comes from packet loss related to hardware problems as described in Section 4.5. Figure 5.3 illustrates the same from the sniffer's point of view (POV), as the even throughput among senders is reflected in Subfigure (b) with the two observed energy levels of the senders being 0.0004 PU and 0.0007 PU, which is confirmed by Subfigure (a). Figure 5.4 shows the cumulative backoff times and the

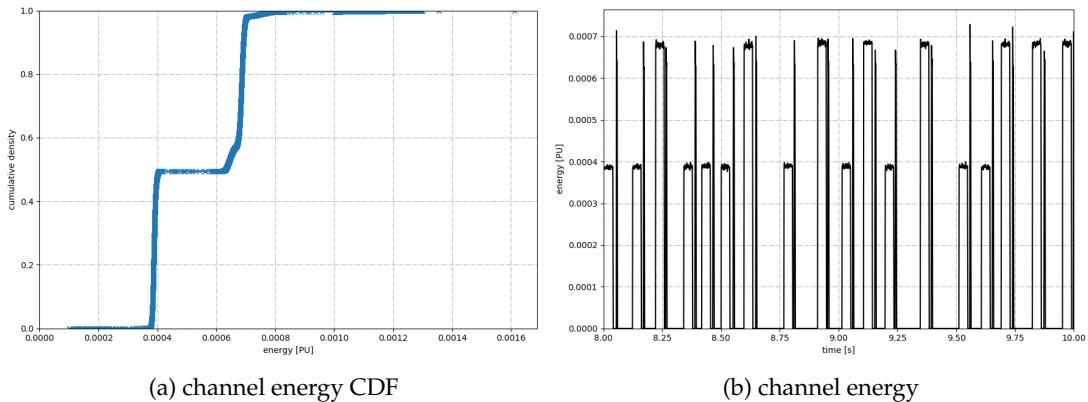


FIGURE 5.3: Energy in the channel detected by the sniffer for CSMA/CA with high parameter set.

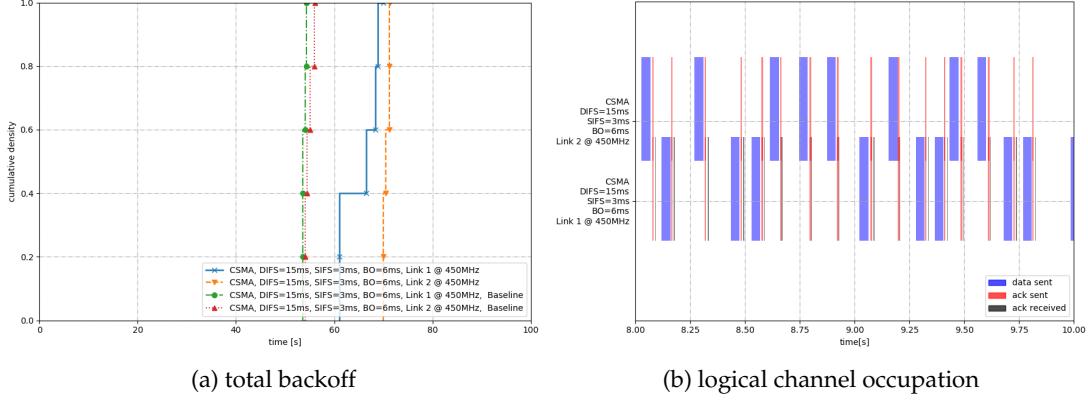


FIGURE 5.4: Measurement results for the CSMA/CA with high parameter set

logical channel occupation which corresponds to the channel energy plot in Figure 5.3 (a).

5.2.3 CSMA/CA With Low Parameter Set

The next thing we were interested in is to what extent we can scale down DIFS, SIFS and BO and still retain collision-free transmission. To this end, we reduced the values to DIFS=5ms, SIFS=1ms, BO=2ms. Reducing these values, especially BO increases the throughput, with $CW_{\min} = 32 \cdot BO$ and uniformly distributed random choice of the backoff slot, we expect a mean delay of $16 \cdot BO = 32ms$ in the first backoff round, which is rather large compared to DIFS and SIFS.

Indeed, comparing throughput using the high parameter set (Figure 5.2(a)) with the low parameter set (Figure 5.5(a)) yields a little less than doubled throughput for cutting the backoff. A problem are collisions of ACKs and consecutive data frames of

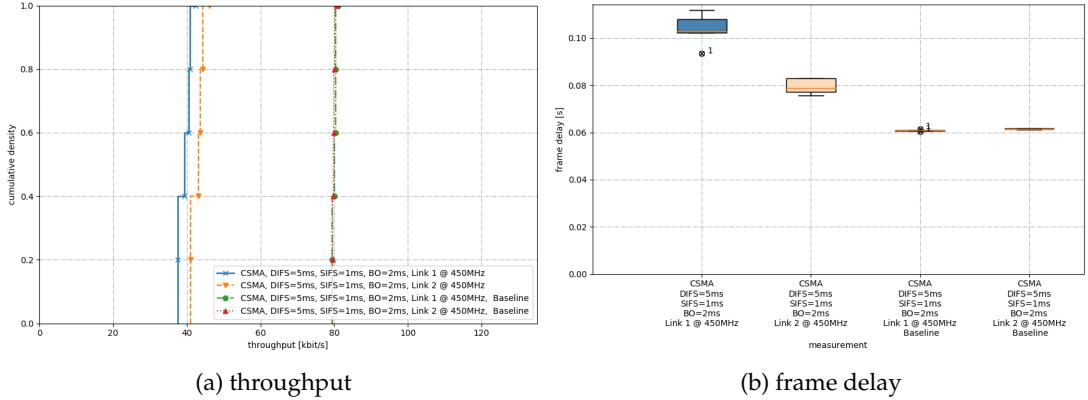


FIGURE 5.5: Measurement results for the CSMA/CA with low parameter set

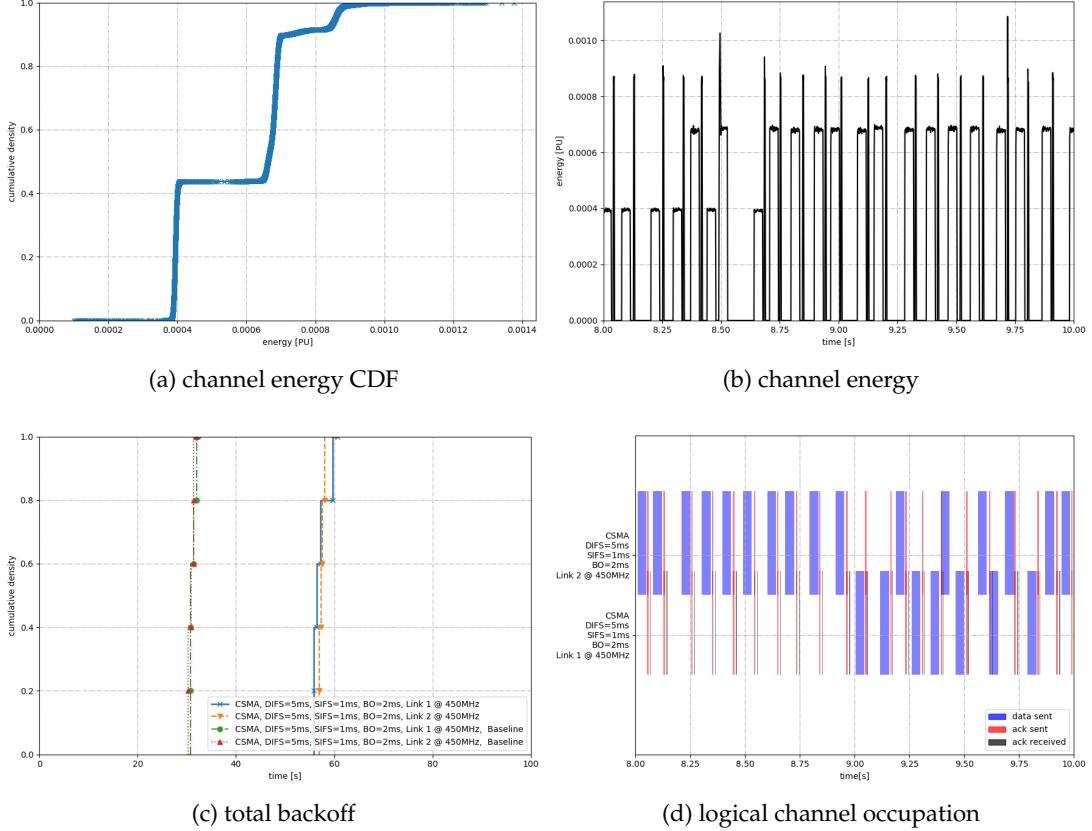


FIGURE 5.6: Measurement results for the CSMA/CA with low parameter set

the sender to whom the ACK was not destined caused by the low DIFS in conjunction with hardware delays. Multiple of these collisions occur in the time window depicted in Figure 5.6(b). These collisions lead to the increased frame delays of Figure 5.5(b), where again the additional packet loss of link 1 reflected in the 20ms increased frame delay compared to link 1 is caused by the hardware problems described in Section 4.5.

5.2.4 CSMA/CA With Medium Parameter Set

Due to the collisions of ACKs with the data packets of sender 1 as described in the preceding Section 5.2.3 we increased DIFS to 9 ms, kept SIFS to 1 ms and BO to 2ms and indeed, as shown in Figure 5.7(b) the frame delay of link 2 is close to the baseline level as result of avoiding collisions. However, since collisions as a result of low DIFS occurred quite infrequently in the last scenario, the metrics roughly stay the same as in the previous scenario, as can be seen in all other subfigures.

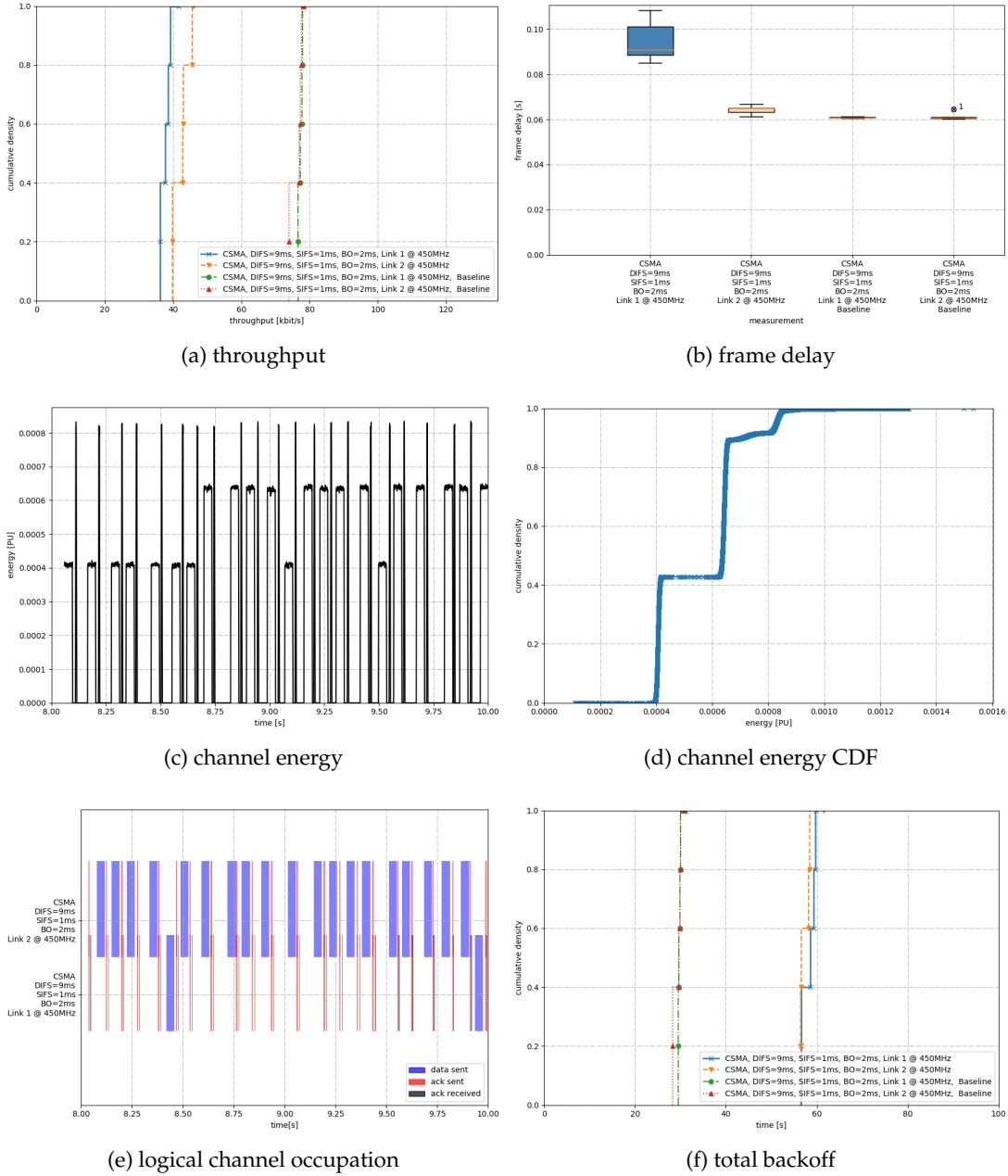


FIGURE 5.7: Measurement results for the CSMA/CA with medium parameter set

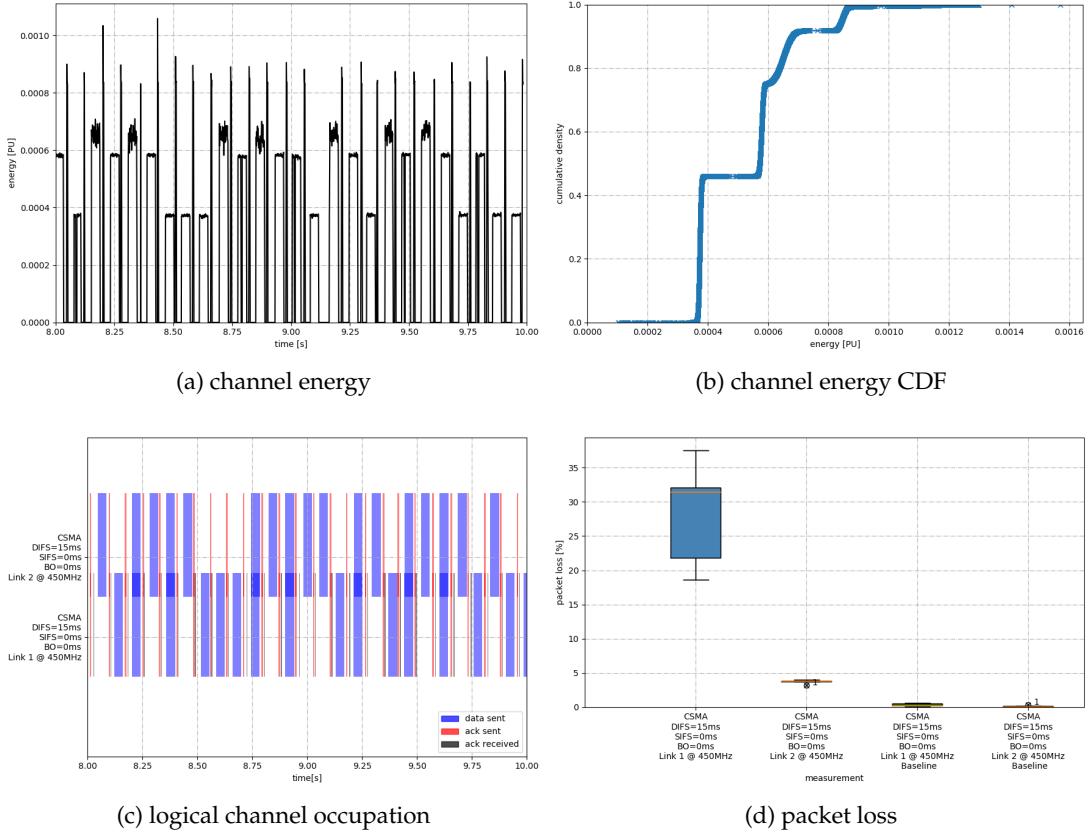


FIGURE 5.8: Measurement results for 1-persistent CSMA

5.2.5 1-persistent CSMA

In the following experiment we examined if a fixed sensing duration of DIFS is enough for harmonious coexistence. As can be seen in Figure 5.8(c) we encounter the problem described in Section 2.1.5.2, namely that both¹ senders try to seize the channel at the same time once a sender finished their transmission. If the time granularity of the system was finer, that is to say its timing accuracy was even higher none of the nodes would start to transmit slightly earlier, leading to even further deteriorated throughput than in Figure 5.9(a). The higher packet loss (Figure 5.8(d)) and thus higher frame delay (Figure 5.9(b)) of link 1 compared to link 2 can be elucidated by the fact that the signal-to-interference-plus-noise ratio (SINR) ratio of sender 2 is bigger than the SINR of sender 1² as can be surmised from Figure 5.8(a). The extra bend in the channel energy CDF (Figure 5.8(b)) roughly from 0.0006 to 0.0007 PU is a consequence of the interference which is also very visible in Figure 5.8(a).

¹actually three, since the receiver is transmitting ACKs

²Also, the SINR of the receiver's ACK signal is higher than all others.

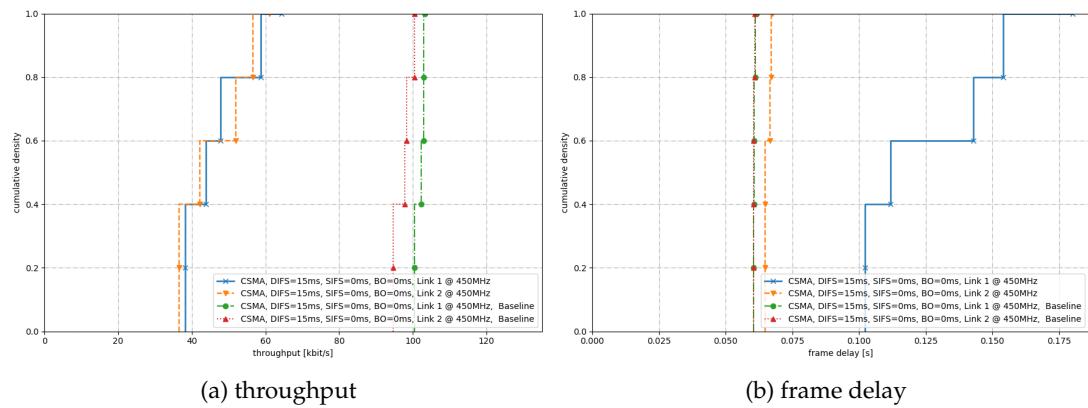


FIGURE 5.9: Measurement results for 1-persistent CSMA

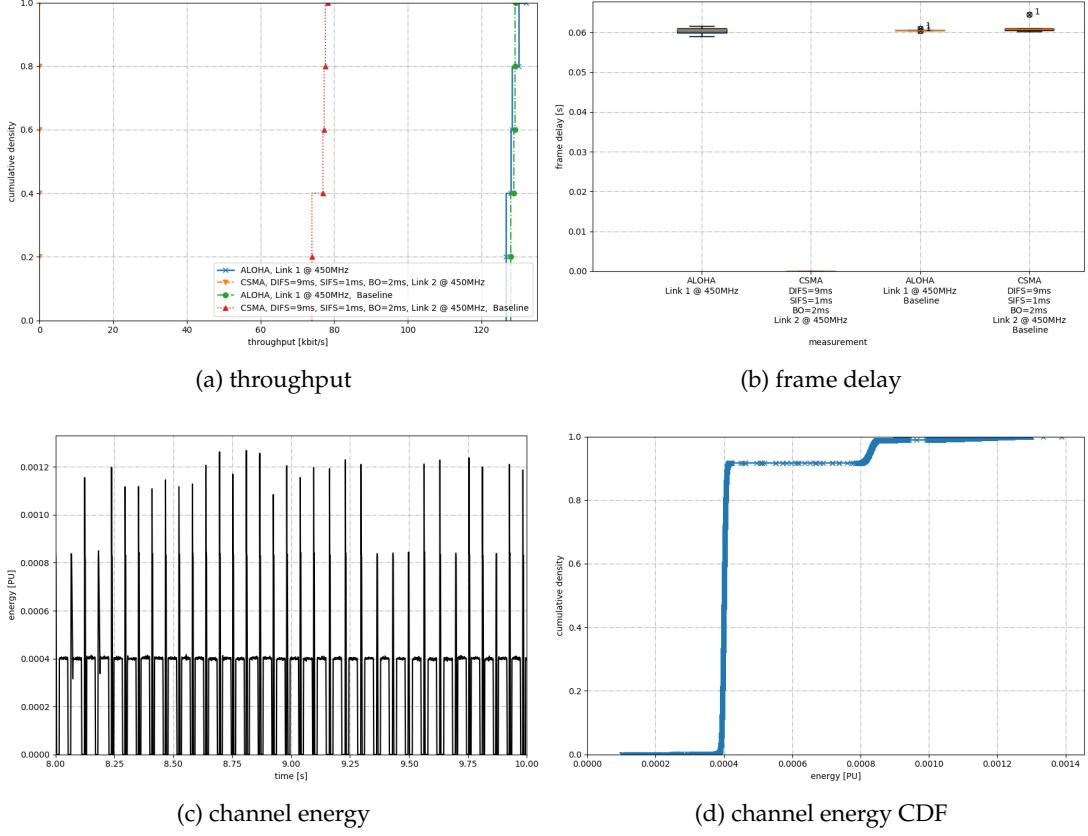


FIGURE 5.10: Measurement results for saturated ALOHA and CSMA/CA (medium parameter set) simultaneously active in the same channel

5.3 DIFFERENT PROTOCOL COMBINATIONS

5.3.1 ALOHA and CSMA/CA

In this experiment we aimed at an experimental confirmation that saturated ALOHA traffic would cause CSMA/CA to stay silent during the whole measurement time. This result is confirmed by every plot in Figure 5.10. The throughput and frame delay³ (Subfigures (a) and (b)) of CSMA/CA are zero, whereas for saturated ALOHA node throughput and frame delay almost perfectly match the baseline values. The channel energy plots in Subfigures (c) and (d) show that only link 1 and the receiver are transmitting during a time window of 2 seconds, which also is generally true for the whole measurement duration.

³since no frame has ever been sent by the CSMA/CA node

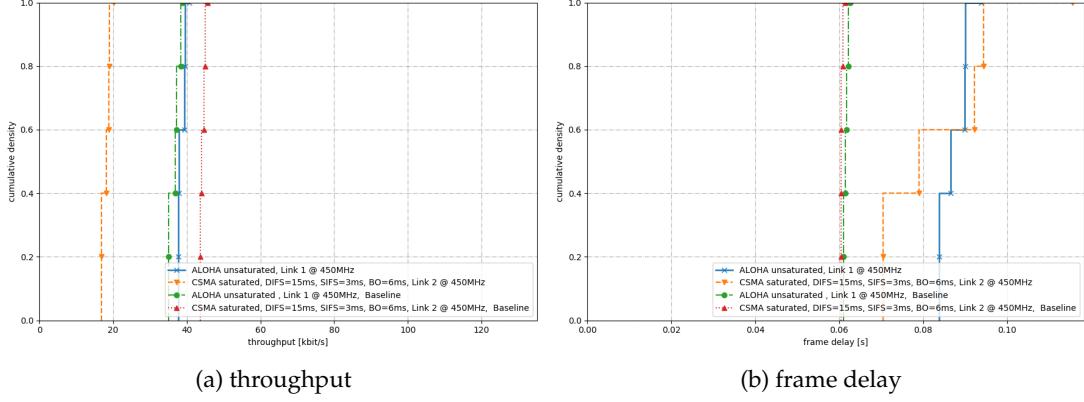


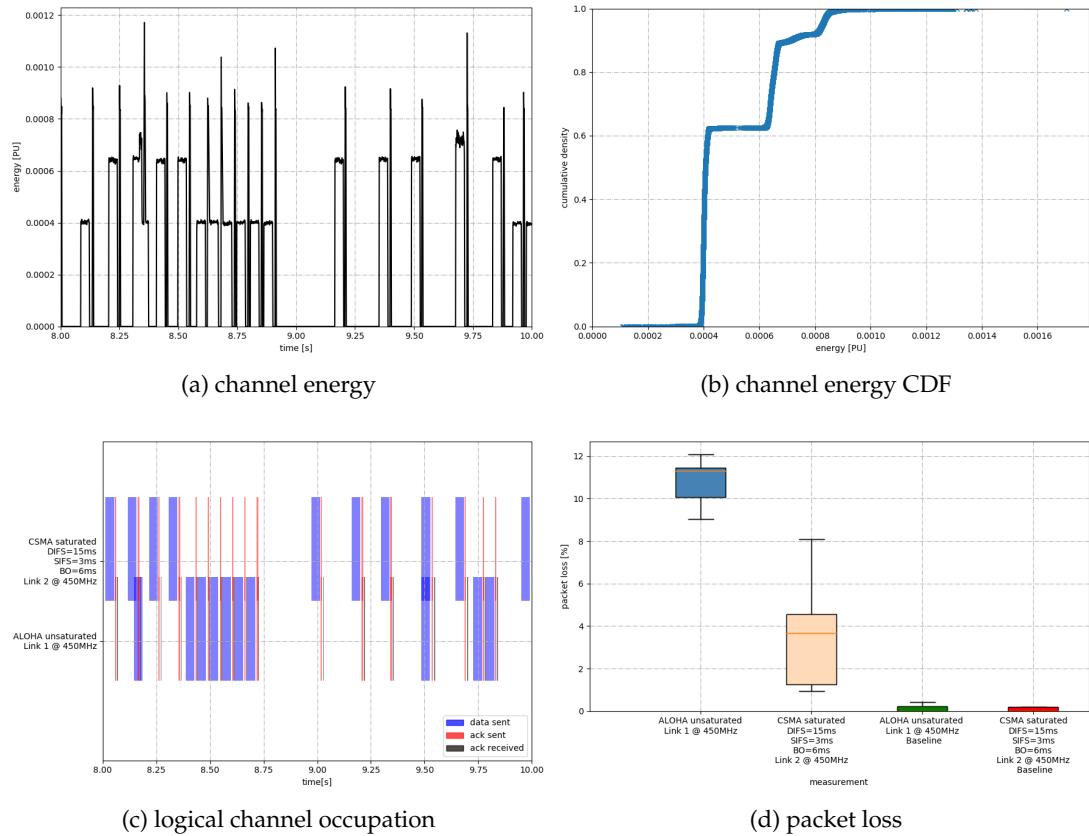
FIGURE 5.11: Measurement results for unsaturated ALOHA combined with saturated CSMA/CA (high parameter set)

5.3.2 Unsaturated ALOHA and CSMA/CA

The next experiment tries to give a feeling, whether relatively low load unsaturated⁴ ALOHA can coexist with CSMA/CA. The throughput of the CSMA/CA sender is reduced to $\frac{18}{45} = 40\%$ of the baseline value as depicted in Figure 5.11 (a), while the ALOHA throughput approximately remains the same.

Only if the CSMA/CA node transmits a frame before the ALOHA node a collision can occur, which happens during seconds 8-10 of the measurement as shown in 5.12(c) from a logical POV or in 5.12(a) from a physical POV. With that in mind, we can explain why the frame delay of CSMA/CA varies much more than the frame delay of ALOHA (Figure 5.11(b)). The number of ALOHA packets generated during data frame transmission time (or any other period of time) is Poisson-distributed and thus the number of collisions, whereas the likelihood of collision from the POV of an ALOHA frame is dependent on the backoff which in our case is uniformly distributed. The same explanation also applies for the differing variances in packet loss as depicted in 5.12(d), whereas the differing values are because ALOHA recklessly pushes its packets into channel, while CSMA does not interfere with ALOHA packets when it senses energy in the channel.

⁴we still refer to exponentially distributed time between each packet with $\frac{1}{\lambda} = 200ms$, which gives us roughly $G_{ALOHA,unsat} \approx \frac{38kbps}{130kbps} \approx 0.3$, where 38 kbps is the baseline throughput of unsaturated ALOHA and 130 kbps the baseline throughput of saturated ALOHA. We can use this approximation, because the offered channel load of ALOHA is independent from other transmitters and saturated ALOHA approximatively consumes the whole channel capacity.



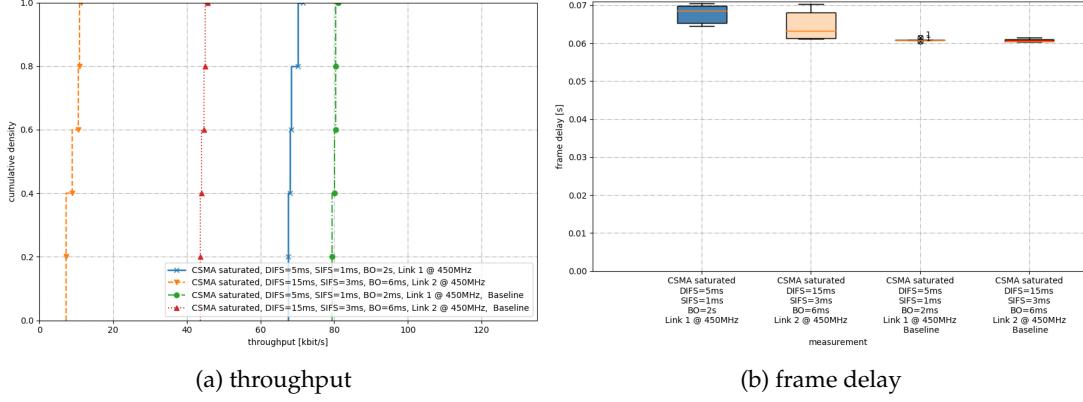


FIGURE 5.13: Measurement results for an inhomogeneous combination of CSMA protocols.

5.3.3 Inhomogeneous CSMA/CA

The goal of our next experiment is to examine how CSMA/CA with different parameter sets behave in the same channel. Link 1 uses the low parameter set, whereas link 2 uses the high parameter set. The throughput of link 2 as depicted in Figure 5.13(a) drops to one seventh of the baseline, whereas it only drops by 20% for link 1. The reason for this is that with reduced DIFS and BO the chance to grab the channel increases as is shown in Figures 5.14(a)(b)(c). The frame delay increases only marginally as shown in Figure 5.13(b), which is due to the packet loss depicted in 5.14(d). The mean packet loss is for link 2 is a little below the expected value of about 1.2%, where as for link 1 it is above that value, which is because the SINR of link 2 is higher than for link 1. The expected packet loss comprises of two components. The first component is the baseline packet loss for this measurement, which is around 0.2%. The second component is the chance that both senders choose the same time to transmit their packet when they sensed the channel idle, which amounts to $\frac{1}{CW_{\min}} \cdot \frac{BO_1}{BO_2} = \frac{1}{96} \approx 1.2\%$. An idea to reduce the chance of collisions due to this phenomenon is to configure the links to have backoff slots durations that are mutually prime, i.e. have no common divisor, which probably only works if the duration of a backoff slot is big compared to the time granularity of the whole system.

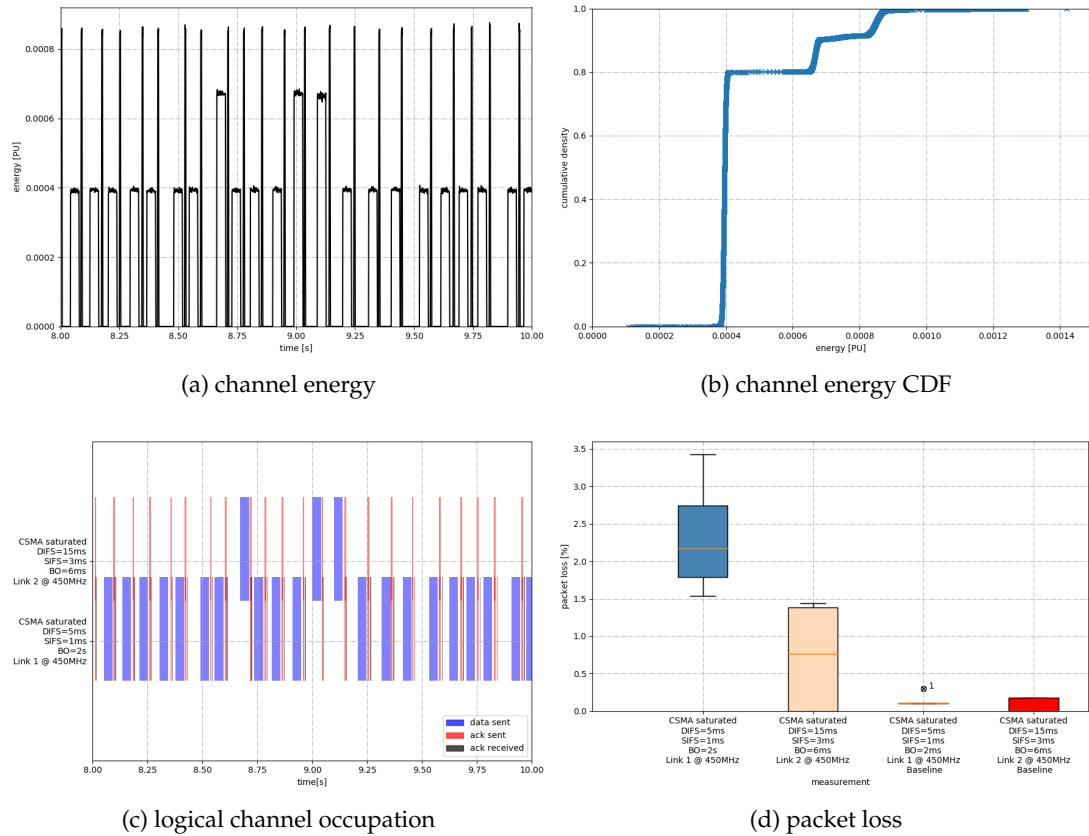


FIGURE 5.14: Measurement results for an inhomogeneous combination of CSMA protocols.

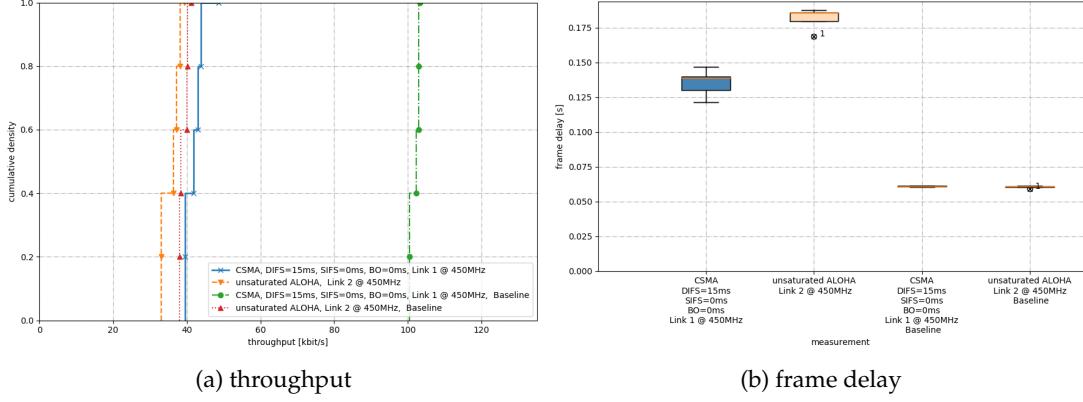


FIGURE 5.15: Measurement results for unsaturated ALOHA combined with 1-persistent CSMA

5.3.4 1-persistent CSMA and unsaturated ALOHA

It would not make much sense to have a second link in a channel if there already is another link that sends saturated ALOHA traffic through the channel as shown in Sections 5.2.1 and 5.3.1. For this reason link 1 uses 1-persistent CSMA and link 2 *unsaturated ALOHA*. In the given scenario it does not make much sense for the CSMA/CA transmitter to back off when the channel is sensed busy either, due to fact that the ALOHA node does not use the LBT mechanism, thus "giving it the chance to transmit" is waste of time as it transmits whenever it wants anyway. Thus, removing the backoff in CSMA/CA promises higher throughput. For this reason we now compare this experiment with the one in Section 5.3.2. The assumption of increased throughput is correct as the comparison between Figures 5.15(a) and 5.11(a) confirms that removing backoff more than doubles CSMA throughput. It would however, make sense to back off after the reception of an ACK to give the ALOHA node a chance to send a packet. The lack of this backoff explains the increased ALOHA mean packet loss ($\approx 34\%$ total, $\approx 300\%$ increase; Figure 5.16(d)), which is still comparatively low due to the high SINR of the ALOHA node. A representative excerpt of the logical channel occupation is given in Figure 5.16(c), where only a single ALOHA frame around second 8.8 does not collide with a CSMA packet. The energy plot in Figure 5.16(a) offers a more detailed physical view the channel, where the peak energy level of colliding data packets is not much higher than the energy level of a successful ALOHA transmission. In conjunction with the energy CDF in Figure 5.16(b), which is taking the whole measurement duration into account and has only a small bend around 0.0006 PU we conclude that if we decrease the TX power of the ALOHA node or use a less robust MCS, such as 64-QAM, ALOHA packet loss would drastically increase.

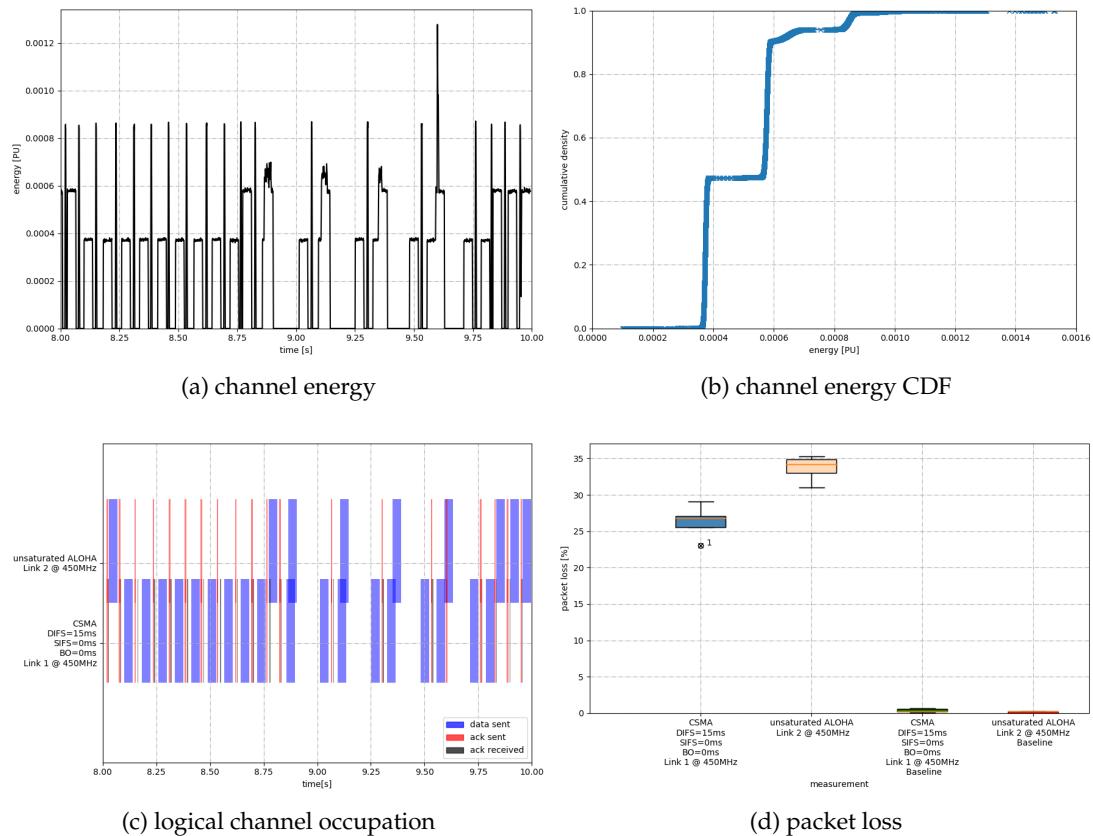


FIGURE 5.16: Measurement results for unsaturated ALOHA combined with 1-persistent CSMA

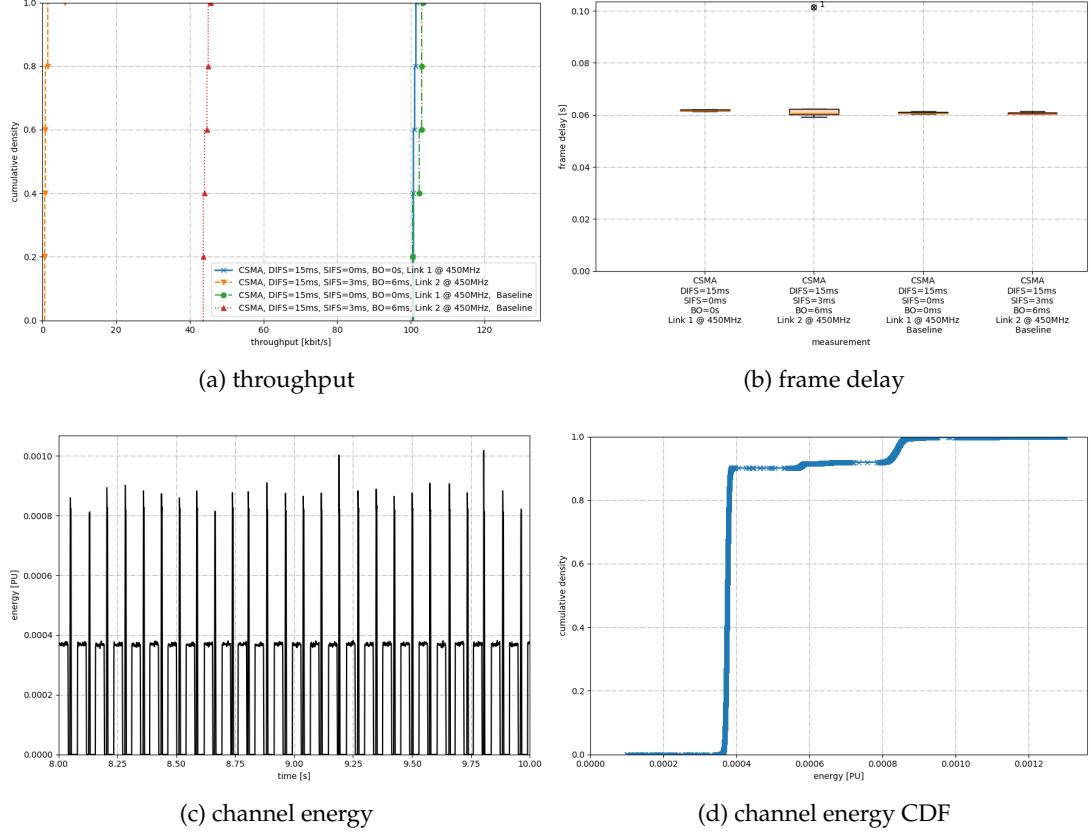


FIGURE 5.17: Measurement results for the CSMA/CA with large parameter set

5.3.5 1-persistent CSMA and CSMA/CA

In our last experiment we show that the greedy 1-persistent CSMA approach starves other CSMA/CA links. From the perspective of a CSMA/CA sender it does not matter, whether it shares a channel with a saturated 1-persistent CSMA sender or saturated ALOHA sender (Section 5.3.1). We can virtually identify no difference in the metrics (Figures 5.17(b)(c)(d) compared to the corresponding ones in Figures 5.10) between those combinations except for the reduced throughput 5.17(d) due to the addition of channel sensing with the duration of DIFS compared to ALOHA. Only if we decreased the offered load of 1-persistent CSMA we could see any and better coexistence⁵ with CSMA/CA at all.

⁵compared to ALOHA due to the LBT mechanism that stops the 1-persistent CSMA sender from interfering with ongoing transmissions

6

CONCLUSIONS AND FUTURE WORK

In the final chapter of the thesis, we will summarize the most important conclusions of the preceding Measurement Results chapter and give an outlook on what we think should be subject of future endeavors.

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
7         "bash -s" < remote_measurement_$link.sh
8 }
9
10 function prepare_measurement
11 {
12     reset
13     measurement_counter=0
14     ## let's make sure all the directories exist
15     printf "\nchecking if paths exists...\n"
16
17     #let's first make absolutely sure the raw data source path
18     #exists
19     if [ ! -d $raw_data_source_path ];
20         then
21             mkdir -p $raw_data_source_path
22             echo $raw_data_source_path" created."
23         else
24             rm -r $raw_data_source_path/*
25         fi
26
27     if [ -d $plot_directory_path ];
28         then
29             echo $plot_directory_path" already existed!"
30             cd $plot_directory_path
31             # create measurement directory
32             while [ -d $measurement_counter ]; do
33                 measurement_counter=$((($measurement_counter+1)))
34             done
35             export measurement_counter;
36         fi
37
38     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
46 created."
47
48 mkdir -p $data_source_path/$measurement_counter
49 echo $data_source_path/$measurement_counter" directory created."
50
51 mkdir -p $jobs_open_path
52 mkdir -p $jobs_done_path
53
54 ## let's check if measurement script is defined
55 # if ${measurement_scripts} undefined:
56 # go through directory and list all python files
57 if [ -z ${measurement_scripts+x} ];
58 then
59     echo "no measurement scripts set,
60           going through files inside of $locate_base_path."
61     echo "please add a the full path of one of the files to
62 \$scripts."
63     #locate -r "$locate_base_path" | grep "\.py$"
64     echo "terminated. ding dong"
65     exit -1
66 fi
67
68 printf "\n"
69
70 function measure
71 {
72     local prematurely_aborted=0
73
74     for ((x = 1 ; x <= $measurement_repetitions ; x += 1)); do
75
76         # get pid to later kill it
77         for i in "${measurement_scripts[@]}"
78         do
79             python $measurement_script_path/$i &
80             done
81
82         for ((y = $timer ; y > 0 ; y -= 1)); do
83             echo "measurement $x/$measurement_repetitions complete in $y
84 second(s)."
85             if [ ${check_if_prematurely_aborted} -eq 1 ];
86             then
87                 if $(ps -p ${measurement_scripts_pid[*]}) | grep
88 ${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 =
17     os.environ["plot_directory_path"]+"/"+os.environ["measurement_counter"]+("/")
17 plot_type = ["cdf", "boxplot"]

```

```

18 throughput_data_files =
19     os.environ["throughput_data_files"].split(",")
20 rtt_data_files = os.environ["rtt_data_files"].split(",")
21 co_data_files = os.environ["co_data_files"].split(",")
22 sniffer_data_files = os.environ["sniffer_data_files"].split(",")
23 retxs_data_files = os.environ["retxs_data_files"].split(",")
24 show_plot = int(os.environ["show_plot_after_measurement"])
25 rtt_mode = os.environ["rtt_mode"]
26 max_retxs = 6
27 eval_mode = "live"
28 timer = int(os.environ["timer"])
29 receiver_mode = os.environ["receiver_mode"]
30
31 #From Python
32 plot_pdf = False
33 boxplot_xticks = [ "measurement "+str(index) for index in
34     measurement ]
35 legend_labels = [ tick.replace("\n", " ", " ) for tick in
36     boxplot_xticks ]
37
38 custom_legend_coordinates = {
39     "rtt": [0.24,0.85,"upper left"],
40     "packet_loss": [1,0,"lower right"],
41     "retxs": [1,0,"lower right"],
42     "throughput": [1,0,"lower right"],
43     "diagnosis_sender": [1,0,"lower right"],
44     "diagnosis_receiver": [1,0,"lower right"],
45     "backoff": [1,0,"lower right"],
46     "channel_occupation": [1,0,"lower right"],
47     "sniffer": [1,0,"lower right"]
48 }
49
50 create_plots = {
51     "rtt": False,
52     "packet_loss": False,
53     "retxs": False,
54     "throughput": True,
55     "diagnostic": True,
56     "backoff": True,
57     "channel_occupation": True,
58     "sniffer": True
59 }
60
61 channel_occupation_mode = {
62     "occupation_mode": ["overview", "zoom"],
63     "zoom": [5,7],
64     "zoom_mode": "interval",
65     "zoom_interval": 2
66 }
67
68 sniffer_settings = {
69     "sniffer_mode": ["physical", "smoothed"],
70     "link": 1,
71 }
```

```

68     "zoom":                      [0.0, timer*repetitions],
69     "zoom_mode":                  "interval",
70     "zoom_interval":              2,
71     "smoothing_difference":      0.0001,
72     "smoothing_derivative":     0.01,
73     "smoothing_range":          [0.0010, 0.0013]
74 }
75
76 #Unimplemented, use later
77 annotations_below = []
78 annotations_other = []
79
80 eval_dict = {
81     "measurement":               measurement,
82     "repetitions":                repetitions,
83     "data_source_path":           data_source_path,
84     "xticks":                     boxplot_xticks,
85     "legend":                      legend_labels,
86     "annotations_below":          annotations_below,
87     "annotations_other":          annotations_other,
88     "throughput_data_files":      throughput_data_files,
89     "retxs_data_files":            retxs_data_files,
90     "rtt_data_files":              rtt_data_files,
91     "show_plot":                   show_plot,
92     "legend_coordinates":        custom_legend_coordinates,
93     "create_plots":                create_plots,
94     "links":                       links,
95     "rtt_mode":                    rtt_mode,
96     "channel_occupation_mode":    channel_occupation_mode,
97     "co_data_files":                co_data_files,
98     "sniffer_data_files":          sniffer_data_files,
99     "sniffer_settings":            sniffer_settings,
100    "timer":                        timer,
101    "plot_pdf":                     plot_pdf
102 }
103
104 for index,a_plot_type in enumerate(plot_type):
105     if plot_type[index] == "cdf":
106         grid = True
107     else:
108         grid = True
109
110     eval_dict["plot_type"] = [plot_type[index]]
111     eval_dict["plot_path"] = plot_path
112     eval_dict["grid"] = grid
113
114     if create_plots["backoff"] == True:
115         print("Creating backoff plot!")
116         backoff.backoff(**eval_dict).plot()
117     if create_plots["rtt"] == True:
118         print("Creating rtt plot!")
119         rtt.rtt(**eval_dict).plot()
120     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,
12                  **kwargs
13                  ):
14
15         print("Hello from myplot_belated.py!")
16
17         self.data           = np.asarray(data).transpose()
18         self.data_x         = data_x
19
20         print("Title is '" + title + "' .")
21         # FIXME: Very hackish, but who cares!
22         if (title == "Retransmissions per Frame"
23             or len(data) == 1
24             or "bar" in plottype
25             or "hist" in plottype
26             or "broken_barh" in plottype):
27             print("Keeping original data format instead of
transposing.")
28             self.data           = data
29
30         self.bins            = bins
31         self.plottype         = plottype
32         self.xlabel           = xlabel
33         self.ylabel           = ylabel
34         # Let's have reasonable figure dimensions
35         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                                         [])
41     self.annotations_other = kwargs.get("annotations_other",
42                                         [])
42     self.legend_coordinates = kwargs.get("legend_coordinates",
43                                         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_ylimits(ymin=0)
82         if self.xlims != False and not (aplot in ["boxplot",
83                                         "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 ==
90 "broken_barh":
91         if len(np.asarray(self.data).transpose()) ==
92 len(self.legend):
93             if self.legend_coordinates == False:
94                 box = self.ax.get_position()
95                 self.ax.set_position([
96                     box.x0,
97                     box.y0+box.height*0.3,
98                     box.width,
99                     box.height*0.7
100                ])
101                self.ax.legend(fancybox=True,
102                               loc='upper center',
103                               bbox_to_anchor=(0.5, -0.15))
104            else:
105                if self.legend_coordinates[2] != "best":
106                    self.ax.legend(fancybox=True,
107                                   loc=self.legend_coordinates[2],
108                                   bbox_to_anchor=(self.legend_coordinates[0],
109                                   self.legend_coordinates[1]))
110                else:
111                    self.ax.legend(fancybox=True, loc="best")
112            else:
113                print ("len(self.data) = "
114                  +
115 str(len(np.asarray(self.data).transpose())))
116                + " and len(self.legend) = "
117                + str(len(self.legend))
118                +" don't match!")
119 # Add annotations:
120 for annotation in self.annotations_other:
121     self.ax.annotate(annotation)
122 # Save and show plot
123 if(savepath):
124     print ("***savepath***")
125     print (savepath)
126     self.save(savepath, aplot)
127 if(show):
128     self.show()
129
130 #plt.close(self.fig)
131 self.fig.clear()
132 #print(self.data)

```

```

131         self.annotate()
132
133     def bar(self):
134
135         colors = ['steelblue', 'orange', 'green', 'red', 'purple',
136 'brown', 'pink']
137         color_repetitions = math.ceil(len(self.data)/len(colors))
138         colors = color_repetitions * colors
139
140         print(self.data)
141         for idx, val in enumerate(self.data):
142             data_points = len(self.data[idx])
143             width = 5 / data_points
144             index = np.arange(data_points)
145
146             self.ax.bar(left=idx * width + width / 2,
147                         height=val,
148                         color=colors[idx],
149                         alpha=0.5
150                     )
151
152         self.setLabels(ylabel=self.ylabel,
153                         xlabel="measurement",
154                         title=self.title
155                     )
156
157     def broken_barh(self):
158         plot_data = []
159         debug_data = []
160         for index, item in
161             enumerate(self.data["occupation_starting"]):
162
163             plot_data.append(list(zip(self.data["occupation_starting"][index],
164                         self.data["occupation_durations"][index])))
165
166             for index, item in enumerate(self.data["acks_received"]):
167
168                 debug_data.append(list(zip(self.data["acks_received"][index],
169                         self.data["acks_received_bar_width"][index])))
170
171             print("plot_data:")
172             #print(plot_data)
173             print("debug_data:")
174             #print(debug_data)
175             print("data_len:")
176             data_len = len(plot_data)
177             print(data_len)
178             debug_len = len(debug_data)
179             print("debug_len:")
180             print(debug_len)
181
182             print("data and ack lengths:")
183             for index, item in enumerate(plot_data):
184
185

```

```

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
196 range(int(data_len/2))])
197             self.xticks = [tick.replace(", ", ",\n") for tick in
198 self.xticks]
199             self.ax.set_yticklabels([self.xticks[index] for index in
200 range(int(data_len/2))])
201
202             self.setLabels(
203                 xlabel="time[s]",
204                 title=self.title
205             )
206
207             colors = ['blue','red','black']
208             transparency=0.7
209
210             patch_a = mpatches.Patch(color=colors[0],
211             alpha=transparency, label="data sent")
212             patch_b = mpatches.Patch(color=colors[1],
213             alpha=transparency, label='ack sent')
214             patch_c = mpatches.Patch(color=colors[2],
215             alpha=transparency, label="ack received")
216
217             if self.legend_coordinates[2] != "best":
218                 self.ax.legend( handles=[patch_a,patch_b,patch_c],
219                             fancybox=True,
220                             loc=self.legend_coordinates[2],
221                             bbox_to_anchor=(self.legend_coordinates[0],
222 self.legend_coordinates[1]))
223             else:
224                 self.ax.legend( handles=[patch_a,patch_b,patch_c],
225                             fancybox=True,
226                             loc="best")
227
228             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         ":" ,"-", "--"]
279         linewidths = [1.8,1.65,1.5,1.35,1.2,1.05,1,0.9,0.8,0.75]
280
281         if self.eval_mode == "belated":
282             cdf_data = np.asarray(self.data).transpose()
283         if self.eval_mode == "live":
284             cdf_data = np.asarray(self.data).transpose()
285             if self.title == "Retransmissions per Frame":
286                 cdf_data = [cdf_data]
287             if len(self.data) == 1:
288                 print("Hooray, my title is "+self.title+".")
289                 cdf_data = self.data
290
291             if len(cdf_data) > 1:
292                 for index,item in enumerate(cdf_data):
293                     print("index:"+str(index))
294                     print("__markers__")
295                     print(markers[index])
296                     x = np.sort(item)
297                     y = np.arange(1,len(x)+1) / len(x)
298                     x = np.insert(x,0,x[0])
299                     y = np.insert(y,0,0)
300                     self.plot = plt.step(x,
301                         y,
302                         marker=markers[index],
303                         linestyle=linestyles[index],
304                         linewidth=linewidths[index],
305                         markevery=range(1,len(x)),
306                         label=self.legend[index])
307             else:
308                 x = np.sort(cdf_data[0])
309                 y = np.arange(1,len(x)+1) / len(x)
310                 x = np.insert(x,0,x[0])
311                 y = np.insert(y,0,0)
312                 print(x)
313                 print(y)
314                 self.plot = plt.step(x,
315                         y,
316                         marker=markers[0],
317                         linestyle=linestyles[0],
318                         linewidth=linewidths[0],
319                         markevery=range(1,len(x)),
320                         label=self.legend[0])

```

```

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',
342 cm(float(index/len(self.patches))))
343
344 def boxplot(self):
345     print(self.data)
346
347     self.plot = plt.boxplot(self.data,
348                             #notch=True,
349                             patch_artist=True,
350                             flierprops=dict(marker='x'))
351
352     colors = ['steelblue', 'peachpuff', 'green', 'red',
353 'purple', 'brown', 'pink']
354     color_repetitions = math.ceil(len(self.data)/len(colors))
355     colors = color_repetitions * colors
356
357     for patch, color in zip(self.plot['boxes'], colors):
358         patch.set_facecolor(color)
359
360     self.setLabels( xlabel="measurement",
361                     ylabel=self.ylabel,
362                     title=self.title)
363
364     boxdict = self.ax.boxplot(self.data)
365     fliers = boxdict["fliers"]
366
367     for j in range(len(fliers)):
368         yfliers = boxdict['fliers'][j].get_ydata()
369         xfliers = boxdict['fliers'][j].get_xdata()
370         ufliers = set(yfliers)
371         for i, uf in enumerate(ufliers):
372             self.ax.text(xfliers[i] + 0.05, uf,
373 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
375 range(len(self.xticks))],self.xticks)
376     #self.ax.set_xticklabels(self.xticks);
377 else:
378     print ( "len(self.data) = "
379           + str(len(self.data))
380           + " and len(self.xticks) = "
381           + str(len(self.xticks))
382           +" don't match!")
383
384 def setLabels(self, xlabel="", ylabel="", title=""):
385     self.ax.set_xlabel(xlabel)
386     self.ax.set_ylabel(ylabel)
387     self.ax.set_title(title)
388
389 def save(self, savepath, plot_type):
390     #savename = self.title
391     savename = self.savename
392     savename = savename.lower()
393     savename = savename.replace(" ", "_")
394     self.fig.savefig(savepath+savename+".png")
395     if self.plot_pdf:
396         self.fig.savefig(savepath+savename+".pdf")
397
398 def show(self):
399     plt.show()
400
401 def annotate(self):
402     pass
403
404 def debug(self):
405     print("data: "+str(self.data))
406     print("bins: "+str(self.bins))
407     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 digital 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

POV point of view

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

2.1	Setup to explain the hidden and exposed node problem. Each node can only reach its direct neighbors.	3
2.2	Classification of MAC techniques	4
2.3	Normalized throughput over offered load according to formulae in [3], [6], [7], , with $a = \tau/T_p$, where τ is the maximum propagation delay and T_p the packet transmission time and under the assumptions made in Section 2.1.5.1.	7
2.4	The three basic flavors of CSMA	8
2.5	Virtual carrier sensing in CSMA/CA, as described in [3] and [4]	9
2.6	Simple do-it-yourself (DIY) radio receiver circuit diagrams	12
2.7	GNU Radio Companion GUI	13
4.1	Photo of the measurement setup.	18
4.2	GRC Receiver Flowgraphs	20
4.3	GRC Pure ALOHA Transmitter Flowgraph	21
4.4	GRC CSMA Transmitter Flowgraph	22
4.5	The three-phase measurement script system	26
4.6	Plots depicting the quality standards of our measurement	27
5.1	Measurement results for two ALOHA senders in the same channel	29
5.2	Measurement results for the CSMA/CA with high parameter set	30
5.3	Energy in the channel detected by the sniffer for CSMA/CA with high parameter set.	30
5.4	Measurement results for the CSMA/CA with high parameter set	31
5.5	Measurement results for the CSMA/CA with low parameter set	31
5.6	Measurement results for the CSMA/CA with low parameter set	32
5.7	Measurement results for the CSMA/CA with medium parameter set	33
5.8	Measurement results for 1-persistent CSMA	34
5.9	Measurement results for 1-persistent CSMA	35
5.10	Measurement results for saturated ALOHA and CSMA/CA (medium parameter set) simultaneously active in the same channel	36
5.11	Measurement results for unsaturated ALOHA combined with saturated CSMA/CA (high parameter set)	37
5.12	Measurement results for unsaturated ALOHA combined with saturated CSMA/CA (high parameter set)	38
5.13	Measurement results for an inhomogeneous combination of CSMA protocols.	39
5.14	Measurement results for an inhomogeneous combination of CSMA protocols.	40
5.15	Measurement results for unsaturated ALOHA combined with 1-persistent CSMA	41

5.16 Measurement results for unsaturated ALOHA combined with 1-persistent CSMA	42
5.17 Measurement results for the CSMA/CA with large parameter set	43

LIST OF TABLES

2.1 Layers in the OSI model [2]	3
4.1 Setup parameters	18

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.lteforum.org/uploads/3/5/6/8/3568127/lte-u_coexistence_mechansim_qual-comm_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

selbstständig 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