Coexistence Study on 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 on 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...
[1] [2] [3]

Contents

ABSTRACT

Abstract here.

1

INTRODUCTION

Introduction here.

2

BACKGROUND

In this chapter the theoretical foundations and software development kit (SDK) for the succeeding work are treated. Firstly, the 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 analyzing weaknesses with respect to the challenges coming up in wireless transmission. The chapter concludes with describing the advantages of software-defined radio (SDR) and how GNU Radio can be used to create SDR.

2.1 MAC PROTOCOLS

2.1.1 MAC Layer in the OSI Model

The OSI model is a layered architecture that divides a telecommunication system into several manageable layers. The original model features seven layers, where the focus in this thesis is on the MAC layer. Note that the data link layer has been split into two sublayers: the medium access and the logical link control sublayers.

2.1.2 ALOHA

ALOHA is arguably the most simple MAC protocol. The basic idea is whenever a user wants to send data he does so. The higher the channel load, i.e. sending requests per

Layer	Responsibilities
Physical Layer	dealing with mechanical, electrical and timing interfaces of data transmission
MAC Sublayer	controlling medium access and frame synchronization
LLC Sublayer	multiplexing to enable different network protocols coexist, flow control and error control
Network Layer	routing and congestion control
Transport Layer	transmission reliability, same-order-delivery, congestion avoidance
Session Layer	token management, dialog control, synchronization
Presentation Layer	abstracting syntax and semantics of transmission
Application Layer	user application protocols, such as http, ftp, smtp and many more

TABLE 2.1: Layers in the OSI model

time unit, the more likely collisions will occur, which render all transmitted information useless.

The question that comes to mind is, how likely is it that a collision will not occur. In other words, how efficient is an ALOHA channel? Making a statement requires a few preliminary assumptions:

1. We are taking a look at pure ALOHA as described above.
2. We simplify the calculation by assuming a fixed frame length.
3. The number of packets generated during a frame time is a poisson-distributed random variable X.
4. The channel load G comprises of two portions: "new" and retransmitted frames.

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

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

The probability of zero frames being generated during the transmission of the frame is $Pr(X = 0) = e^{-G}$ (assumption 3). If no collision occurs during the transmission of frame F no other frame was sent off 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 collision may corrupt data is two frame times (assumption 2).

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

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

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

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

As an aside, the throughput can be doubled with slotted ALOHA. In contrast to pure ALOHA, slotted ALOHA allows transmission only 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.

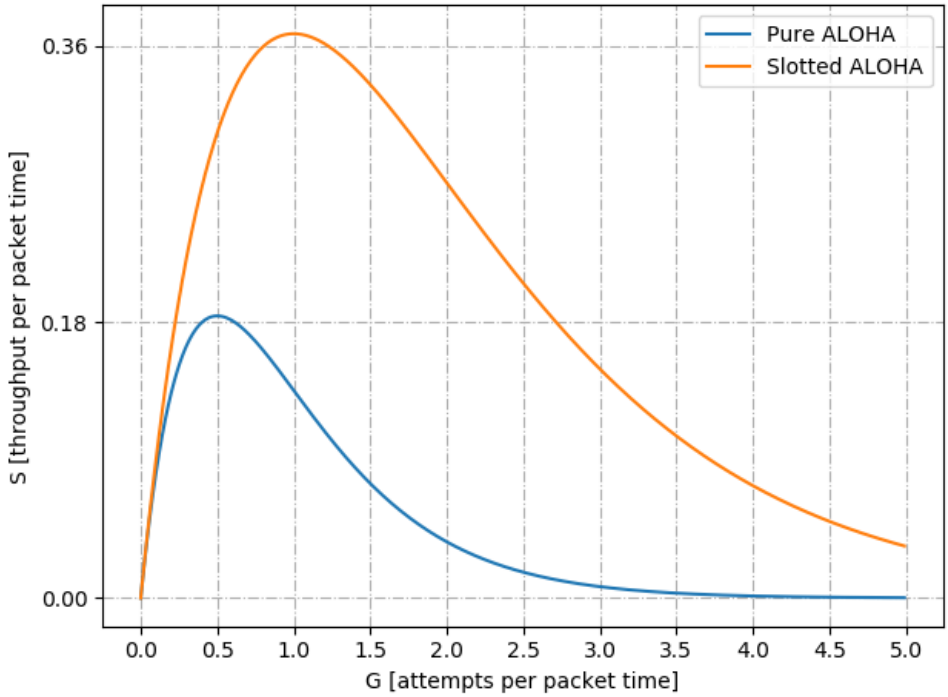


FIGURE 2.1: Pure ALOHA and slotted ALOHA's performance.

As shown above in figure 2.1 and the preceding paragraphs ALOHA's performance is discouraging and improvements over ALOHA were found.

2.1.3 CSMA

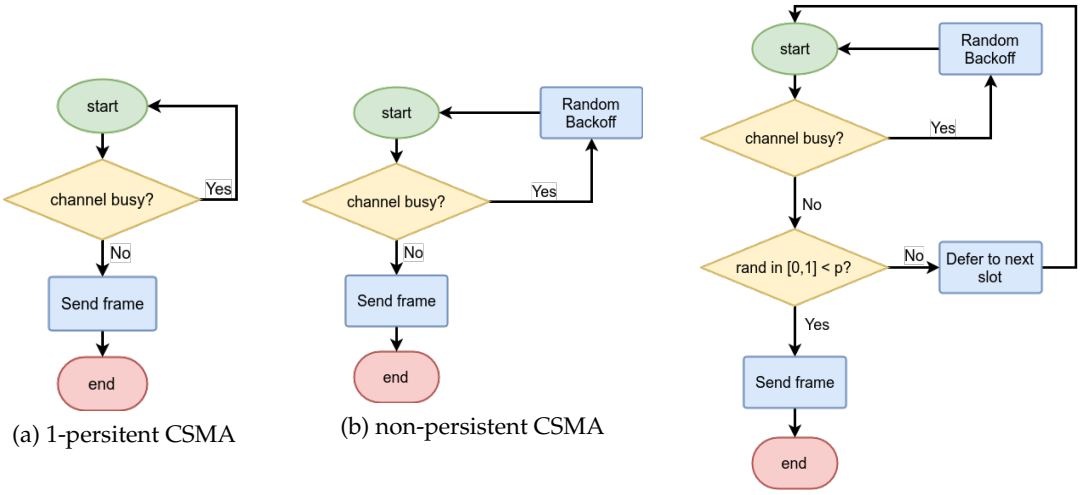
Main problem of ALOHA is the negligence of concurrent traffic in the channel. A solution to this problem is offered by the "listen before talk" (LBT) mechanism, which means in order to avoid collisions we sense the channel and refrain from sending should it be busy. This is the simple, yet effective basic idea of carrier sensing multiple access (CSMA) which comes in three flavors, as depicted in figure 2.2 which will be discussed next.

2.1.3.1 1-Persistent CSMA

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

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

- 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



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

FIGURE 2.2: The three flavors of CSMA

C are waiting for their turn. Once A finished transmission B and C will lunge onto the channel like a pack of wolves leading to collision.

- If propagation delay is not negligible the protocol suffers from an additional problem. In this 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 this protocol's performance.

2.1.3.2 *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 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.

2.1.3.3 *P-Persistent CSMA*

P-persitent CSMA is a protocol for slotted channels. Whenever a node A wishes to send off a packet the channel is sensed. If the channel is found idle it transmits its packet with a probability of p. With a probability $1 - p$ it defers the transmissions to the next slot. This process is repeated until one either the packet is sent off 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.

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 worlds: minimal delays as in 1-persistent CSMA, as well as high channel efficiency as in non-persistent CSMA.

2.1.4 CSMA with Collision Detection

A way to further improve CSMA-family 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 both bandwidth and time.

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 signal's energy or pulse width, 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.

$$P = \int_A I(\vec{x})\, dA \tag{2.6}$$

Where P is the power, I the intensity as function of the position $\vec{x}$ and dA the differential element of a closed surface around the source. Assuming that the integration takes place over the surface of a sphere with the radius r the term simplifies to:

$$P = \quad |I(r)| \cdot 4\pi r^2 \tag{2.7}$$

$$\Leftrightarrow \quad |I(r)| = \quad \frac{P}{4\pi r^2} \tag{2.8}$$

Another, and more common quantity in telecommunications is signal-to-noise ratio (SNR), which is defined as follows, where P is the signal power and N the noise power:

$$SNR[dB] = 10\, log\left(\frac{P}{N}\right) \tag{2.9}$$

Equations 2.8 and 2.9 imply if we increase the distance r by $\sqrt{2}$ the signal's intensity halves or lose 3dB in SNR, respectively. To make up for the loss in signal strength we would have to employ expensive signal processing in order to recover fainter signals. Alternatively, we could increase the transmit power, but this increases interference with other nodes, as well as electricity consumption.

2.1.5 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 2.3. Further challenges, such as energy limitations will also be delineated.

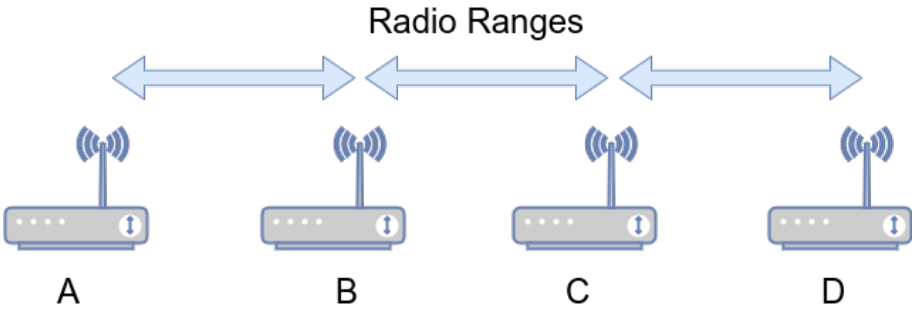


FIGURE 2.3: Setup to explain the hidden and exposed node problem. Each node can only reach its neighbors.

2.1.5.1 *The Hidden Node and the Exposed Node Problem*

Suppose that the node's radio range is limited to the neighboring nodes and A would like to transmit to B. If C just started transmitting A won't hear C and falsely assume that the channel is idle and start transmitting. This is the hidden node problem.

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 despite collisions would only take place between B and C, where it does not matter. This is the exposed node problem.

2.1.5.2 *Further Challenges*

Further challenges to MAC protocol design include the power conservation when faced with constrained power resources, as in wireless sensor networks (WSN) where devices rely on batteries for their supply with power. Attempts to mitigate waste of energy have been made in several specialized, duty-cycle based MACs such as Sensor MAC, Timeout MAC and Berkeley MAC as in more detail shown in section 2.1.7.

On the same page, due to 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's lifetime. It is due to this fact that security is paramount in biomedical or military fields of application.

2.1.6 *CSMA with Collision Avoidance*

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.

Beside physical carrier sensing previously simply referred to as carrier sensing another mechanism, namely virtual carrier sensing in combination with RTS/CTS exchange is employed to mitigate the trouble caused by hidden nodes.

In order to explain these mechanisms we refer to the setup of figure 2.3 with a slight modification in so far as that each node's radio range shall span across two neighboring nodes in both directions. That is to say, A can hear B and C, but not D and so on. Figure 2.4 visualizes the chain of events whose explanation follows.

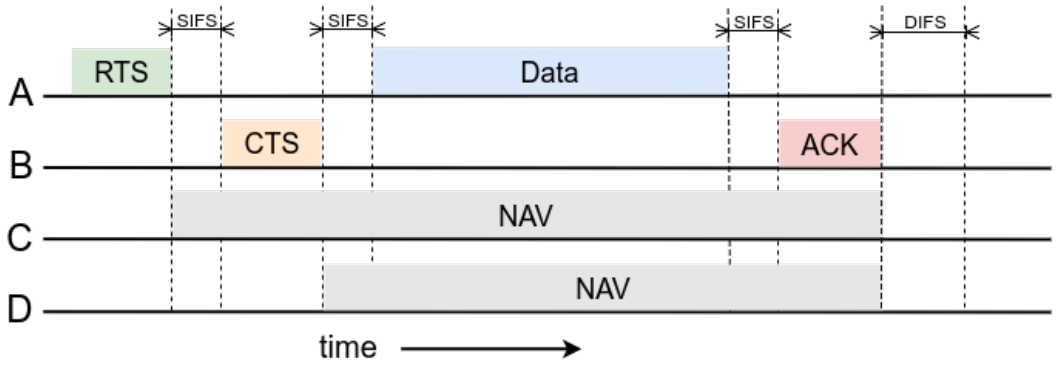


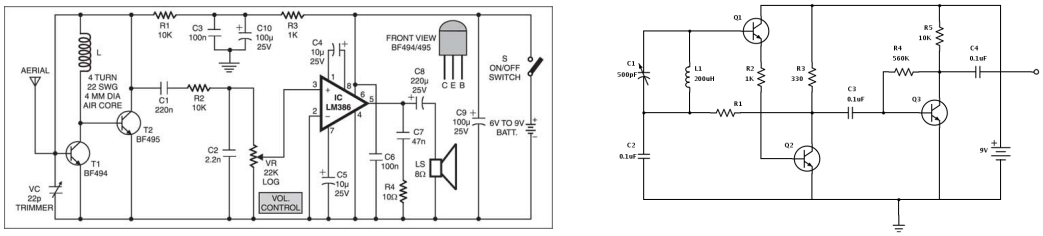
FIGURE 2.4: Virtual Carrier Sensing in CSMA/CA

A wants to send to B, hence issues a request to send (RTS). Every node receiving the RTS is shut down, except for B that in response to the RTS creates a clear to send (CTS) frame. Not only A receives this CTS frame, but also D, a hidden node from A'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 in this case A wants to transmit. Based on this length C and D setup the so-called network allocation vector (NAV), which are node-internal timers reminding C and D that the channel is still in use. Due to the fact, that no physical process is running to detect the channel status this mechanism get its name virtual carrier sensing. Shutting down nodes has the beneficial side-effect of reducing overhearing and therefore reduces energy consumption.

As further depicted in figure 2.4 there are named intervals of specified 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 party out of the two parties in dialog send off their frame without interference by another node. 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 previously consciously left out intervals, namely point coordination function interframe spacing (PIFS) and extended interframe spacing (EIFS). If 802.11 is operates in an alternative mode of operation, where a base station acts as the coordinator of traffic the standard prescribes an interval of length PIFS to allow the base station to send certain control (beacon and poll) frames. EIFS is used to report the reception of a bad or unknown frame and due to this action's low priority is the longest interval among the mentioned four.



(a) FM Receiver

(b) AM Receiver

FIGURE 2.5: Simple DIY Radio Receiver Circuit Diagrams

2.1.7 Duty-Cycle-Based MAC Protocols

2.1.7.1 Sensor MAC

2.1.7.2 Timeout MAC

2.1.7.3 Berekeley MAC

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 building at the cost of very limited flexibility in operation.

In contrast to hardware-defined radio there is SDR, where components such as filters, amplifiers, modulators, detectors and many more are implemented in software and run on general-purpose processors.

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 2.5, 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 this thesis SDR enables the variation of different MAC mechanisms' parameters.

2.2.2 What is GNU Radio?

The GNU Radio (GR) project is dedicated to the evolution of a free and open-source 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 flow charts simply by connecting signal processing blocks into a directed graph. Its audience is not merely limited to research and industry, but also encompasses academia, government and private users.

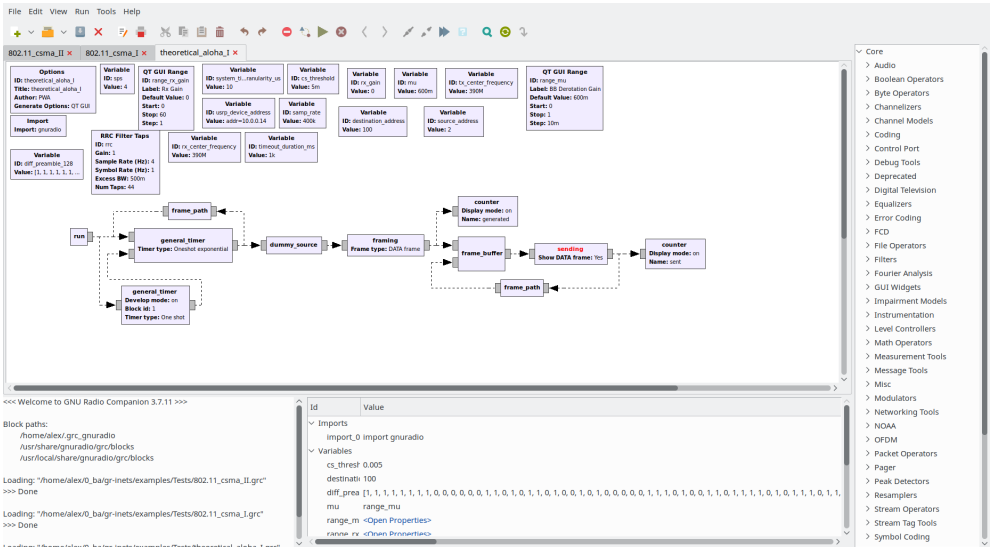


FIGURE 2.6: GNU Radio Companion GUI

2.2.3 *Core Concepts of GNU Radio*

2.2.3.1 *Flowcharts 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.

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.

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.

SWIG (simplified wrapper and interface generator) 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

Related Work here.

4

Measurement Methodology

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

4.1 GNU Radio Flowgraphs

4.1.1 Common Variables

Our GNU Radio pure ALOHA and non-persistent/1-persistent CSMA implementations are placed on top of a common PHY layer warranting comparability. The specific PHY layer implementation is beyond this work's scope, but a few parameters common to all flowgraphs shall be discussed nonetheless. Hereinafter, these variables and their values will not be mentioned unless they are important concerning the interpretation of results.

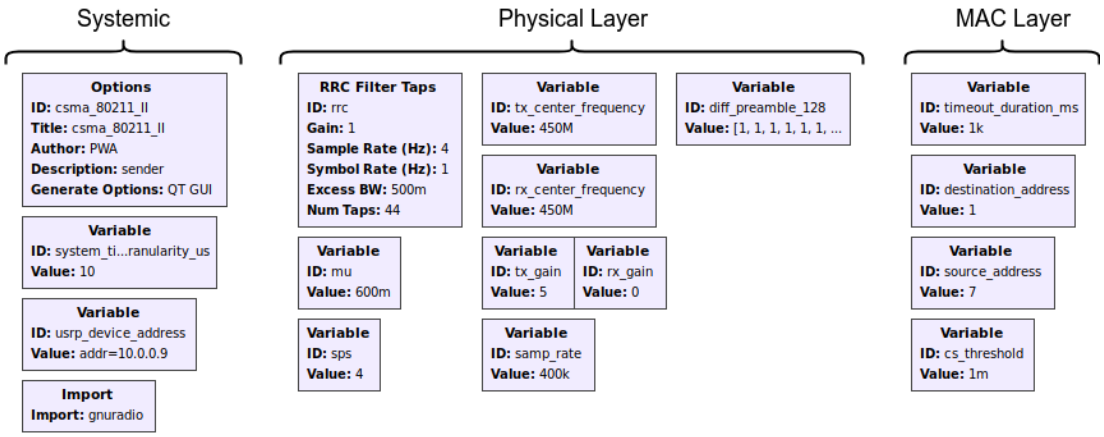
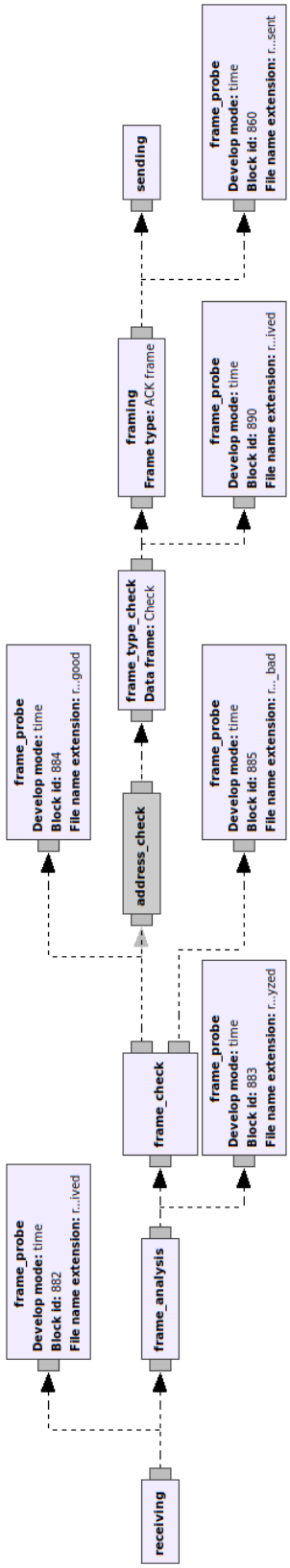
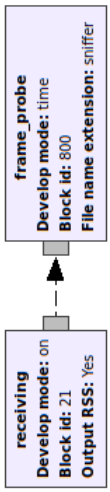


Figure 4.1: Variables Common to All Flowgraphs



(a) Receiver



(b) Sniffer

FIGURE 4.2: GRC Receiver Flowgraphs

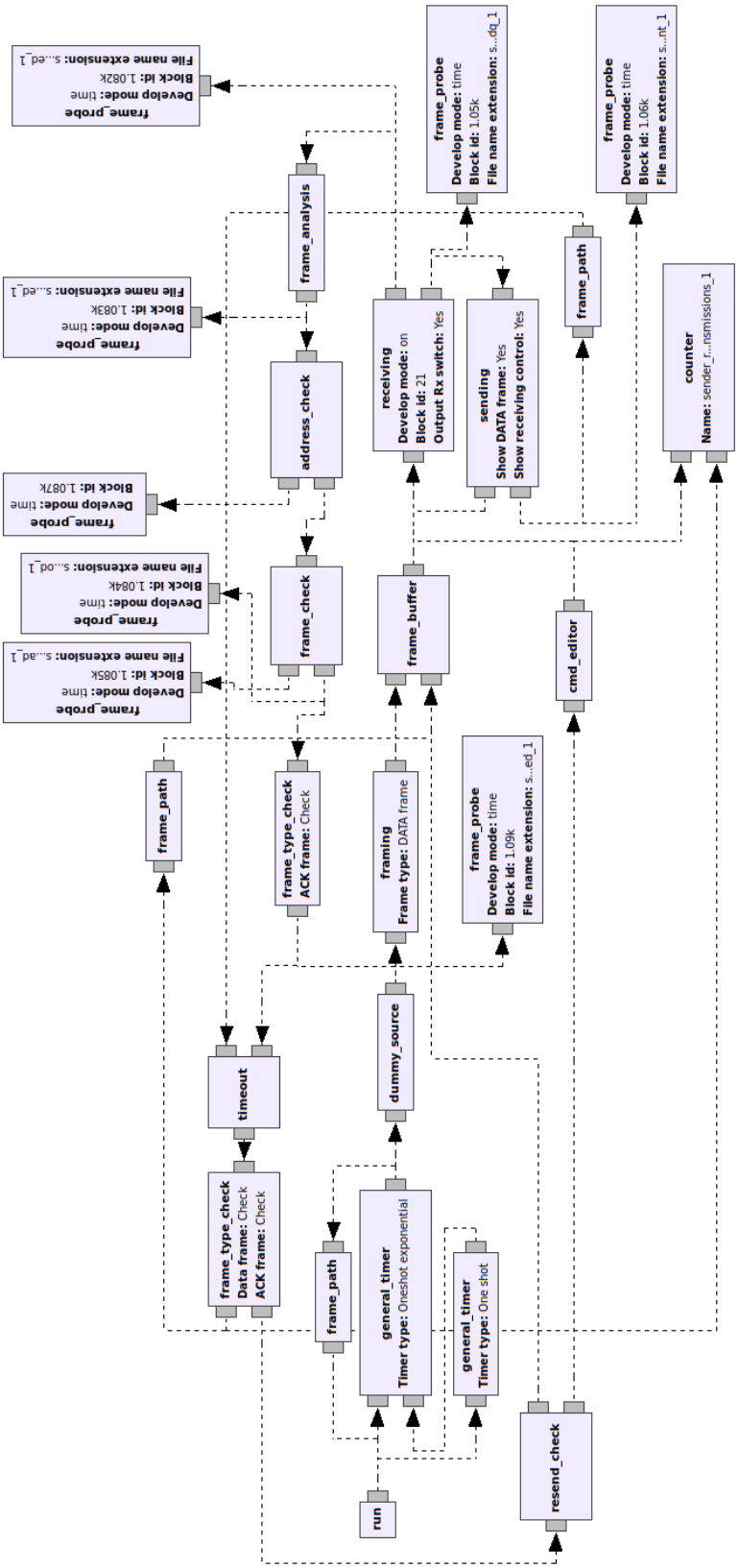


FIGURE 4.3: GRC Pure ALOHA Transmitter Flowgraph

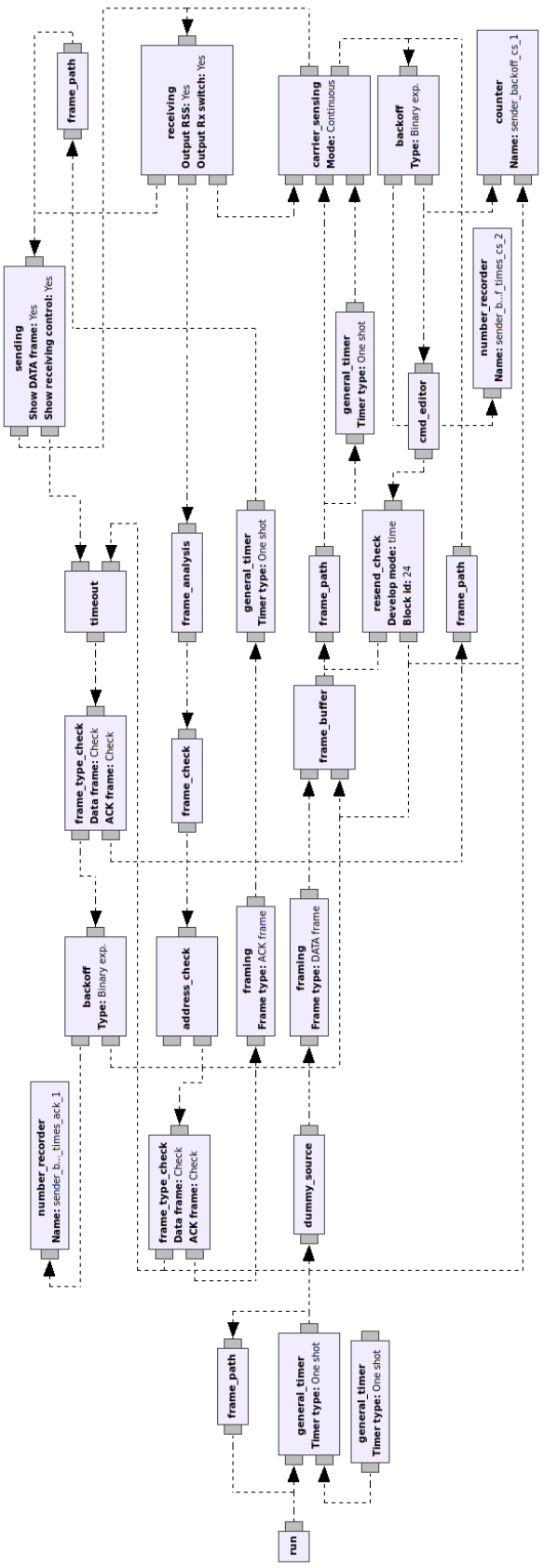


FIGURE 4.4: GRC CSMA Transmitter Flowgraph

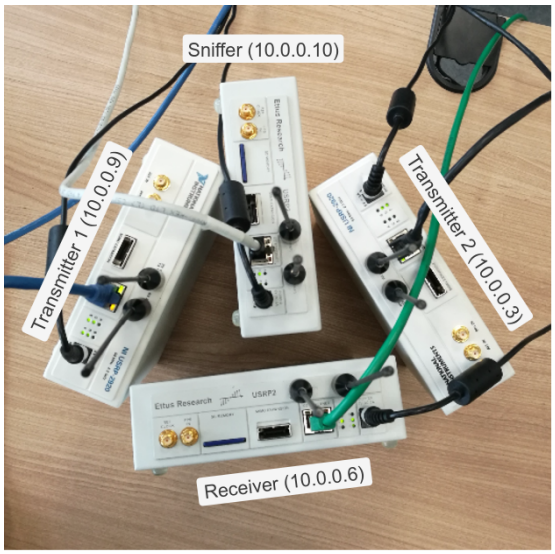


FIGURE 4.5: Photo of the Measurement Setup

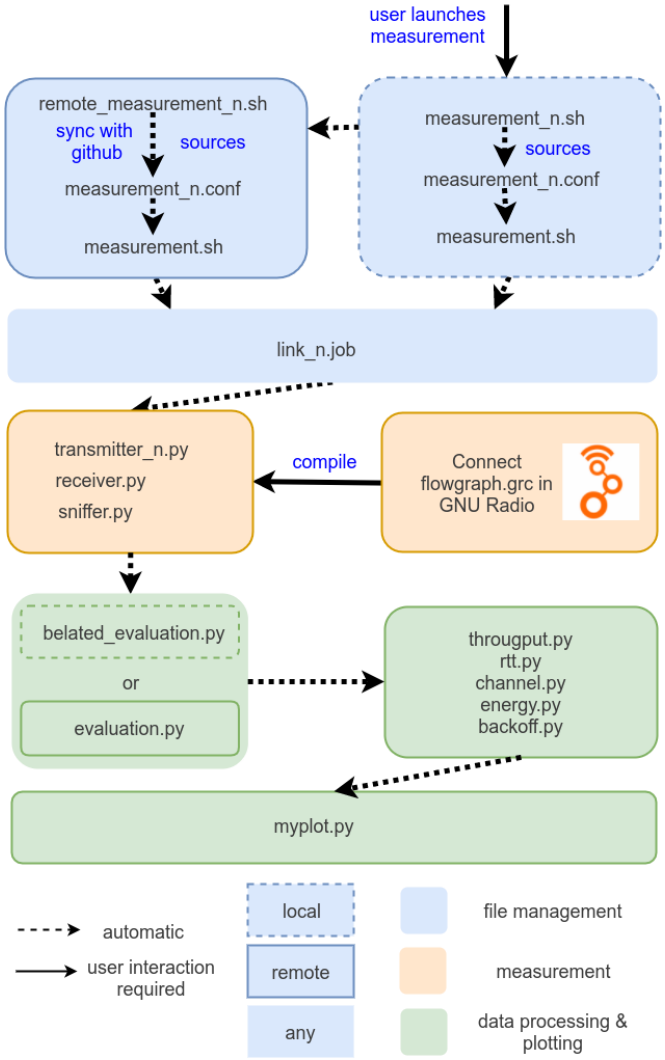


FIGURE 4.6: The Three-Phase Measurement Script System

5

MEASUREMENT RESULTS

For this chapter different combinations of two links employing pure ALOHA, non-persistent CSMA and 1-persitent-CSMA-like MAC protocols were run on the same channel. In-depth analysis of different metrics and a consecutive protocol assessment are provided.

5.1 SAME PROTOCOL COMBINATIONS

5.1.1 Pure ALOHA

5.1.2 Non-Persistent CSMA With Large Parameter Set

5.1.3 Non-Persistent CSMA With Small Parameter Set

5.1.4 Non-Persistent CSMA With Medium Parameter Set

5.1.5 DIFS-Only

5.2 DIFFERENT PROTOCOL COMBINATIONS

5.2.1 ALOHA and Non-Persistent CSMA

5.2.2 Unsaturated ALOHA and Non-Persistent CSMA

5.2.3 Inhomogeneous Non-Persistent CSMA

5.2.4 DIFS-Only and ALOHA

5.2.5 DIFS-Only and Non-Persistent CSMA

6

Conclusions and Future Work

Conclusions and Future Work here.

A

Bash and Python Scripts

This appendix aims at giving in-depth insight in the scripts used in the three phases of the measuring, data processing and plotting process. The basic principle, however, is depicted in section 4.4. Minor edits were made for format and aesthetic reasons.

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

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

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

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

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

LISTING A.1: measure.sh

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

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

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

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

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

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

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

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

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

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

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

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

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

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

LISTING A.2: evaluation.py

B

ABBREVIATIONS

AM amplitude modulation

CSMA carrier sense multiple access

CSMA/CA CSMA with collision avoidance

CSMA/CD CSMA with collision detection

CTS clear to send

DCF distributed coordination function

DIFS DCF interframe spacing

DIY do it yourself

EIFS extended interframe spacing

FM frequency modulation

GNU GNU is not Unix

GR GNU Radio

GRC GNU Radio Companion

GUI graphical user interface

LAN local area network

LBT listen before talk

MAC medium access control

NAV network allocation vector

PCF point coordination function

PDU packet data unit

PHY physical (layer)

PIFS PCF interframe spacing

PMT polymorphic type

QPSK quadrature phase-shift keying

RF radio frequency

RTS request to send

SDK software development kit

SDR software defined radio

SDU service data unit

SIFS short interframe spacing

SNR signal-to-noise ratio

SWIG simplified wrapper and interface generator

USRP universal software radio peripheral

WLAN wireless LAN

WSN wireless sensor networks

BIBLIOGRAPHY

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

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

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

Eidesstattliche Versicherung

Name, Vorname

Matrikelnummer (freiwillige Angabe)

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

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

Ort, Datum

Unterschrift

*Nichtzutreffendes bitte streichen

Belehrung:

§ 156 StGB: Falsche Versicherung an Eides Statt

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

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

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

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

Die vorstehende Belehrung habe ich zur Kenntnis genommen:

Ort, Datum

Unterschrift