Summary 4

Alexander Pastor

July 17, 2017

Contents

1 Organizational

1.1 Thesis Outline Proposal

1 Introduction

- 1.1 Abstract
- 1.2 Motivation and Context
  - 1.2.1 The LTE-U/LAA Debate
  - 1.2.2 Definition of Scope

2 Fundamentals and Preparation

- 2.1 MAC Protocols
  - 2.1.1 MAC in the OSI Model
  - 2.1.2 CSMA/CA
  - 2.1.3 CSMA/CA with CTS/RTS
  - 2.1.4 (Slotted) ALOHA
  - 2.1.5 ...
- 2.2 GNU Radio
  - 2.2.1 What is GR and GRC
  - 2.2.2 Motivation to use GR
  - 2.2.3 Limitations of GR

3 Measurement Results and Interpretation

- 3.1 Scenario Overview / Justification
- 3.2 Scenario A
- 3.3 Scenario B
- 3.4 Scenario C
- 3.5 ...
- 3.6 Interpretation

4 Conclusions

5 Appendices

- 5.1 How-to: Conduct Measurements Automatically
- 5.2 Alphabetical Glossary / Acronym Expansions
- 5.3 Alphabetical Index
- 5.4 Bibliography / References
- 5.5 Statement of Authorship

1.2 Device Parameters

Remote PC IP-address: 134.130.223.151
USRPs: 10.0.0.8, 10.0.0.10

1.3 Measurement Plan

For CW 25 the following measurement parameters were commissioned:

- 2 sets of 5 measurements each with a duration of 5 minutes.
- SIFS: 1ms / 3ms
- DIFS: 5ms / 15ms
- Backoff slot: 2s / 6ms
- The results should be plotted with either Matlab or Matplotlib
- **Update:** Matplotlib was chosen, because it allows easy and performant integration in the automatic measurement script (python script).

Thesis scenarios should be subject of the next meeting.

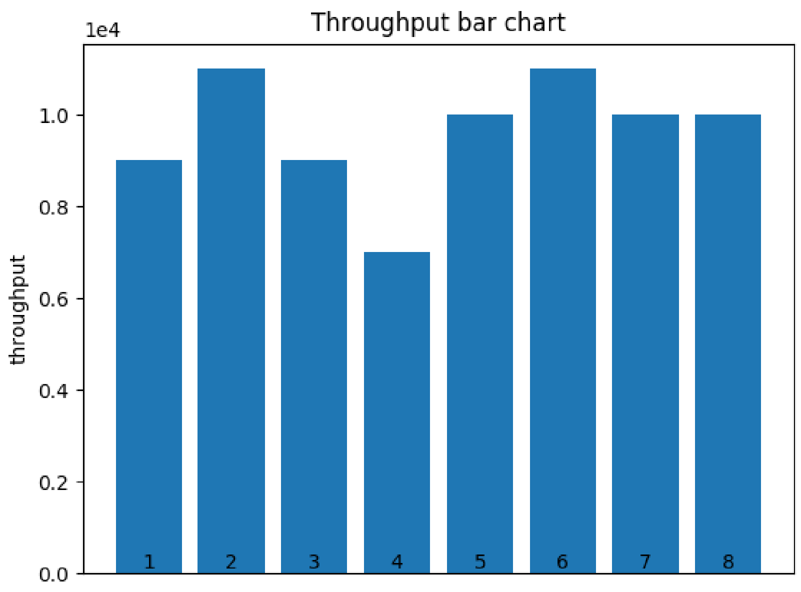
Note: The DIFS time in 802.11 can be calculated as twice the backoff time plus one SIFS time.

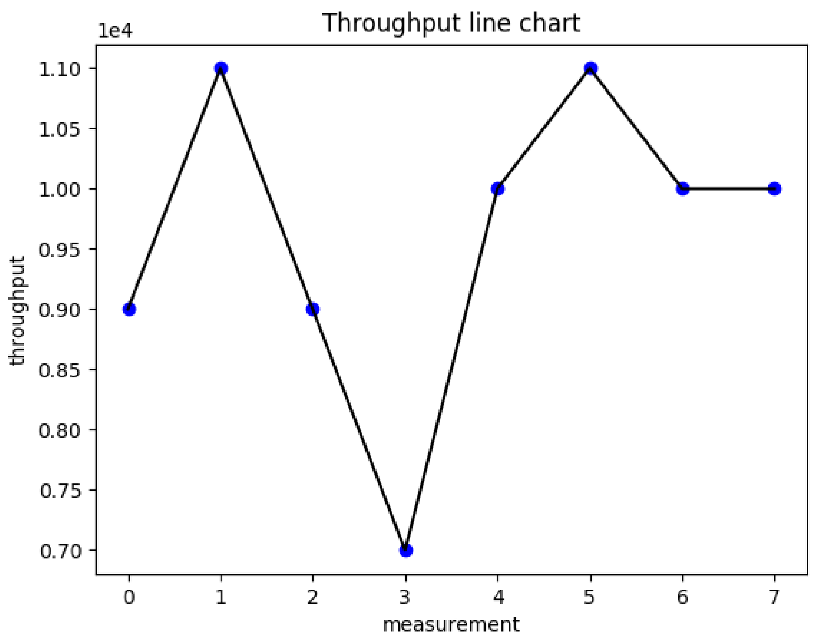
2 USRP Hardware Problems

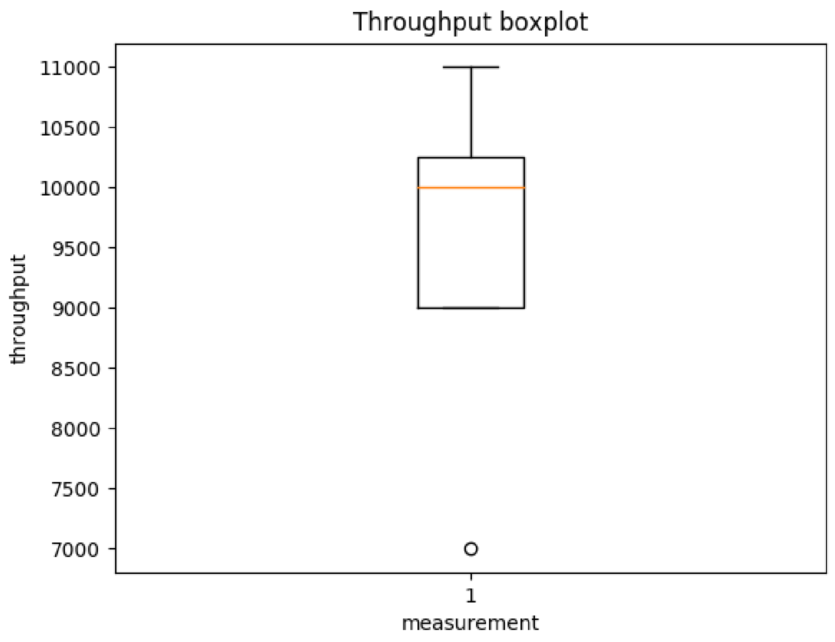
2.1 How It Is Supposed to Work

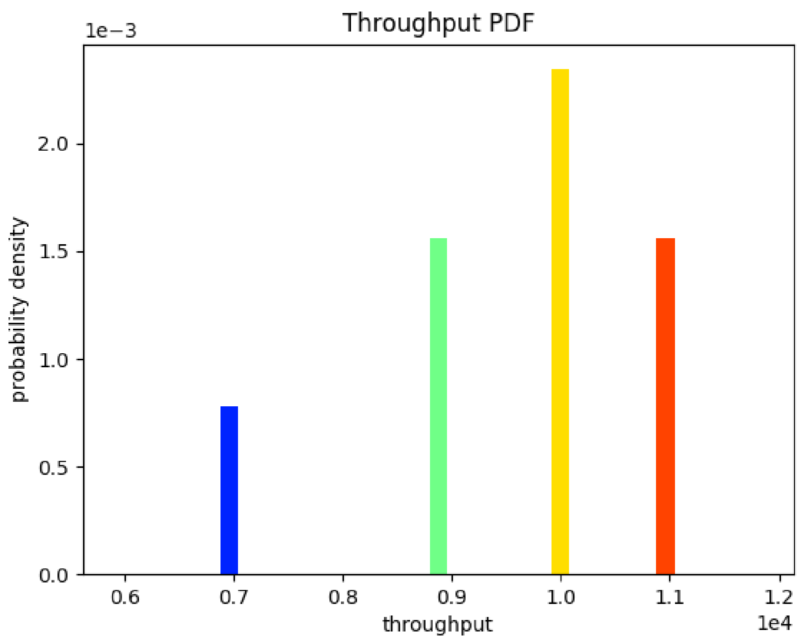
The following plots show the output of a successful measurement with two USRPs using CSMA/CA, although measurement 4 is a little bit messed up as well. Note that wrong x-axis on the line chart has been fixed by the time writing this.

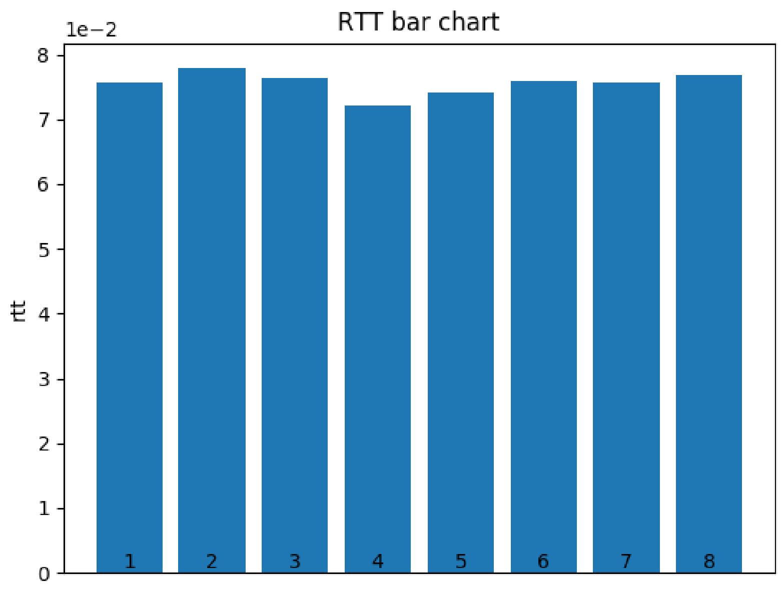
Note: Always take a look at the y-axis scaling. While such data representation might confuse the skimming reader it allows for higher resolution. Also note that not every plot adds more information, this also has the purpose of showcasing different plot types. And as a FYI: the linear interpolation in the line chart can be replaced with any other form of interpolation making it potentially more informative than the bar chart.

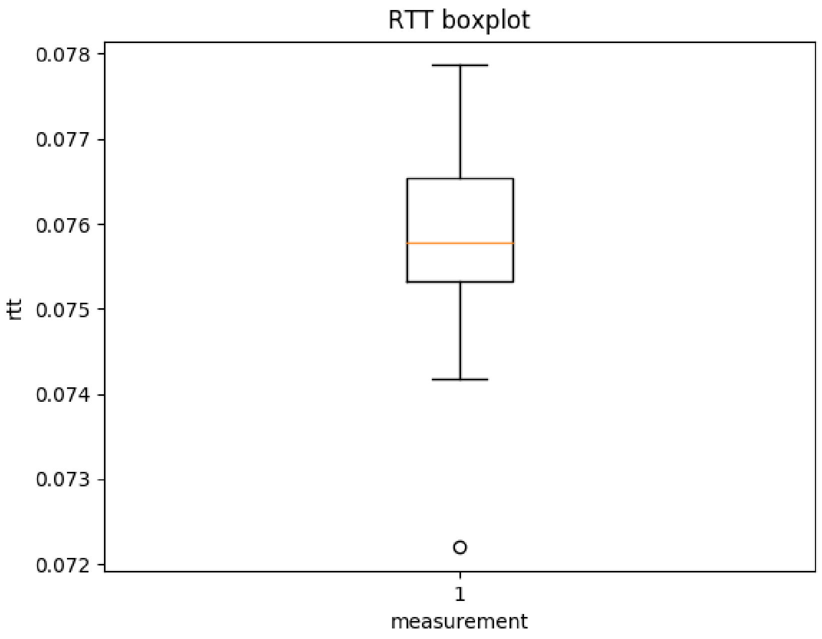


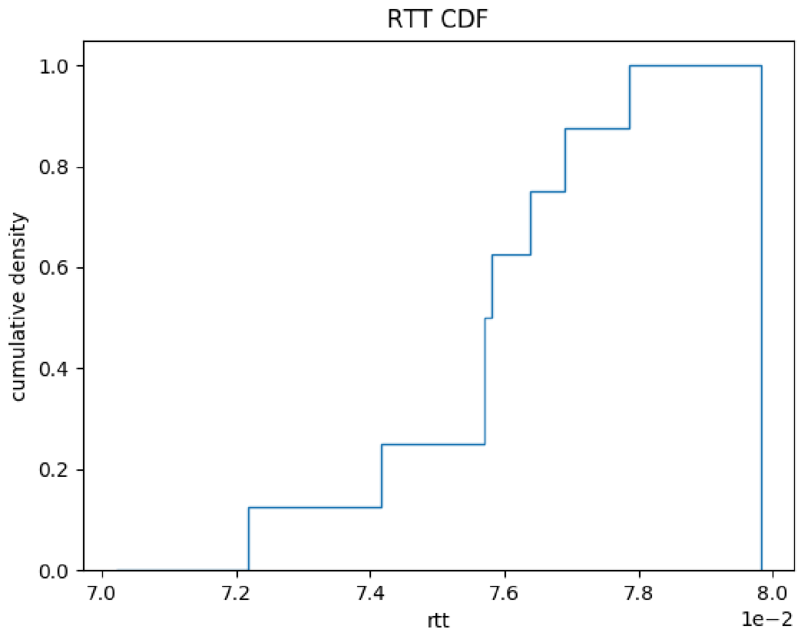

Throughput line chart
1e4
1.10
1.05
1.00
0.95
0.90
0.85
0.80
0.75
0.70
throughput
0
1
2
3
4
5
6
7
measurement


Throughput boxplot
11000
10500
10000
9500
9000
8500
8000
7500
7000
throughput
1
measurement


Throughput PDF
1e−3
probability density
throughput
1e4
0.0
0.5
1.0
1.5
2.0
0.6
0.7
0.8
0.9
1.0
1.1
1.2


RTT bar chart
1e−2
rtt
0
1
2
3
4
5
6
7
8
1
2
3
4
5
6
7
8


RTT boxplot
rtt
0.078
0.077
0.076
0.075
0.074
0.073
0.072
1
measurement

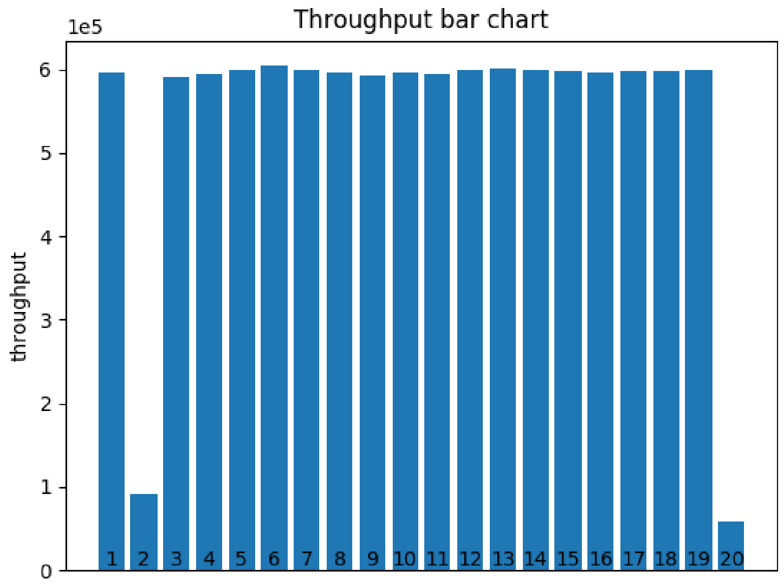

RTT CDF
cumulative density
1.0
0.8
0.6
0.4
0.2
0.0
7.0
7.2
7.4
7.6
7.8
8.0
1e−2
rtt

2.2 Failure Pattern 1

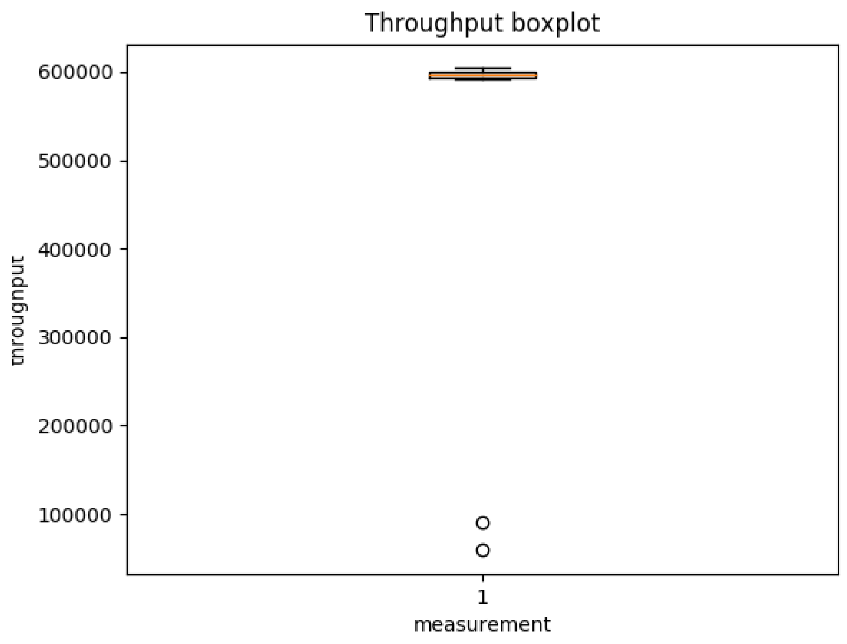
The following six figures visualize an issue I'm experiencing with the USRPs whose origin I have difficulties to track down. In seemingly random patterns the devices fail for the rest of the transmission. To make matters worse, it is not obvious whether the problem is enitrely hardware-related or has some software-related component. Let me first explain what happens in the following plots, though.

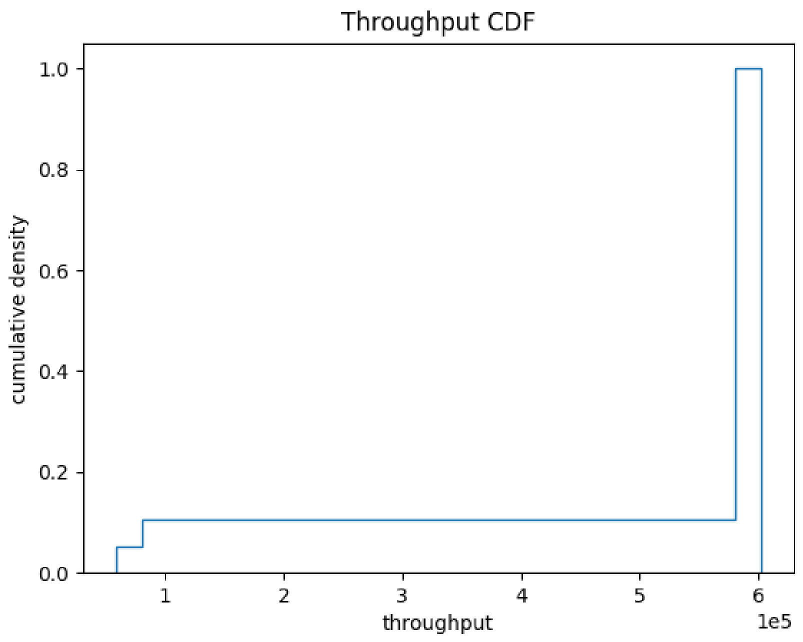
In the throughput bar chart it is obvious the device failed during the measurements 2 and 20. The logs of the measurement and this suggest that this could very well be a software problem. After 17 seconds the sender doesn't send any frames anymore (frame_probe 850 is not reached anymore). Once the sender fails it doesn't recover until the measurement script and the next measurement is launched, which is a reason to suspect a software failure on either my side or Gnu Radio's.

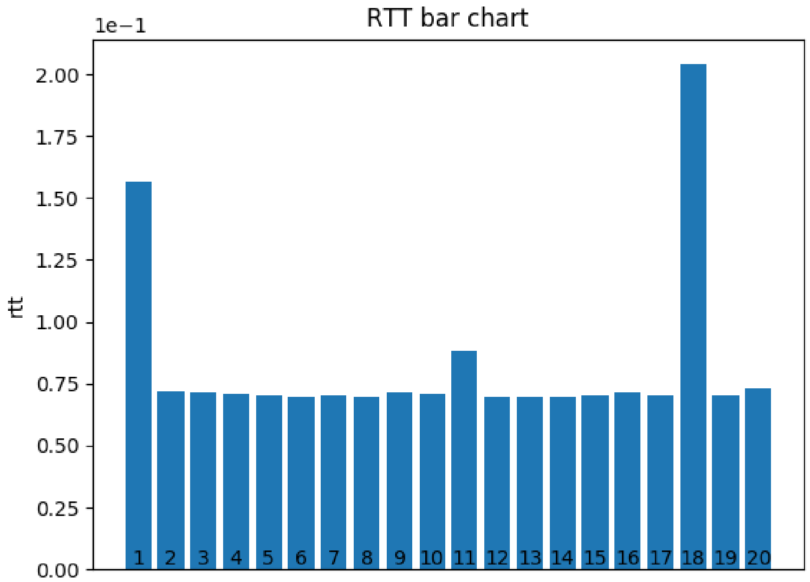
Note that the obviously wrong RTT in measurement 18 is due to the fact that retransmissions are taken into account when calculating the RTT. A fix for that is yet to be found. Alternatively, or as an alternative mode taking retransmissions into account with retransmission counters to calculate the frame delay instead of the RTT is being considered. However, at the moment there exist practical implementation barriers that are to be removed.

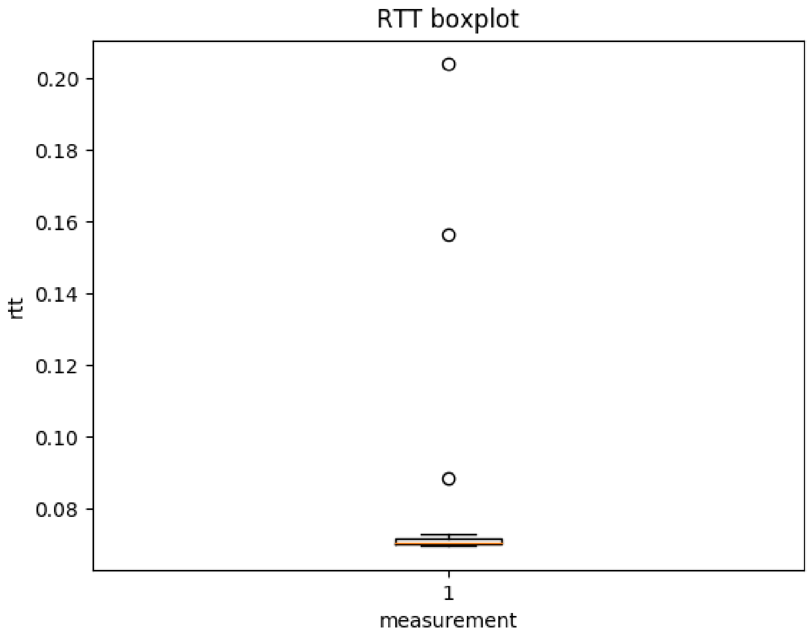


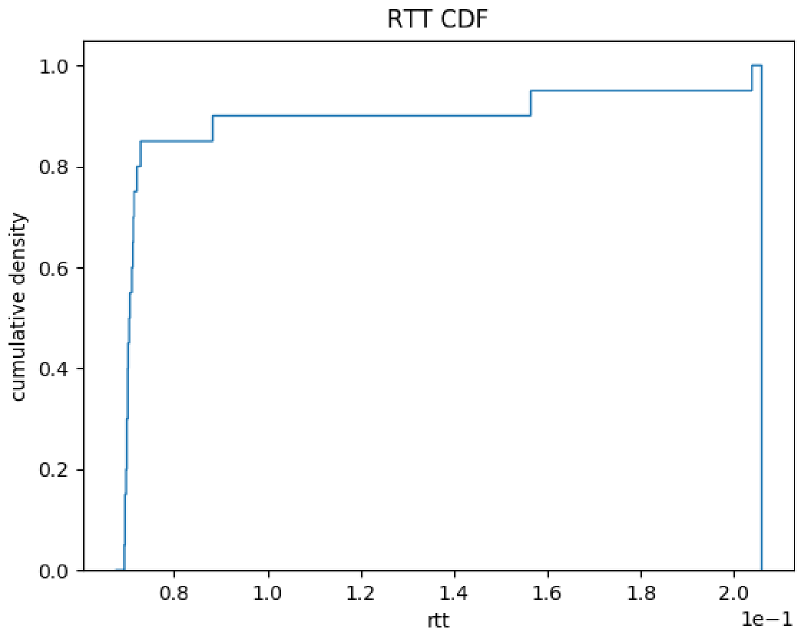
```
### This is from the measurements logs
Measurement 2/20 complete in 100 second(s).
# ... cut ...
Measurement 2/20 complete in 83 second(s).
++++ frame_probe ID: 860 receives a frame at time 71.8424s ++++
++++ frame_probe ID: 1010 receives a frame at time 71.846s ++++
++++ frame_probe ID: 890 receives a frame at time 71.8708s ++++
the 100th message counter  has been visited 89 times
++++ frame_probe ID: 850 receives a frame at time 71.9534s ++++
INFO: Detected an invalid packet at item 8544
INFO: Parser returned #f
++++ frame_probe ID: 860 receives a frame at time 71.9941s ++++
++++ frame_probe ID: 1010 receives a frame at time 71.9994s ++++
++++ frame_probe ID: 890 receives a frame at time 72.0241s ++++
the 100th message counter  has been visited 90 times
++++ frame_probe ID: 850 receives a frame at time 72.0925s ++++
INFO: Detected an invalid packet at item 8640
INFO: Parser returned #f
++++ frame_probe ID: 860 receives a frame at time 72.1332s ++++
++++ frame_probe ID: 1010 receives a frame at time 72.1437s ++++
++++ frame_probe ID: 890 receives a frame at time 72.1685s ++++
the 100th message counter  has been visited 91 times
Measurement 2/20 complete in 82 second(s).
Measurement 2/20 complete in 81 second(s).
Measurement 2/20 complete in 80 second(s).
Measurement 2/20 complete in 79 second(s).
Measurement 2/20 complete in 78 second(s).
Measurement 2/20 complete in 77 second(s).
Measurement 2/20 complete in 76 second(s).
Measurement 2/20 complete in 75 second(s).
Measurement 2/20 complete in 74 second(s).
Measurement 2/20 complete in 73 second(s).
Measurement 2/20 complete in 72 second(s).
# ... cut ...
Measurement 2/20 complete in 2 second(s).
Measurement 2/20 complete in 1 second(s).
```


Throughput boxplot
600000
500000
400000
300000
200000
100000
throughput
1
measurement


Throughput CDF
1.0
0.8
0.6
0.4
0.2
0.0
cumulative density
1
2
3
4
5
6
1e5
throughput


RTT bar chart
1e−1
rtt
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20


RTT boxplot
rtt
0.20
0.18
0.16
0.14
0.12
0.10
0.08
1
measurement

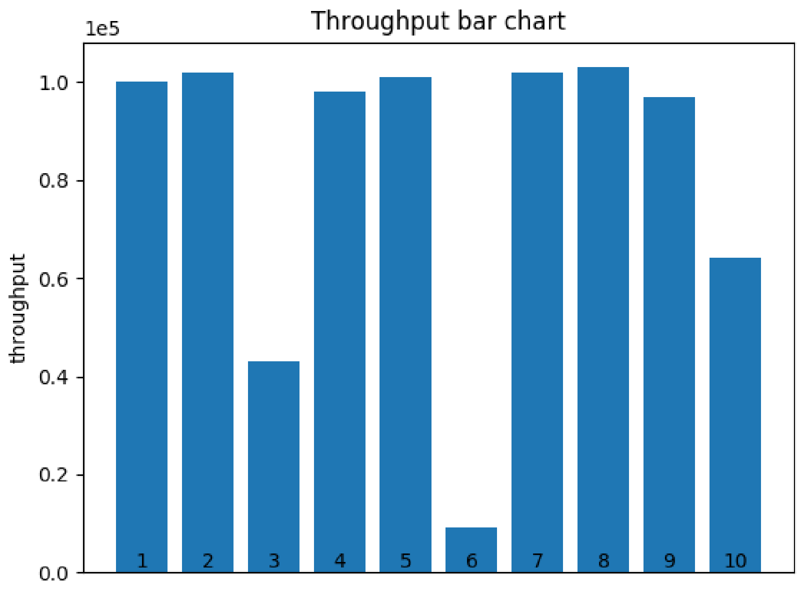

RTT CDF
cumulative density
rtt
1e−1
0.0
0.2
0.4
0.6
0.8
1.0
0.8
1.0
1.2
1.4
1.6
1.8
2.0

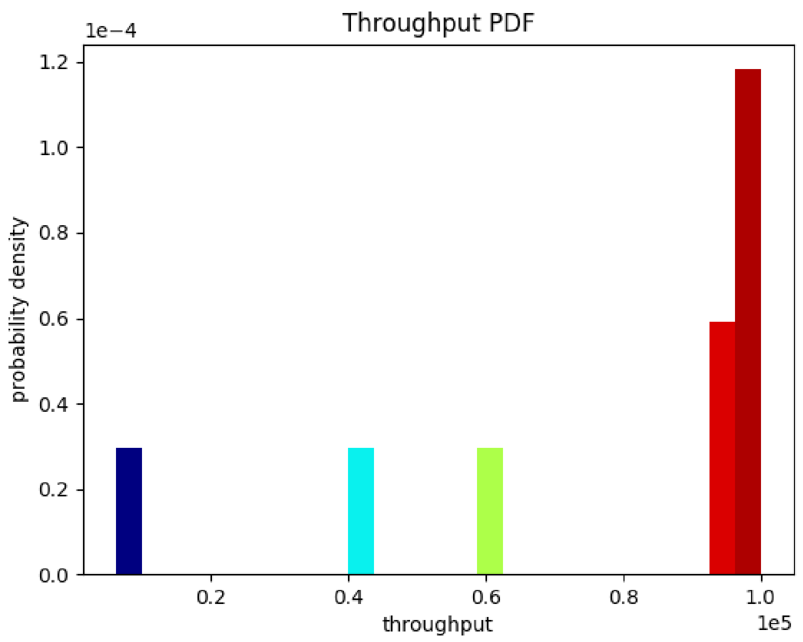
2.3 Failure Pattern 2

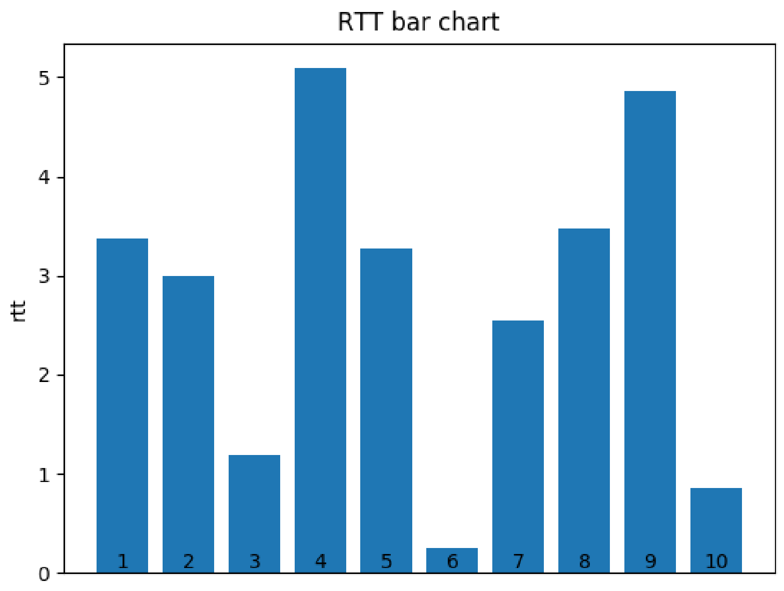
However, the failures are not permanent and show no repetitive pattern. Furthermore, during the problematic sections `uhd_find_devices` does not detect any USRPs anymore. From these two facts one can deduce the problem at least has a hardware component. Here are some plots of a typical measurement with hardware failure.

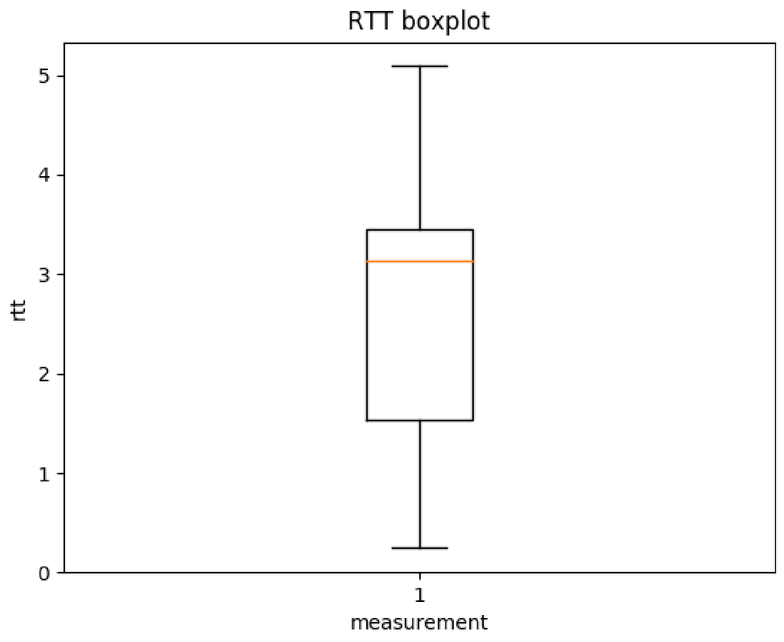
Although for 7 out of 10 measurements the throughput seems - i.e. it is NOT - reasonable a lot of ACKs got lost during the transmission. Note that due to the problem described in the previous section the RTT calculation is wrong, because lost ACKS affect the RTT value for all following data transmissions (continuous additive error on fail). The throughput is reduced by a factor 6 for unknown reasons. Compare this to the last section if you want to see what I mean (both measurements had single submeasurement durations of 100s).

Proof for lost ACKs can be found as mentioned in the RTT charts as well as in the logs. Note that the setup and parameters including receive and transmission gains were chosen no different from those in the previous section, which is a strong indication of either hardware failure or external influence which in should have been covered by choosing an unused sending frequency.




Throughput PDF
1e−4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
probability density
0.2
0.4
0.6
0.8
1.0
1e5
throughput


RTT bar chart
5
4
3
2
1
0
rtt
1
2
3
4
5
6
7
8
9
10


RTT boxplot
rtt
measurement
1
0
1
2
3
4
5

2.4 Justification of Measurement Results

2.4.1 RTT

The following section is concerns the CSMA/CA two-way handshake (no CT-S/RTS). Packet size is 1000 bytes (837 bytes payload). SIFS is set to 0.018s and DIFS to 0.006s.

In the "flawless" measurement plots both the RTT and the throughput don't differ too much from the theoretical calculation following:
Theoretical:

$$\begin{aligned} RTT &= 2 \cdot t_{prop} + t_{data} + t_{ack} + 2 \cdot t_{proc} + \text{DIFS} + \text{SIFS} \\ &= \delta_{prop} + 0.04s + 0.08s + 2t_{proc} + 0.018s + 0.006s \\ &= 0.072s + \delta_{prop} + \delta_{proc} \end{aligned}$$

Where $\delta_{prop} \approx 10^{-10}s \approx 0 \approx \delta_{proc}$.

Measured: 0.075s

2.4.2 Throughput

Theoretical (assuming RTT is correctly determined):

$$\begin{aligned} \text{throughput} &= \frac{s_{frame}}{t_{frame}} = \frac{\text{packet size}}{\text{RTT}} \\ &\approx \frac{1000B}{0.075s} \\ &\approx 13333B/s \end{aligned}$$

So for a 100s measurement the theoretical throughput should be roughly $1.3 \cdot 10^6 B$.
Measured: $6 \cdot 10^5 B$

The RTT measurements can be further backed up by measuring the RTT with linear increasing SIFS. The expected RTT then would be a linear function of SIFS.

Analogously, throughput should be a function of packet size.

Note that for both such measurements a sufficiently large measurement time must be chosen to counteract various random influences.

2.4.3 Retransmissions

Result can be roughly verified by comparing throughput versus theoretical throughput (channel busy time not taken into account with only one pair of sender and transmitter). For a multiple sender scenario one has to take backoff times into account as well.

3 Good to Know...

3.1 Bash

3.1.1 Efficient Remote Control

```
##Opening a remote connection with X forwarding
#Using -X instead of -Y adds security check, but reduces performance
#Use -C to get it more stable (gzip-compression)
#Optional -c aes128-ctr: for AES 128 encryption
#Optional -4: forces the usage of IPv4 addresses
ssh -YC4c aes-128ctr inets@134.130.223.135

##In a second terminal mount file system to local folder
#This has the advantage of the ability to treat files as though they were local
mkdir -p mnt/134.130.223.135
cd mnt/134.130.223.135

#Tip: in the next line for the last argument type . and then expand with Tab :)
#If it is necessary sshfs also allows user-mapping if file ownership is an issue
sshfs inets@134.130.223.135:/home/inets /home/alex/mnt/134.130.223.135/

#Then edit all you like as though the mounted partition was local
cd source/gr-inets/lib
atom *some_file*
```

To further avoid any repetitive commands I added to my launcher tool:

```
"inets")
    if ( mount | grep inets  )
    then
        echo "The mount point is in use, confirm unmount with your password."
        sudo umount /home/alex/mnt/134.130.223.151
    fi
    echo "Please enter the server password to mount the target directory."
    sshfs inets@134.130.223.151:/home/inets /home/alex/mnt/134.130.223.151
    gnome-terminal --tab -e "bash -c 'ssh -YC4 inets@134.130.223.151'" \
    --tab --working-directory=/home/alex/0_ba/git/measurements \
    #--tab --working-directory=/home/alex/mnt/134.130.223.151 \
    texstudio
    sleep 1
    cd ~/0_ba/git
    git pull
    git fetch
    atom ~/0_ba/git/measurements/measurement.sh
    ;;
```

3.1.2 Listing Directories Only

Stackoverflow Source

```
##Possibility 1 (fastest):
echo */
#List all subsubfolders as well:
echo */*/

##Possibility 2 (straightforward ls):
ls -d */

##Possibility 3
#where ^ means beginning of a line
ls -l | grep "^d"

##Possibility 4
#If you need to list and process all directories in a bash-script (slow)
for i in $(ls -d */); do echo ${i%%/}; done
```

3.1.3 Redirecting GRC Stuff to Files

If one wants to do heavy analysis, built-in Linux console tools including, but not limited to utilities such as `awk`, `grep`, `sed`, `cat` might be a great help. Since executing a GRC flowgraph is running a generated python script, we can achieve our goal easily by the simple means of:

```
## Just as a concept
#These could be some lines in my alohaTestSuite.sh
mkdir -p logs
time python2 theoretical_aloha_rx.py &> logs/theoretical_aloha_rx.log
time python2 theoretical_aloha.py &> logs/theoretical_aloha.log
```

3.2 Python

3.2.1 *args and **kwargs

`*args` can be used to pass an arbitrary amount of non-keyword arguments to a function like this:

```
    def my_func(*args):
        for index, value in enumerate(args):
            print( "{0}: {1}".format(index,value) )

    # let's call the function
    >>> my_func("this", "is", "an", "example")

    # output
```

```
0: this
1: is
2: an
3: example
```

`**kwargs` can be used to pass an arbitrary amount of keyword arguments to a function as follows below. Note that `**` is also used to convert dictionaries to the format specified used by `kwargs`, which is `keyword = value, keyword2 = value`.

```
def my_func(**kwargs):
    for name, value in kwargs.items():
        print( "{0} = {1}".format(name,value) )

# let's cal this function
>>> my_dict = { "fruits": ["apple","banana"], "vegetables": ["tomato", "cucumber"]}
>>> my_func(**my_dict)

# equivalent would be:
>>> my_func(fruits=["apple","banana"], vegetables=["tomato","cucumber"])

# output
fruits = ["apple", "banana"]
vegetables = ["tomato", "cucumber"]
```

Note: `*args` and `**kwargs` are conventional, thus strongly recommended, but the real syntax just is `*varname` and `**varname`.

3.2.2 List Comprehension

List Comprehension is a powerful language feature of Python. It offers an elegant way to create lists with an easy syntax.

The examples below are taken from here.

```
# Calculate powers of two
powers = [2**x for x in range(10)]
# Calculate cubic numbers
cubes = [x**2 for x in range(10)]
# Extract even numbers from cubes
even = [x for x in cubes if x%2 == 0]
```

We can also create lists of lists with list comprehension. One can also nest list comprehensions. However, for readability's sake one should not exaggerate the nesting.

```
words = "This is an example sentence with fancy words".split()
cool_list = [[word.upper(), word.lower(), len(word)] for word in words]

for item in cool_list:
        print(item)

# would create the following output:
["THIS", "this", 4]
["IS", "is", 2]
["AN", "an", 2]
["EXAMPLE", "example", 7]
["SENTENCE", "sentence", 8]
["WITH", "with", 4]
["FANCY", "fancy", 5]
["WORDS", "words", 5]
```

3.2.3 map and zip

The section summarizes this source.

`zip(*args)` takes equal length iterables and merges them into an iterable of tuples.

`map(function, iterable)` applies function to each element of iterable. It conducts the mathematical operation of mapping x to f(x).

3.3 Other

3.3.1 Atom: Folder-Wide Search for Substrings

Use the Ctrl+Shift+F shocut and all "project folders" will be searched. To add multiple folders use the Ctrl+Shift+A shortcut.

3.3.2 Using a static IP address for USRPs

```
#Get to know your interface names
ifconfig
#E.g. set eth0 interface IP to 10.0.0.100/24
#Where up opens the eth0 interface
#and is not necessary if it showed up when using infconfig
ifconfig eth0 10.0.0.100 netmask 255.255.255.0 up

###Just FYI:
#Shutdown network interface
ifconfig eth0 down
##List all interfaces
ifconfig -a
```

```
#OR
ip link show
```

3.3.3 Git Personal Access Tokens

Git access tokens are OAuth2 tokens that you can use instead of your password to perform any github operation. Furthermore, it also is possibly to restrict the permitted operations of a token on the github settings page. Let me walk you through the whole process:

1 Go to https://github.com/settings/tokens

2 Create your personal access token with the desired permissions.

3 E.g. use your token like this:

```
git push https://username:token@github.com/username/project.git
```

Note: Don't include in any source code you upload to github. It will scan your code for tokens and immediately revoke any active token they find in your code.

3.3.4 Latex input and include

Here's the source.

> `\input{filename}` imports the commands from filename.tex into the target file; it's equivalent to typing all the commands from filename.tex right into the current file where the `\input` line is.
>
> `\include{filename}` essentially does a `\clearpage` before and after `\input{filename}`, together with some magic to switch to another .aux file, and omits the inclusion at all if you have an `\includeonly` without the filename in the argument. This is primarily useful when you have a big project on a slow computer; changing one of the include targets won't force you to regenerate the outputs of all the rest.
>
> `\include{filename}` gets you the speed bonus, but it also can't be nested, can't appear in the preamble, and forces page breaks around the included text.