AGH University of Science and Technology

FACULTY OF COMPUTER SCIENCE, ELECTRONICS AND TELECOMMUNICATIONS

DEPARTMENT OF TELECOMMUNICATIONS

Engineering Thesis

Impact of Periodic Interference on IEEE 802.11ax Networks

Wpływ cyklicznych interferencji na działanie sieci standardu IEEE 802.11ax

Author:	Filip Nożkiewicz
Degree programme:	ICT
Supervisor:	Szymon Szott PhD

Kraków 2019

Uprzedzony o odpowiedzialnosci karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całosci lub częsci cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystycznego wykonania albo publicznie zniekształca taki utwór, artystycznego wykonanie, fonogram, wideogram lub nadanie.", a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictie wyższym (t,j, Dz. U. z 2012r. poz. 572, z późń. zm.): "Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialnośc dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskimsamorządu studenckiego, zwanym dalej «sądem koleżeńskim».",oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wyienione w pracy.

I would like to express my sincere gratitude to my supervisor, Szymon Szott, for the continuous support and guidance during research and writing of this thesis.

Table of contents

1. Introduction

2. Background

3. Testbed description

Doing Researches require proper testing environment, which is able to run required test-cases and is relatively comfortable to use. The vital thing is that the results must be reliable and can give desired conclusions about particular problems. In this case ns-3 simulator has been chosen. This chapter focuses on an overview of this tool and scenario prepared using it.

3.1 Overview of the ns-3 simulator

Ns-3 simulator is an open source tool written in C++ programming language with bindings available for Python.It is targeted primarily for research and educational use and is under license of GNU GPLv2. Ns-3 is a discrete-event simulator, so models the operation of a system as a discrete sequence of events in time. Interface of this tool are files with extension of 'cc' or 'py', so models and simulations are directly written in C++ or Python programming languages. Simulation's events are executed in order which is determined by scheduler and the simulation stops when all of the events finish. There is a documentation for ns-3 available on it's official page as well as many scripts and examples. The only operating system which is supported by ns-3 is a Linux. During writing of the thesis 'Ubuntu 16.0.54 LTS' version was used and the version of simulator was 3.29.

3.2 Scenario description

In order to research specified in the title of the thesis, the particular scenario has been configured. The conditions of simulation's were dynamically changing so automation was needed to run simulation relatively fast and without additional manual chcnages in test scenario. There were separated tools involved to observe specific behaviours and analyze results like statistics libraries of python and wireshark. Everything had to be done in a proper order starting from configuring the scenario, writing automation script and proceeding and analyze essential data produced in time of simulation.

3.2.1 Topology

Topology consists of two wireless Access Points. The first one is to receive traffic from slow stations and the second one from fast stations. Slower stations send data with much lower rate. Simulation scenario includes those 2 types of stations to observe different explore how stations with completely different data rates behave when it comes to share frequency range with LTE signals. There are also cases where there is only one Access Point. The number of stations is variable and can be changed whenever it is needed also while simulation is running. Each staion send data to it's own access point.

Essential part of the topology is an interferer which is configured to act as a LTE base station and generate signals which interfere with 802.11 transmission. In some scenarios Interferer must be excluded so and then it's transmit power is decreased to zero. Basic Topology is shown in Figure 3.1.

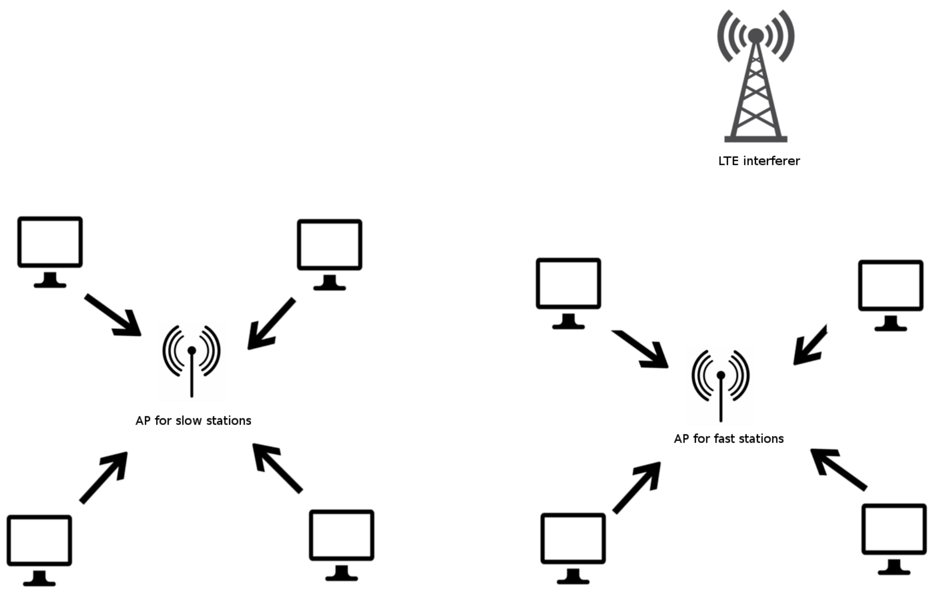
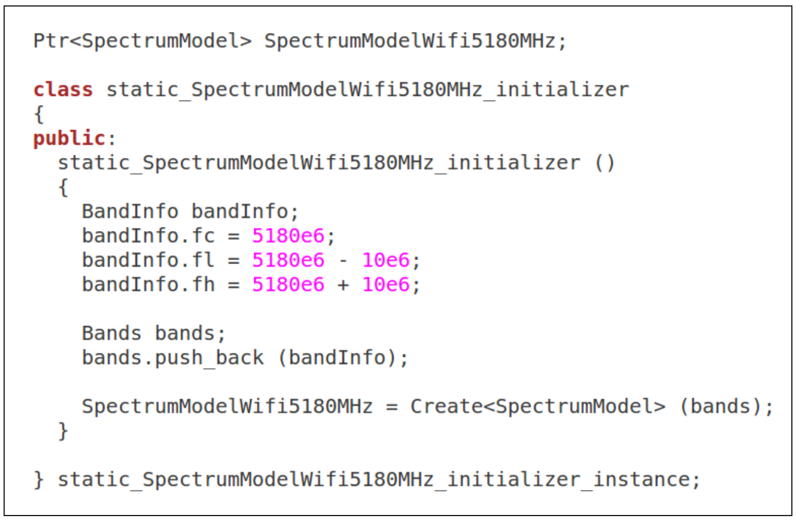


Figure 3.2.1: Basic topology

3.2.2 Code description

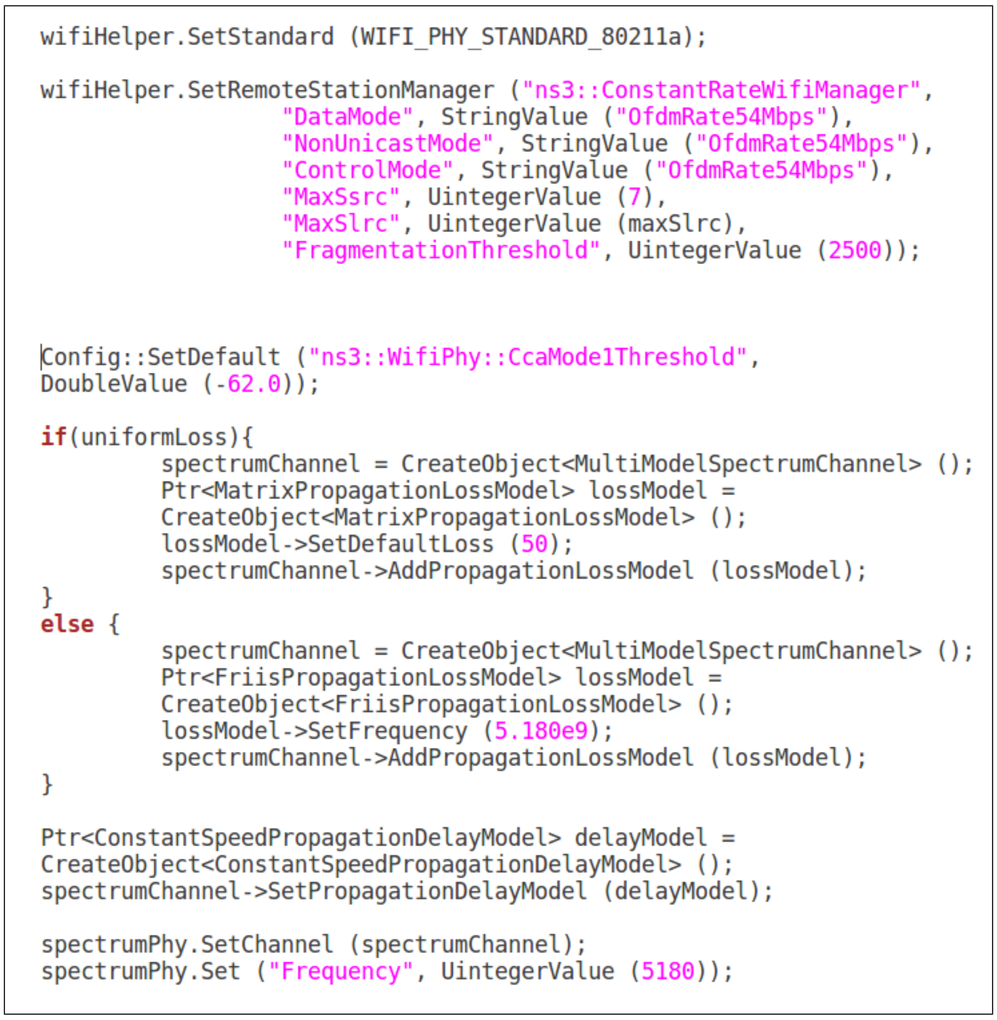
Figure 3.2 shows part of the code which is responsible for generating particular frequency range which is used during simulation and specific channel width. In 802.11 ax channel width can be choosen from 20 MHz, 40 MHz, 80 MHz, 80+80 MHz and 160 MHz.



```
Ptr<SpectrumModel> SpectrumModelWifi5180MHz;

class static_SpectrumModelWifi5180MHz_initializer
{
public:
  static_SpectrumModelWifi5180MHz_initializer ()
  {
    BandInfo bandInfo;
    bandInfo.fc = 5180e6;
    bandInfo.fl = 5180e6 - 10e6;
    bandInfo.fh = 5180e6 + 10e6;

    Bands bands;
    bands.push_back (bandInfo);

    SpectrumModelWifi5180MHz = Create<SpectrumModel> (bands);
  }

} static_SpectrumModelWifi5180MHz_initializer_instance;
```

Figure 3.2.2: Spectrum generator

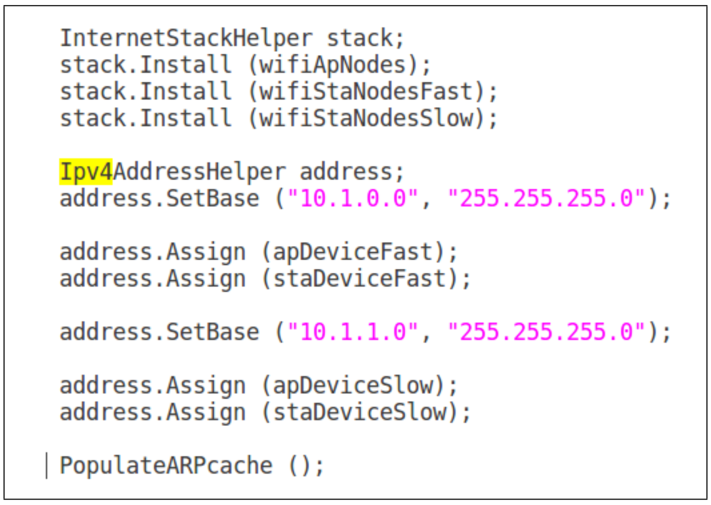
Next essential step was to configure Physical layer. Figure 3.2.3 demonstrate sample settings with built-in class WifiHelper. On this stage must be set parameters like MCS (modulation coding scheme), fragmentation treshold and any other parameter which is strictly connected to physical layer. PrpagationLossModel and PropagationDelayModel are classes provided by ns-3 which models the propagation loss and delay thrue the transmission medium.



```
wifiHelper.SetStandard (WIFI_PHY_STANDARD_80211a);

wifiHelper.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
                  "DataMode", StringValue ("OfdmRate54Mbps"),
                  "NonUnicastMode", StringValue ("OfdmRate54Mbps"),
                  "ControlMode", StringValue ("OfdmRate54Mbps"),
                  "MaxSsrc", UintegerValue (7),
                  "MaxSlrc", UintegerValue (maxSlrc),
                  "FragmentationThreshold", UintegerValue (2500));



Config::SetDefault ("ns3::WifiPhy::CcaMode1Threshold",
DoubleValue (-62.0));

if(uniformLoss){
         spectrumChannel = CreateObject<MultiModelSpectrumChannel> ();
         Ptr<MatrixPropagationLossModel> lossModel =
         CreateObject<MatrixPropagationLossModel> ();
         lossModel->SetDefaultLoss (50);
         spectrumChannel->AddPropagationLossModel (lossModel);
}
else {
         spectrumChannel = CreateObject<MultiModelSpectrumChannel> ();
         Ptr<FriisPropagationLossModel> lossModel =
         CreateObject<FriisPropagationLossModel> ();
         lossModel->SetFrequency (5.180e9);
         spectrumChannel->AddPropagationLossModel (lossModel);
}

Ptr<ConstantSpeedPropagationDelayModel> delayModel =
CreateObject<ConstantSpeedPropagationDelayModel> ();
spectrumChannel->SetPropagationDelayModel (delayModel);

spectrumPhy.SetChannel (spectrumChannel);
spectrumPhy.Set ("Frequency", UintegerValue (5180));
```

Figure 3.2.3: physical layer settings

Network layer configuration consisted of creating separated ipv4 address range for slow stations and fast stations and setup arp caching in order to make communication between data link layer and network layer possible . Configuration of network layer illustrates Figure 3.2.4.

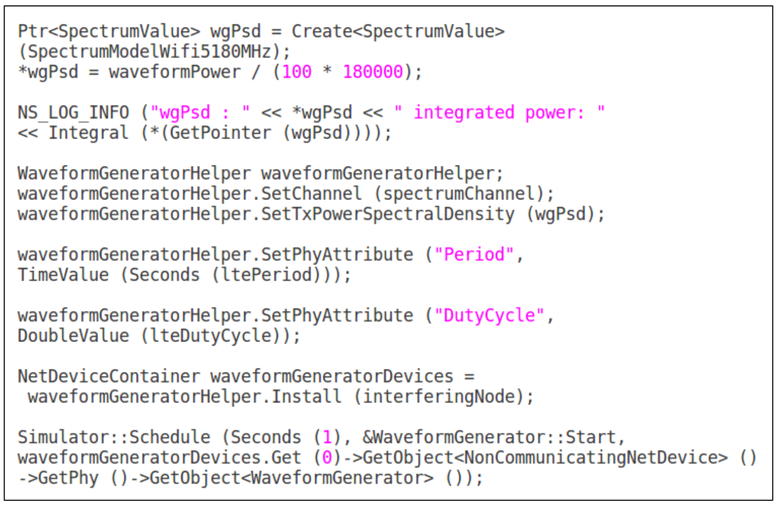


```
InternetStackHelper stack;
stack.Install (wifiApNodes);
stack.Install (wifiStaNodesFast);
stack.Install (wifiStaNodesSlow);

Ipv4AddressHelper address;
address.SetBase ("10.1.0.0", "255.255.255.0");

address.Assign (apDeviceFast);
address.Assign (staDeviceFast);

address.SetBase ("10.1.1.0", "255.255.255.0");

address.Assign (apDeviceSlow);
address.Assign (staDeviceSlow);

PopulateARPcache ();
```

Figure 3.2.4: network layer configuration

Transport and Application layer were configure in a single function. Ns-3 has built-in class called InetSocketAdress which was used to create layer 4 socket using port and address of specific node. This 2 values were being incremented and assign to each station. All the data transmission was done with UDP protocol. Application layer has been configured with ApplicationContainer class which allow to set required time to start and optional time to stop simulation.

Lte interferer was created due to the WaveformGeneratorHelper class. It is used to inject specific noise to the channel. It allows to set Spectral power density and physical parameters like period and dutycycle. Finally is installed on a NetDeviceContainer which includes all configured stations. After that when specific lever of power is set waveformgenerato is able to disrupt wifi transmission. Sample settings are presented in Figure 3.2.5.



```
Ptr<SpectrumValue> wgPsd = Create<SpectrumValue>
(SpectrumModelWifi5180MHz);
*wgPsd = waveformPower / (100 * 180000);

NS_LOG_INFO ("wgPsd : " << *wgPsd << " integrated power: "
<< Integral (*(GetPointer (wgPsd))));

WaveformGeneratorHelper waveformGeneratorHelper;
waveformGeneratorHelper.SetChannel (spectrumChannel);
waveformGeneratorHelper.SetTxPowerSpectralDensity (wgPsd);

waveformGeneratorHelper.SetPhyAttribute ("Period",
TimeValue (Seconds (ltePeriod)));

waveformGeneratorHelper.SetPhyAttribute ("DutyCycle",
DoubleValue (lteDutyCycle));

NetDeviceContainer waveformGeneratorDevices =
 waveformGeneratorHelper.Install (interferingNode);

Simulator::Schedule (Seconds (1), &WaveformGenerator::Start,
waveformGeneratorDevices.Get (0)->GetObject<NonCommunicatingNetDevice> ()
->GetPhy ()->GetObject<WaveformGenerator> ());
```

Figure 3.2.5: LTE interferer configuration

3.2.3 Input parameters

To make simulations more comfortable to use and overall process more efficient group of parameters had been defined. Those parameters were being changed before and while running simulation due to the automation process. Their values needed to be present in output results in order to group data by specific values. Parameters are presented in Figure 3.2.7.

```
./waf --run "scratch/lte-wifi --RngRun=1 --offeredLoad=40 --nFast=3"
```

Figure 3.2.6: Run simulation script with command line arguments

```
bool enableRtsCts;  //parameters which enable
//or desable RTS/CTS frames
int nFast; //number Fast stations
int nSlow; //number of slow stations Slow stations
bool debug; // activate or deactivate debug mode
bool pcap; // enable or disable gathering info
// about data flow o the files
bool xml; // activate xml mode
string offeredLoad; //data offered load
string mcs;// used modulation coding scheme
int radius; //radius of station placement [m]
bool uniformLoss = false;
double waveformPower; //trasnmitting power used by interferer
double ltePeriod; // period of lte transmission [s]
double lteDutyCycle; // LTE duty cycle [%]
string outputCsv; // results output file
int maxSlrc = 7;
int packetSize = 1500; // ip packet size [B]
unsigned int RngRun; // add randomness to the simulation
```

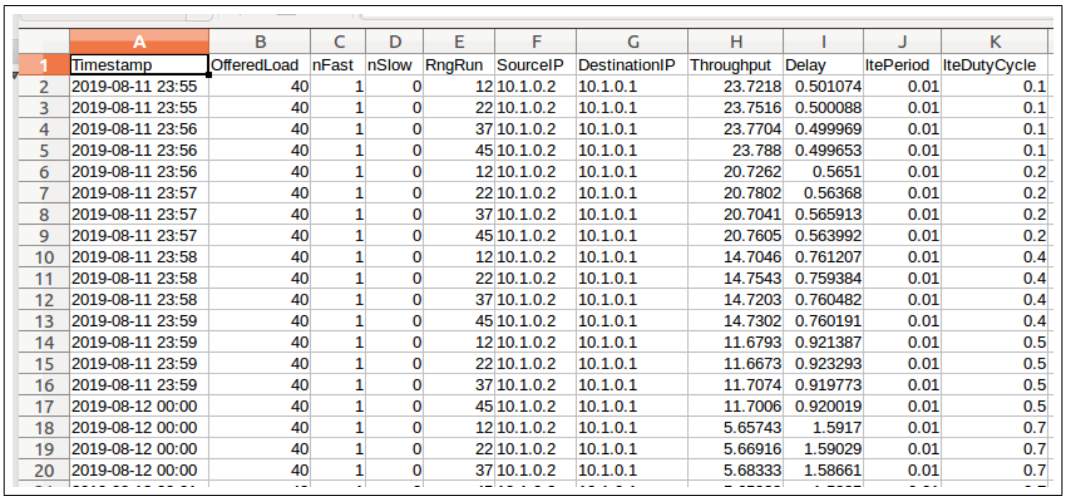
Figure 3.2.7: Configuration parameters

3.2.4 Flow Monitor

FlowMonitorHelper is a class in ns-3 which helps to enable ip monitoring on a set of nodes. Iw was used in all of the simulation. FlowMonitor has 'starttime' attribute which is usually set to value larger than 'starttime' of simulation because there is a need to wait a warmup time to get only reliable data. Data were being dumpded to the csv files during simulation run. Ipv4FlowClassifier is a tool which helped to properly classify data before dumping it to the file. This class differentiate single flows by their parameters. Packets are assigned to specific flows and Flow is characterized by Five element tuple (Source IP , Destination IP , Source Port , Destination Port , Transport Layer Protocol) . Classifier check what is the time of first and last transmitted packet as well as first and last received packet , check numjber of packets being transmitted and received which is usefull to count loss. Parameters which were given as an input are also taken into account because they are assigned to particular flows. Figure 3.2.8 presents configuration of the classifier and Figure 3.2.9 part of the csv file with gathered simulation data.

```
Ptr<Ipv4FlowClassifier> classifier =
DynamicCast<Ipv4FlowClassifier> (flowmon.GetClassifier ());
std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats ();
for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator i = stats.begin (); i != stats.end (); ++i)
 {
        Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (i->first);
        flowThr=i->second.rxPackets * packetSize * 8.0 /
        (i->second.timeLastRxPacket.GetSeconds ()
         - i->second.timeFirstTxPacket.GetSeconds ()) / 1000 / 1000;

        flowDel=i->second.delaySum.GetSeconds () / i->second.rxPackets;

        NS_LOG_UNCOND ("Flow " << i->first  << " (" << t.sourceAddress
        << " -> " << t.destinationAddress << ")\tThroughput: " <<  flowThr
          << " Mbps\tTime: " << i->second.timeLastRxPacket.GetSeconds ()
        - i->second.timeFirstTxPacket.GetSeconds () << "\tDelay: " << flowDel
        << " s\tTx packets " << i->second.txPackets << " s\tRx packets "
        << i->second.rxPackets << "\n");

        myfile << std::put_time(&tm, "%Y-%m-%d %H:%M") << "," << offeredLoad
        << "," << nFast << "," << nSlow << "," << RngSeedManager::GetRun() << ","
        << t.sourceAddress << "," << t.destinationAddress << "," << flowThr << ","
        << flowDel << "," << ltePeriod << "," << lteDutyCycle;
        myfile << std::endl;
}
myfile.close();
```

Figure 3.2.8: Run simulation script with command line arguments

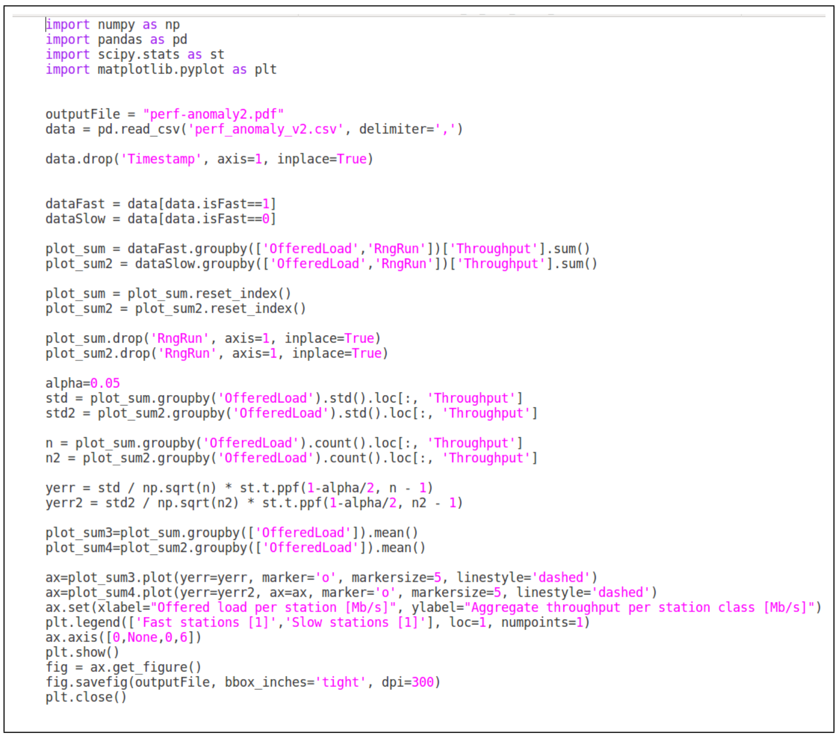


	A	B	C	D	E	F	G	H	I	J	K
1	Timestamp	OfferedLoad	nFast	nSlow	RngRun	SourceIP	DestinationIP	Throughput	Delay	ltePeriod	lteDutyCycle
2	2019-08-11 23:55	40	1	0	12	10.1.0.2	10.1.0.1	23.7218	0.501074	0.01	0.1
3	2019-08-11 23:55	40	1	0	22	10.1.0.2	10.1.0.1	23.7516	0.500088	0.01	0.1
4	2019-08-11 23:56	40	1	0	37	10.1.0.2	10.1.0.1	23.7704	0.499969	0.01	0.1
5	2019-08-11 23:56	40	1	0	45	10.1.0.2	10.1.0.1	23.788	0.499653	0.01	0.1
6	2019-08-11 23:56	40	1	0	12	10.1.0.2	10.1.0.1	20.7262	0.5651	0.01	0.2
7	2019-08-11 23:57	40	1	0	22	10.1.0.2	10.1.0.1	20.7802	0.56368	0.01	0.2
8	2019-08-11 23:57	40	1	0	37	10.1.0.2	10.1.0.1	20.7041	0.565913	0.01	0.2
9	2019-08-11 23:57	40	1	0	45	10.1.0.2	10.1.0.1	20.7605	0.563992	0.01	0.2
10	2019-08-11 23:58	40	1	0	12	10.1.0.2	10.1.0.1	14.7046	0.761207	0.01	0.4
11	2019-08-11 23:58	40	1	0	22	10.1.0.2	10.1.0.1	14.7543	0.759384	0.01	0.4
12	2019-08-11 23:58	40	1	0	37	10.1.0.2	10.1.0.1	14.7203	0.760482	0.01	0.4
13	2019-08-11 23:59	40	1	0	45	10.1.0.2	10.1.0.1	14.7302	0.760191	0.01	0.4
14	2019-08-11 23:59	40	1	0	12	10.1.0.2	10.1.0.1	11.6793	0.921387	0.01	0.5
15	2019-08-11 23:59	40	1	0	22	10.1.0.2	10.1.0.1	11.6673	0.923293	0.01	0.5
16	2019-08-11 23:59	40	1	0	37	10.1.0.2	10.1.0.1	11.7074	0.919773	0.01	0.5
17	2019-08-12 00:00	40	1	0	45	10.1.0.2	10.1.0.1	11.7006	0.920019	0.01	0.5
18	2019-08-12 00:00	40	1	0	12	10.1.0.2	10.1.0.1	5.65743	1.5917	0.01	0.7
19	2019-08-12 00:00	40	1	0	22	10.1.0.2	10.1.0.1	5.66916	1.59029	0.01	0.7
20	2019-08-12 00:00	40	1	0	37	10.1.0.2	10.1.0.1	5.68333	1.58661	0.01	0.7

Figure 3.2.9: Configuration parameters

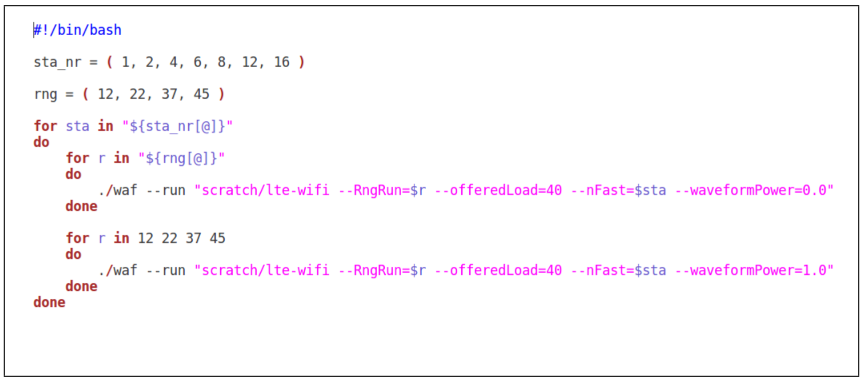
3.2.5 Automation

To run simulations without constant manual changes, automation scripts were created. Because ns-3 works on a Linux operating system scripts were written in bash. The process of automation consist of defining sets of parameters and loops inside which were commands starting simulation scenario with parameters from previously define sets. Results were usually appended to the csv file and when simulation were finished data were proceeded and analyzed with python script. Python scripts were based mostly on modules like pandas, matplotlib and scipy. Scripts generated plots and graphs which presented data in a form that enable to find dependencies between them and draw final conclusions.



```python
import numpy as np
import pandas as pd
import scipy.stats as st
import matplotlib.pyplot as plt


outputFile = "perf-anomaly2.pdf"
data = pd.read_csv('perf_anomaly_v2.csv', delimiter=',')

data.drop('Timestamp', axis=1, inplace=True)


dataFast = data[data.isFast==1]
dataSlow = data[data.isFast==0]

plot_sum = dataFast.groupby(['OfferedLoad','RngRun'])['Throughput'].sum()
plot_sum2 = dataSlow.groupby(['OfferedLoad','RngRun'])['Throughput'].sum()

plot_sum = plot_sum.reset_index()
plot_sum2 = plot_sum2.reset_index()

plot_sum.drop('RngRun', axis=1, inplace=True)
plot_sum2.drop('RngRun', axis=1, inplace=True)

alpha=0.05
std = plot_sum.groupby('OfferedLoad').std().loc[:, 'Throughput']
std2 = plot_sum2.groupby('OfferedLoad').std().loc[:, 'Throughput']

n = plot_sum.groupby('OfferedLoad').count().loc[:, 'Throughput']
n2 = plot_sum2.groupby('OfferedLoad').count().loc[:, 'Throughput']

yerr = std / np.sqrt(n) * st.t.ppf(1-alpha/2, n - 1)
yerr2 = std2 / np.sqrt(n2) * st.t.ppf(1-alpha/2, n2 - 1)

plot_sum3=plot_sum.groupby(['OfferedLoad']).mean()
plot_sum4=plot_sum2.groupby(['OfferedLoad']).mean()

ax=plot_sum3.plot(yerr=yerr, marker='o', markersize=5, linestyle='dashed')
ax=plot_sum4.plot(yerr=yerr2, ax=ax, marker='o', markersize=5, linestyle='dashed')
ax.set(xlabel="Offered load per station [Mb/s]", ylabel="Aggregate throughput per station class [Mb/s]")
plt.legend(['Fast stations [1]','Slow stations [1]'], loc=1, numpoints=1)
ax.axis([0,None,0,6])
plt.show()
fig = ax.get_figure()
fig.savefig(outputFile, bbox_inches='tight', dpi=300)
plt.close()
```

Figure 3.2.10: Sample python script for data analysis



```
#!/bin/bash

sta_nr = ( 1, 2, 4, 6, 8, 12, 16 )

rng = ( 12, 22, 37, 45 )

for sta in "${sta_nr[@]}"
do
    for r in "${rng[@]}"
    do
        ./waf --run "scratch/lte-wifi --RngRun=$r --offeredLoad=40 --nFast=$sta --waveformPower=0.0"
    done

    for r in 12 22 37 45
    do
        ./waf --run "scratch/lte-wifi --RngRun=$r --offeredLoad=40 --nFast=$sta --waveformPower=1.0"
    done
done
```

Figure 3.2.11: Sample automation script in bash

4. Simulations and Analysis

5. Summary and Conclusions