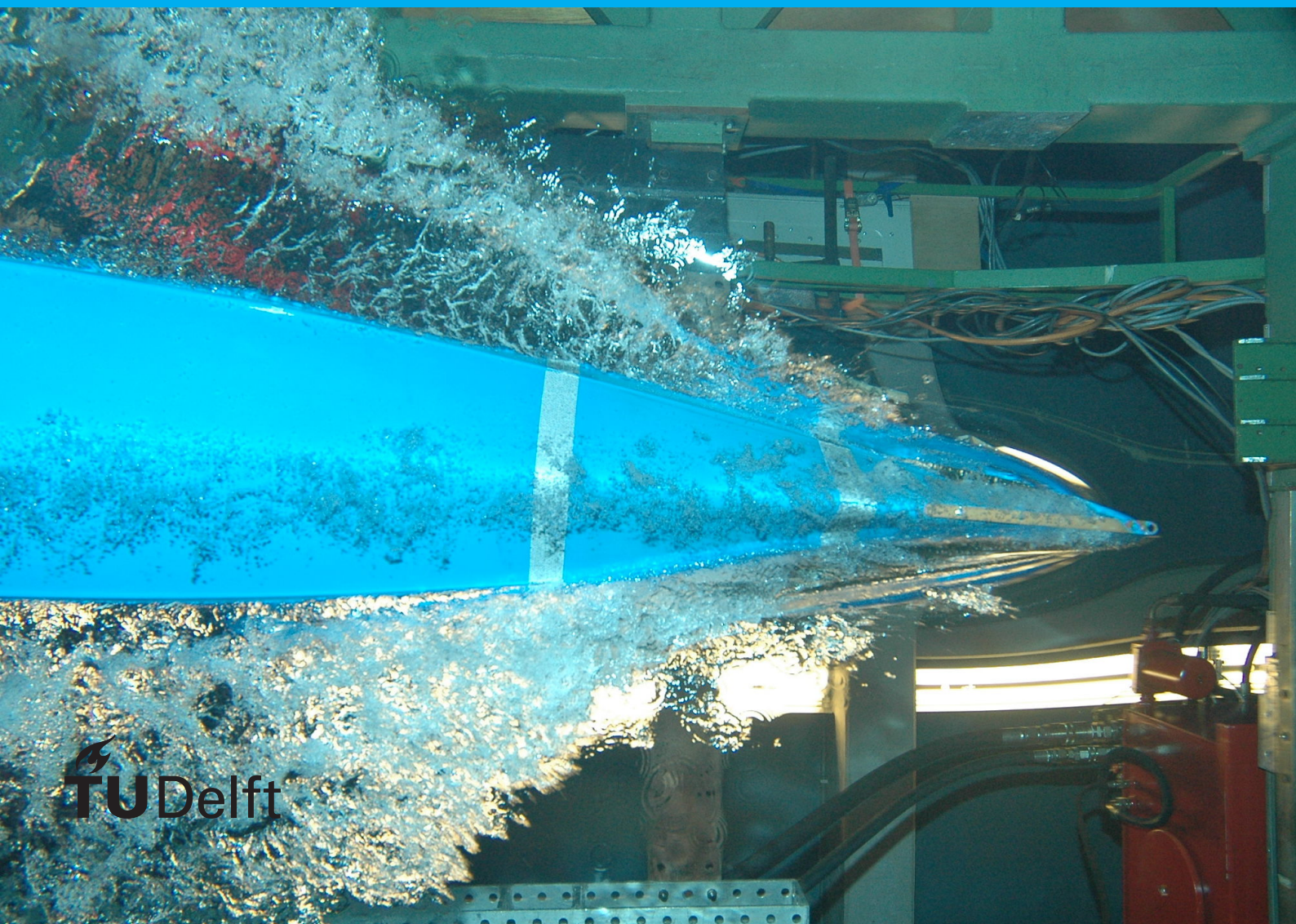
NS-3 Assignment

J.B Miog



NS-3 Assignment

by

J.B Miog

.

Student number: 4336313
Date: April 2016

An electronic version of this report and the project files are available at
`https://github.com/JBmiog/ET4394-Wireless-Networking`.

TU Delft

Contents

1 Introduction

1.1. Assignment description

This assignment is one of two assignments given in the course of Wireless Networks [4393] at the Technical University of Delft. The assignment consist of writing a simulation in C++, using the NS-3 Network simulator, in order to simulate IEEE802.11b properties with respect to changing network properties.

1.2. The NS-3 network simulator

NS-3 is a discreet-event network simulator, which is free for download at `https://www.nsnam.org/`. It contains models for Wi-Fi and Wi-Max, among others. The software is licensed under the GNU GPLv2 license. Although simulation files are originally written in c++, recent features also support python scripts. The simulator supports WiFi simulation up to layer 3. During this project, version ns-3.24.1 is used.

1.3. 802.11b specification

802.11b operates in the 2.4 GHz band and is capable of transferring data with rates up to 11Mbps, if this does not succeed, it can change to data rates of 5.5Mbps, 2Mbps, or 1 Mbps. 802.11b uses the CSMA/CA technique to let nodes communicate. It is Complementary Code Keyed (CCK) modulated, which is based on Direct Sequence Spread Spectrum modulation (DSSS) [1].

1.4. Github page

The project files for this course can be found on my Github page, `https://github.com/JBmiog/ET4394-Wireless-Networking`

2

project description

2.1. Objective

In this simulation, results are obtained to formulate an answer to the following question;

what happens if an encreasing amount of AP fail in a network with respect to - the average througput per node - regarding which data rate's? - constant - regarding which packet size

What happens to the average throughput of stations in a 802.11b network when an increasing number of Acces Points (AP's) fail, with respect to the different data rates and different packet sizes?

During simulation, the amount of nodes stays constantly 30, while the number of AP's decreases. The failure of an AP is mimicked by redoing the simulation with an altered distribution of stations, with one or more AP less.

2.2. Hypothesis

As the number of nodes per AP will increase, the average throughput of all nodes is expected to decrease. Because now, the CSMA/CA machanism backs off more often, having larger contention windows, which will further decrease the throughput.

3

Implementation

3.1. Overview

The simulation consist of multiple runs, each run corresponds to pre-determined amount of access points k and stations n as depicted in figure 3.1. After each run, the average throughput is measured.

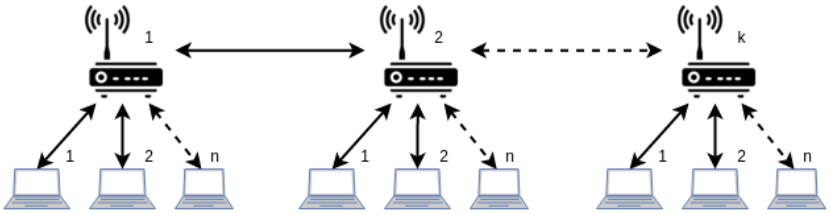


Figure 3.1

3.2. Script description

The simulator script is heavily based on the wifi-wired-bridging.cc file that can be found in the examples/wireless folder of the NS3 installation. The nodes are arbitrarily instantiated using the corresponding containers and helpers, which, for the purpose of this simulation, are placed in a C++ std::vector variable for easy scalability. The AP's that form the backbone architecture of this simulation are suited with a CSMA NetDevice, and form a CSMA network. This is choosen as the CSMA implementation in NS3 are instantaneous, faster than light, and transmission do not get jammed [2]. Thus, during the simulation we can purely focus on the 802.11b, and can leave IEEE 802.3 inter AP communication issues out of scope. In reality, this would not be fair, but for this simulation focus is laid on the station to AP communication.

YansWifiHelper and YansChanneHelper are used in order to make the connection between AP and stations, as well as defining the properties of the channels.

Next, the Mobility helper is used to let the nodes move around in a pre-defined square, and the stations are connected to the AP's. Next, IPv4 packets are generated by an application which is installed on each node using the OnOffHelper class. The packets are transmitted to access point k, the last AP added in the simulation.

Unfortunately, the Random Mobility Model does not provide a random walk every run, as can be seen using NetAnim. It only provide one random walk, and thus, simulation results of multiple runs result in the same output.

The average throughput of all nodes is calculated using FlowMonitor.

The average throughput is measured as:

$average_throughput = (throughput_of_all_flows / number_of_flows)$

4

Results

4.1. Througput vs. Stations per Access Points

with a constant amount of 30 stations, the following average throughput rates are obtained for the 802.11b supported 11, 5.5, 2, and 1 Mbps data rates, with respect to the throughput and the number of stations per access point. This simulation uses the random direction 2d mobility model, with a constant payload size of 512 byte. the results are shown in 4.1.

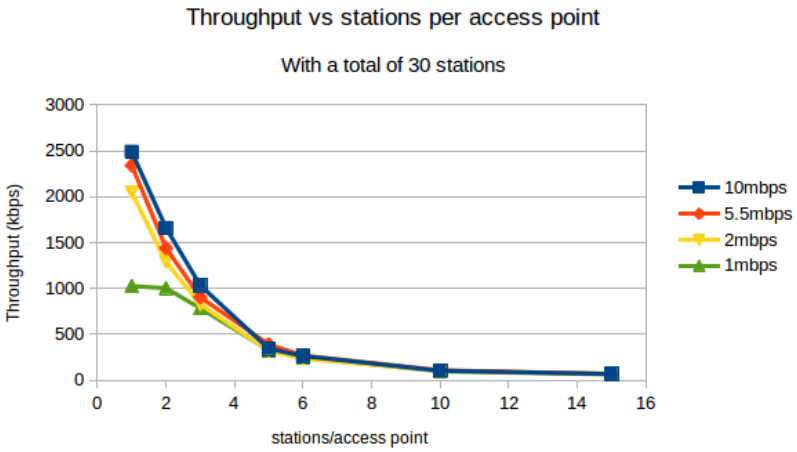


Figure 4.1

This figure shows that when an increasing number of AP fail in the network, and the stations are divided among the remaining nodes, the average throughput goes done exponentially, for four different standard data rates.

In order to obtain a clean results, certain station/AP ratio's are omitted as these would result in stations having a fractional component in the number of stations connected to them. As can be seen in the figure, at least one station is connected to one AP, and at least two AP's are active in the network.

4.2. Throughput vs. packet size

The results depicted in 4.2 are obtained by setting the data rate to 11Mbps, and using the random direction 2d mobility model.Three different packet sizes, being 1472, 1000 and 500 bytes are selected for simulation. The figure shows that a larger packet size leads to a better average throughput when relatively few stations are connected to each AP. With an increasing amount of stations/AP, we see that the size of the packets does not effect the average throughput as much.

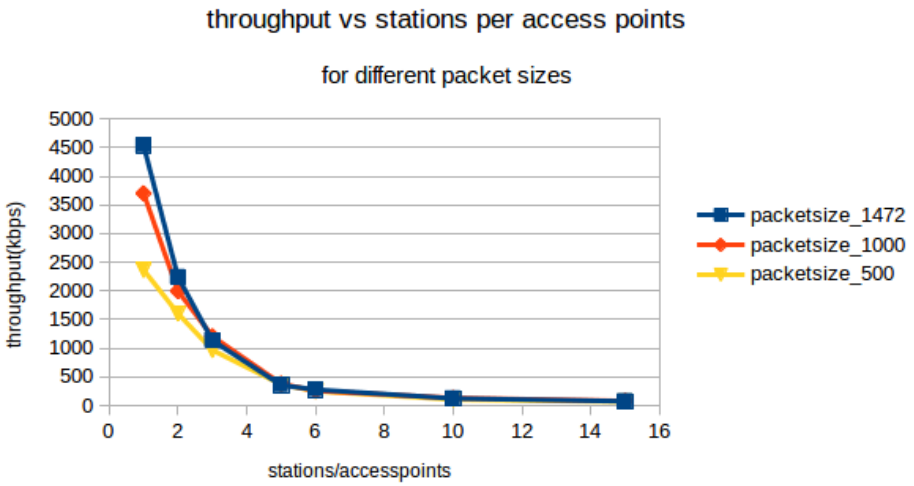

throughput vs stations per access points
for different packet sizes
throughput(kbps)
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
0
2
4
6
8
10
12
14
16
stations/accesspoints
packetsize_1472
packetsize_1000
packetsize_500

Figure 4.2

5

Conclusion

From the results obtained by the simulations, we can draw the conclusion that 802.11b, using IPv4 packets, throughput decrease when more stations join the same access points. This is simulated with a variety of stations per access point, three different packet sizes, four different data rates and 30 stations in total sending packets to one node in the network.

6

Reflection and tips

6.1. About the randomness in the mobility models

As stated in the project assignment, the goal had been to also display reliable data by repeating the simulation and show the variance in the obtained results. Unfortunately, the randomness mobility follows the same "random path" (as observed in NetAnim) every time I run the simulation. Up to this point, the variance, which is zero, is there for discarded. Until the randomness issue is solved.

6.2. Experience with NS3

As my network simulation knowledge was limited to the Cisco simulators, I had expected this simulator to be a bit more user friendly. The lack of drag-and-drop GUI in NS-3 negatively effected the time needed to test some easy situations, and gain some insight in the inner workings.

Both easy and complicated scenario's seem relatively complex to simulate. The simulator does not provide to the point debug information very ofter. Valid attributes for instances are uneasy to find, as you'll have to compile the code first before you can request the possible attributes using WAF. Which, to me, feels a little strange. I think most of these issues can be resolved if the NS-3 tutorial would hint to a code editor/compiling GUI.

The Python support does not really pave the way for a smooth simulation setup, as again, errors are hard to comprehend and the cause is not always pointed out at once.

Often, I've experienced that a trial and error approach produced best results when combined with existing pieces of code. The preferred approach of designing the simulation and implement it while trusting on debugging tools seemed futile. The compiler quickly referred to segmentation errors, and pointed to Valgrind as the solution.

6.3. Project road map

Initially, I had a hard time trying to disconnect one AP while the simulation is running, and let the stations reconfigure themselves. Unfortunately, no possible implementation is found, as NS-3 feedback often only indirectly points at the source of the problem. Therefor I had to decide to run the simulation multiple times and distribute the stations among the AP's in a hard coded fashion for each run.

I started writing the simulation in python, but eventually I was forced to obtain a deeper knowledge of the C++ language before continuing to write the simulation. I decided to start all over again and write the whole simulation in C++. This step by step approach didn't work out smoothly.

Lastly, I found myself in the situation in which I was modifying a pile of existing C++ examples, forcing them to act in a proper way to fulfill the simulation requirements. If one implementation did not provided

the results needed, I would go on and find another example that could be extended.

Trying to add multiple AP's to one network had been rather hard. Luckily, I came across the wifi-wired-briding.cc example, which already implemented a large part of the backbone architecture.

6.4. Tips

I would like to close this report stating some simple tips for future . - Use C++ for implementation, python integration does offer a reliable/usable alternative jet.
- Use NetAnim for animations, which will reveal that the random paths are identical with each simulation.
- Use Eclipse as code editor (and as code editor only) as it provides hints, auto completion, templates, typeId's and errors, which are invisible in other text editors. Opening Ns-3 as an eclipse project is described by Hitesh Choudhary on youtube, `https://www.youtube.com/watch?v=npv8gBoySyk`, who has great introduction video's to the NS-3 simulator as well.
- Build upon existing code, rather than figuring everything out from the start.
- Use FlowMonitor to obtain measurement results.

7

Simulation

The simulation can be started by issuing the following command:

./waf - -run "scratch/wifi-wired-bridging" &>results.csv

The results are piped into results.csv which can be by most spreadsheet editors.

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License version 2 as
* published by the Free Software Foundation;
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
*/

//
// Default network topology includes some number of AP nodes specified by
// the variable nWifis (defaults to two).  Off of each AP node, there are some
// number of STA nodes specified by the variable nStas (defaults to two).
// Each AP talks to its associated STA nodes.  There are bridge net devices
// on each AP node that bridge the whole thing into one network.
//
// Enabling debug while running the script will enable a single run
// simulation, otherwise it'll go through the pre-defined runs
// and output data dat can be piped into a csv file. with &> "...". txt
//
//      +-------+      +-------+            +-------+      +-------+
//      | STA |       | STA |              | STA |       | STA |
//      +-------+      +-------+            +-------+      +-------+
//    192.168.0.2  192.168.0.3         192.168.0.5  192.168.0.6
//      -----------    -----------        -----------    -----------
//      WIFI STA      WIFI STA            WIFI STA      WIFI STA
//      -----------    -----------        -----------    -----------
//         ((*))         ((*))       |      ((*))          ((*))
//                                   |
//                ((*))              |             ((*))
//               -------                          -------
//               WIFI AP   CSMA ========= CSMA    WIFI AP  ============== k bridges with nStas
//               -------   -----          -----   -------
//               ##############           ##############
//                   BRIDGE                   BRIDGE
//               ##############           ##############
//                 192.168.0.1              192.168.0.4
//                +-----------+            +-----------+
//                | AP Node |                | AP Node |
//                +-----------+            +-----------+
//

#include <stdint.h>
#include <iostream>
#include <map>
#include <sstream>
#include <string>
#include <vector>

#include "../build/ns3/address.h"
#include "../build/ns3/application-container.h"
#include "../build/ns3/boolean.h"
#include "../build/ns3/bridge-helper.h"
#include "../build/ns3/command-line.h"
#include "../build/ns3/csma-helper.h"
#include "../build/ns3/data-rate.h"
#include "../build/ns3/double.h"
#include "../build/ns3/flow-classifier.h"
#include "../build/ns3/flow-monitor.h"
#include "../build/ns3/flow-monitor-helper.h"
#include "../build/ns3/inet-socket-address.h"
```

```
#include "../build/ns3/internet-stack-helper.h"
#include "../build/ns3/ipv4-address.h"
#include "../build/ns3/ipv4-address-helper.h"
#include "../build/ns3/ipv4-flow-classifier.h"
#include "../build/ns3/ipv4-interface-container.h"
#include "../build/ns3/log.h"
#include "../build/ns3/mobility-helper.h"
#include "../build/ns3/net-device-container.h"
#include "../build/ns3/node-container.h"
#include "../build/ns3/nqos-wifi-mac-helper.h"
#include "../build/ns3/nstime.h"
#include "../build/ns3/on-off-helper.h"
#include "../build/ns3/ptr.h"
#include "../build/ns3/rectangle.h"
#include "../build/ns3/simulator.h"
#include "../build/ns3/ssid.h"
#include "../build/ns3/string.h"
#include "../build/ns3/trace-helper.h"
#include "../build/ns3/uinteger.h"
#include "../build/ns3/wifi-helper.h"
#include "../build/ns3/wifi-phy-standard.h"
#include "../build/ns3/yans-wifi-helper.h"

using namespace ns3;

void simulator(uint32_t nWifis, uint32_t nStas, bool writeMobility, bool debug,
DataRate dataRate, uint32_t packetSize){
        uint32_t totalNNodes =  nWifis * nStas;
        uint32_t totalNFlows = totalNNodes;

        NodeContainer backboneNodes;
        NetDeviceContainer backboneDevices;
        Ipv4InterfaceContainer backboneInterfaces;
        std::vector<NodeContainer> staNodes;
        std::vector<NetDeviceContainer> staDevices;
        std::vector<NetDeviceContainer> apDevices;
        std::vector<Ipv4InterfaceContainer> staInterfaces;
        std::vector<Ipv4InterfaceContainer> apInterfaces;

        InternetStackHelper stack;
        CsmaHelper csma;
        Ipv4AddressHelper ip;
        ip.SetBase ("192.168.0.0", "255.255.255.0");

        backboneNodes.Create (nWifis);
        stack.Install (backboneNodes);

        backboneDevices = csma.Install (backboneNodes);

        YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();
        wifiPhy.SetPcapDataLinkType (YansWifiPhyHelper::DLT_IEEE802_11_RADIO);

        for (uint32_t i = 0; i < nWifis; ++i){
                // calculate ssid for wifi subnetwork
                std::ostringstream oss;
                oss << "wifi-default-" << i;
                Ssid ssid = Ssid (oss.str ());

                NodeContainer sta;
                NetDeviceContainer staDev;
                NetDeviceContainer apDev;
                Ipv4InterfaceContainer staInterface;
                Ipv4InterfaceContainer apInterface;
                MobilityHelper mobility;
                BridgeHelper bridge;
                WifiHelper wifi = WifiHelper::Default ();
                wifi.SetStandard (WIFI_PHY_STANDARD_80211b);

                NqosWifiMacHelper wifiMac = NqosWifiMacHelper::Default ();

                YansWifiChannelHelper wifiChannel = YansWifiChannelHelper::Default ();
                wifiChannel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");
                wifiPhy.SetChannel (wifiChannel.Create ());

                sta.Create (nStas);
                mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
                "MinX", DoubleValue (0.0),
                "MinY", DoubleValue (0.0),
                "DeltaX", DoubleValue (5.0),
                "DeltaY", DoubleValue (5.0),
                "GridWidth", UintegerValue (1),
                "LayoutType", StringValue ("RowFirst"));

                // setup the AP.
                mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
                mobility.Install (backboneNodes.Get (i));
                wifiMac.SetType ("ns3::ApWifiMac",
                "Ssid", SsidValue (ssid));
                apDev = wifi.Install (wifiPhy, wifiMac, backboneNodes.Get (i));

                NetDeviceContainer bridgeDev;
                bridgeDev = bridge.Install (backboneNodes.Get (i), NetDeviceContainer (apDev, backboneDevices.Get (i)));

                apInterface = ip.Assign (bridgeDev);

                stack.Install (sta);

                mobility.SetMobilityModel ("ns3::RandomDirection2dMobilityModel",
                "Bounds", RectangleValue (Rectangle (-20, 20, -20, 20)),
                "Speed", StringValue ("ns3::ConstantRandomVariable[Constant=4]"),
                "Pause", StringValue ("ns3::ConstantRandomVariable[Constant=1]"));

                mobility.Install (sta);

                wifiMac.SetType ("ns3::StaWifiMac",
                "Ssid", SsidValue (ssid),
                "ActiveProbing", BooleanValue (false));
                staDev = wifi.Install (wifiPhy, wifiMac, sta);
                staInterface = ip.Assign (staDev);

                staNodes.push_back (sta);
                apDevices.push_back (apDev);
```

```
                apInterfaces.push_back (apInterface);
                staDevices.push_back (staDev);
                staInterfaces.push_back (staInterface);

                Ipv4Address addr;
                uint32_t nStaNodes = sta.GetN();
                if(debug){
                        for(uint32_t j = 0 ; j < nStaNodes; j++) {
                                addr = staInterface.GetAddress(j);
                                std::cout << "_Node_" << i << "." << j << "\t_"<< "IP_Address_"
                                <<addr << std::endl;
                        }
                }
        }

        if(debug){
                Ipv4Address addr;
                std::cout << "wifi_ap_node_addresses.." << "\n";
                uint32_t nApNodes = backboneNodes.GetN();
                for(uint32_t i = 0 ; i < nApNodes; i++) {
                        addr = apInterfaces[i].GetAddress(0);
                        std::cout << "_Node_" << i << "\t_"<< "IP_Address
_____________________"<< addr << std::endl;
                }
        }

        Address dest;
        std::string protocol;

        dest = InetSocketAddress (apInterfaces[1].GetAddress (0), 1025);
        protocol = "ns3::UdpSocketFactory";

        OnOffHelper onoff = OnOffHelper (protocol, dest);
        onoff.SetConstantRate (DataRate (dataRate), packetSize);

        for(uint32_t i = 0; i< nWifis; i++){
        ApplicationContainer apps = onoff.Install (staNodes[i]);
        apps.Start (Seconds (3.0));
        apps.Stop (Seconds (5));
        }

        //Generate the Pcap files to look into the traffic using wireshark
        //wifiPhy.EnablePcap ("wifi-wired-bridging", apDevices[0]);
        //wifiPhy.EnablePcap ("wifi-wired-bridging", apDevices[1]);

        FlowMonitorHelper flowmon;
        Ptr<FlowMonitor> monitor = flowmon.InstallAll();

        if (writeMobility)
        {
        AsciiTraceHelper ascii;
        MobilityHelper::EnableAsciiAll (ascii.CreateFileStream ("wifi-wired-bridging.mob"));
        }

        Simulator::Stop (Seconds (6.0));
        Simulator::Run ();

        // credits to Rizqi Hersyandika for Flow Monitor example.
        monitor->CheckForLostPackets ();
        Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier> (flowmon.GetClassifier ());
        std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats ();

        float throughput[totalNFlows];

        for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator i = stats.begin (); i != stats.end (); ++i) {
                Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (i->first);
                if(debug){
                        std::cout << "Flow_" << i->first << "_(" << t.sourceAddress << "_->_" << t.destinationAddress << ")\n";
                        std::cout << "_Tx_Bytes:_" << i->second.txBytes << "\n";
                        std::cout << "_Rx_Bytes:_" << i->second.rxBytes << "\n";
                        std::cout << "_Throughput:_" << i->second.rxBytes * 8.0 / (i->second.timeLastRxPacket.GetSeconds() - i->second.timeFirstTxPacket.GetSeconds())/1024 << "_kbps\n";
                        std::cout << "_Delay_:_" << i->second.delaySum / i->second.rxPackets << "\n";
                }
                throughput[i->first] = i->second.rxBytes * 8.0 / (i->second.timeLastRxPacket.GetSeconds() - i->second.timeFirstTxPacket.GetSeconds())/1024;
        }
        Simulator::Destroy ();

        float totalThroughPut = 0;
        for (uint32_t i = 1; i <= (totalNFlows); i++){
                totalThroughPut += throughput[i];
        }
        if(debug){
                std::cout << "average_throughput_of_system_=_" << "**" << totalThroughPut / totalNFlows << "**" <<  "kbps" <<"\n";
        }
        std::cout <<  nWifis << "_" << nStas*nWifis << "_" << nStas << "_" <<  totalThroughPut / totalNFlows << "_" << dataRate << "_" << packetSize << "\n";
}

int main (int argc, char *argv[])
{

        Time::SetResolution (Time::NS);
        LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
        LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);

        uint32_t nWifis = 2;
        uint32_t nStas = 2;
        bool debug = false;
        bool writeMobility = false;
        DataRate dataRate = 5000*1000;
        uint32_t packetSize = 512;

        CommandLine cmd;
        cmd.AddValue ("nWifis", "Number_of_wifi_networks", nWifis);
        cmd.AddValue ("nStas", "Number_of_stations_per_wifi_network", nStas);
        cmd.AddValue ("debug", "print_debug_info_or_use_settings_to_print_to_csv_file", debug);
        cmd.AddValue ("writeMobility", "Write_mobility_trace", writeMobility);
        cmd.AddValue ("dataRate", "set_the_dataRate_in_bps", dataRate);
        cmd.AddValue ("packetSize", "set_the_packetsize_in_byte", packetSize);
        cmd.Parse (argc, argv);
        if(debug){
                simulator(nWifis, nStas, writeMobility, debug, dataRate, packetSize);
        } else {
```

```
            //run simulations for the througput vs. node per access point, for different datarates
            //mind that there are as many flows as there are stations
            std::cout <<  "througput_vs_node/AP"<< "\n";
            std::cout <<  "nWifis" << "␣" << "nStas*nWifis" << "␣" << "nStas" << "␣" <<  "averageThroughput" << "␣" << "dataRate" <<"packetsize"<< "\n";
            simulator(30, 1, writeMobility, debug, 11000*1000, packetSize);

            simulator(15, 2, writeMobility, debug, 11000*1000, packetSize);
            simulator(10, 3, writeMobility, debug, 11000*1000, packetSize);
            simulator(6, 5,  writeMobility, debug, 11000*1000, packetSize);
            simulator(5, 6,  writeMobility, debug, 11000*1000, packetSize);
            simulator(3, 10,  writeMobility, debug, 11000*1000, packetSize);
            simulator(2, 15,  writeMobility, debug, 11000*1000, packetSize);

            simulator(30, 1, writeMobility, debug, 5500*1000, packetSize);
            simulator(15, 2, writeMobility, debug, 5500*1000, packetSize);
            simulator(10, 3, writeMobility, debug, 5500*1000, packetSize);
            simulator(6, 5,  writeMobility, debug, 5500*1000, packetSize);
            simulator(5, 6,  writeMobility, debug, 5500*1000, packetSize);
            simulator(3, 10,  writeMobility, debug, 5500*1000, packetSize);
            simulator(2, 15,  writeMobility, debug, 5500*1000, packetSize);

            simulator(30, 1, writeMobility, debug, 2000*1000, packetSize);
            simulator(15, 2, writeMobility, debug, 2000*1000, packetSize);
            simulator(10, 3, writeMobility, debug, 2000*1000, packetSize);
            simulator(6, 5,  writeMobility, debug, 2000*1000, packetSize);
            simulator(5, 6,  writeMobility, debug, 2000*1000, packetSize);
            simulator(3, 10,  writeMobility, debug, 2000*1000, packetSize);
            simulator(2, 15,  writeMobility, debug, 2000*1000, packetSize);

            simulator(30, 1, writeMobility, debug, 1000*1000, packetSize);
            simulator(15, 2, writeMobility, debug, 1000*1000, packetSize);
            simulator(10, 3, writeMobility, debug, 1000*1000, packetSize);
            simulator(6, 5,  writeMobility, debug, 1000*1000, packetSize);
            simulator(5, 6,  writeMobility, debug, 1000*1000, packetSize);
            simulator(3, 10,  writeMobility, debug, 1000*1000, packetSize);
            simulator(2, 15,  writeMobility, debug, 1000*1000, packetSize);

            //run the simulation with different packet sizes,
            std::cout <<  "payload_vs_node/AP"<< "\n";
            std::cout <<  "nWifis" << "␣" << "nStas*nWifis" << "␣" << "nStas" << "␣" <<  "averageThroughput" << "␣" << "dataRate" << "packetsize"<<"\n";
            simulator(30, 1, writeMobility, debug, 11000*1000, 1472);
            simulator(15, 2, writeMobility, debug, 11000*1000, 1472);
            simulator(10, 3, writeMobility, debug, 11000*1000, 1472);
            simulator(6, 5,  writeMobility, debug, 11000*1000, 1472);
            simulator(5, 6,  writeMobility, debug, 11000*1000, 1472);
            simulator(3, 10,  writeMobility, debug, 11000*1000, 1472);
            simulator(2, 15,  writeMobility, debug, 11000*1000, 1472);

            simulator(30, 1, writeMobility, debug, 11000*1000, 1000);
            simulator(15, 2, writeMobility, debug, 11000*1000, 1000);
            simulator(10, 3, writeMobility, debug, 11000*1000, 1000);
            simulator(6, 5,  writeMobility, debug, 11000*1000, 1000);
            simulator(5, 6,  writeMobility, debug, 11000*1000, 1000);
            simulator(3, 10,  writeMobility, debug, 11000*1000, 1000);
            simulator(2, 15,  writeMobility, debug, 11000*1000, 1000);

            simulator(30, 1, writeMobility, debug, 11000*1000, 500);
            simulator(15, 2, writeMobility, debug, 11000*1000, 500);
            simulator(10, 3, writeMobility, debug, 11000*1000, 500);
            simulator(6, 5,  writeMobility, debug, 11000*1000, 500);
            simulator(5, 6,  writeMobility, debug, 11000*1000, 500);
            simulator(3, 10,  writeMobility, debug, 11000*1000, 500);
            simulator(2, 15,  writeMobility, debug, 11000*1000, 500);
        }
}
```

Bibliography

[1] title. URL http://www.radio-electronics.com/info/wireless/wi-fi/ieee-802-11b.php.

[2] Ns-3 csma block description. URL https://www.nsnam.org/docs/release/3.13/models/html/csma.html.