ET4394 Wireless Networking 2015-16
NS3 Assignment

Frits Kastelein, 4044533

April 30, 2016

Github `https://github.com/fpk47/NS3_fkastelen_4044533`

Contents

1 Introduction

1.1 IEEE 802.11b

`IEEE 802.11b` is an extension to 802.11 specification developed by the IEEE for wireless LAN (WLAN) technology that applies to wireless LANS and provides $11Mbps$ transmission (with a fallback to 5.5, 2 and $1Mbps$) in the $2.4GHz$ band. 802.11b uses only `DSSS`. `DSSS`, Direct Sequence Spread Spectrum, is a technique where the send data is combined with a pseudo random high rate noise signal to spread it over the channel. Originally designed to make it harder to jam frequency bands it also allows for multiple users using the channel at the same time. IEEE 802.11b was a 1999 ratification to the original 802.11 standard, allowing wireless functionality comparable to Ethernet. IEEE 802.11b uses `CSMA/CA`, which is a commonly used method to prevent collisions on a wireless network. The maximum packet size of `UDP` is 1472 bytes. I[1][2]

1.2 NS3

NS3 is a network simulator that is capable of simulating all kind of networks. Simulations can be written in either C++ or Python and it is free and open-source. This assignment will be making use of NS3 and some python scripts.

Outline Section 2, *Project Description* will describe in short what is expected from this assignment. Section 3, *Approach* is going to lay-out all the parameters that are going to be experimented with. In section 4, *Hypothesis* a hypothesis for all the influence of the different parameters is stated. In the next section 5, *Script* the script is being explained and if needed more information is given. In section 6, *Results* the results are presented and discussed. Finally section 7, *Conclusion* gives an conclusion and finishes this assignment.

2 Project Description

Using NS-3 test `IEEE 802.11b` performance against the number of stations. Setup WiFi nodes in an infrastructure mode with one Access Point and experiment with parameters that NS3 provides.

3 Approach

3.1 Scenario

A sensor network is being read-out by a single access point (AP). The AP send some data to every station (STA) and all the STAs will send sensor data tot the AP. In this assignment a UDP echo server is being used, so this means that you have to make the assumptions that the packet size is the same in both directions (which is usually not the case).

3.2 Basic Setup

You have an 2D space with where at (0,0) an AP is located. The 2D space is bounded by $\min_x$, $\min_y$, $\max_x$ and $\max_y$ with

$$\min_x < 0 \ \textit{and} \ \min_y < 0 \ \textit{and} \ \max_x > 0 \ \textit{and} \ \max_y > 0$$

Initially all the STAs are located in a perfect circle around origin with a radius

$$r \leq \min(-\min_x, -\min_y, \max_x, \max_y)$$

and $-\min_x = -\min_y = \max_x = \max_y$.

The AP sends a message to every connected STA and if an STA receives a message it sends the message back to the AP, i.e., echoClients and echoServer are being used. The send interval to every STA is the same and the system is setup that expected throughput of all the nodes adds up to the data rate of the channel, i.e., $1Mbps$, $2Mbps$, $5.5Mbps$ or $11Mbps$. Note that we are not interested in QoS, only in the throughput. See section 5, *Script* for more information.

3.3 Parameters

The throughput is analysed by changing the following parameters with 1 to 10 nodes. See section 5, *Script* for more information on the parameters.

- Data rate (total throughput)
  11 $Mbps$, 5.5 $Mbps$, 2 $Mbps$ and 1 $Mbps$. initialRadius = $50m$, movingNodes = false and packetSize = 1024 bytes for all tests.
- Data rate (per node throughput)
  11 $Mbps$, 5.5 $Mbps$, 2 $Mbps$ and 1 $Mbps$. initialRadius = $50m$, movingNodes = false and packetSize = 1024 bytes for all tests.
- Packetsize (total throughput)
  500 bytes, *1024 bytes* and *1472 bytes*. initialRadius = $50m$, movingNodes = false and dataRate = $11Mbps$ for all. tests
- Packetsize (per node throughput)
  500 bytes, *1024 bytes* and *1472 bytes*. initialRadius = $50m$, movingNodes = false and dataRate = $11Mbps$ for all. tests
- Standing STAs with initial radius (total throughput)
  $60m$, $65m$, $70m$ and $75m$. packetSize = 1024 bytes, dataRate = $11Mbps$ for all tests.
- Moving STAs with initial radius (total throughput)
  $60m$, $65m$, $70m$ and $75m$. packetSize = 1024 bytes, dataRate = $11Mbps$ for all tests.

4 Hypothesis

When increasing the packet size, the percentage of header data will decrease therefore increasing the the throughput. If a faster data rate is chosen a higher throughput can be achieved, the scaling (when you increase the number of nodes, the throughput goes down) will be the same for all the rates. APs that are moving will result in a lower throughput.

5 Script

Github `https://github.com/fpk47/NS3_fkastelen_4044533`

How to Run See the files in the Github link above. In order to properly run this NS3 script you need to place the `fkastelen_script.c` NS3 script in the `scratch` folder and copy folder `fkastelein` to the main NS3 folder where `waf` is located. Now go in your terminal application and go to the `fkastelein` folder and you can run the python script use the command `python x.py` where x is the name of the python script you want to run. The results of the simulations are already provided, it is possible to delete all the .txt files and start over. `run_DataRate.py`, `run_PacketSize.py` and `run_Distance.py` will all recreate the .txt files. All this is also explained in `README.txt`

5.1 NS3 Script

See the attached script at the end of this assignment. The script is an adapted version of `third.c` located in the folder `tutorial`. `third.c` is also available in the `fkastelein` folder. The simulation runs for 5 seconds. Below some key elements are explained. Basic NS3 facts are not covered here and can be found on the NS3 website. [4]

Parameters There are a number of arguments that you can pass, all are explained in table 1 and the default values are given

Parameter	Values	Default
nWifi	The number of *STA*s [1-10]	1
dataRateSetting	$1 = 1Mbps$, $2 = 2Mbps$, $5.5 = 5.5Mbps$ and $11 = 11Mbps$	11
fileName	fileName to append to	Temp.txt
packetSize	packet size of the UDP message [0-1472] bytes	1024
mode	1 = write total throughput and 2 = write per node throughput	1
showNodePositions	true = log AP positions, false = do not log AP positions	true
movingNodes	true = RandomWalk2dMobilityModel, false = ConstantPositionMobilityModel	false
initialRadius	true = initial radius of all the APs from the origin [0-79] m	50

Table 1: List of parameters used for `fkastelen_script.c`

Most of the parameters are bounded, i.e., when passing a wrong value the program will terminate and log an error message. Note that the maximum allowed value for the initialRadius is 79, using 80 the *STA*s are not able to receive or send anything to and from the *AP*.

Initial placing of *AP*s As section 3.2, *Basic Setup* explained are all the nodes initially equally spread in a circle around the origin. The formula used for the placement of each AP is:

$$AP_x = \cos(\alpha) \cdot initialRadius$$
$$AP_y = \sin(\alpha) \cdot initialRadius$$

with $\alpha = 2\pi \cdot \frac{i}{N}$ if there are N *STA*s for $0 \geq i < N$

Mobility Model There are 2 models used: `ConstantPositionMobilityModel` and `RandomWalk2dMobilityModel`. In this script `RandomWalk2dMobilityModel` is configured such that it walks in a random direction for $\frac{\text{initialRadius}}{5.0} m$ and then repeat it's process by walking again in a random direction. The space it can move in is bounded by maxRadius. `ConstantPositionMobilityModel` is a static model, i.e., the APs do not move during the simulation. When you see *standing* `ConstantPositionMobilityModel` is used and when talking about *moving* `RandomWalk2dMobilityModel` is used. `RandomWalk2dMobilityModel` will yield the same result every time you run the simulation.

FlowMonitor he NS3 FlowMonitor class is used to calculate the throughput and since we are interested in the real throughput of the system, the header size of the UDP message is not taking into account (see the -28 in the script). Because the data rate is given in $Mbps$ the throughput is calculated in $kbps$ and not $Kibps$. Note the division by 1000 and not 1024. [3]

5.2 Python Scripts

There are multiple python scripts, below each one is explained. Node that to be able to use the same terminal again you first need to close the generated figure.

- `run_DataRate.py`
  First clears and then saves results to `dataRatePerNode.txt` and `dataRateTotal.txt`. Runs the data rate experiments explained in section 3, *Approach* for $1Mbps$, $2Mbps$, $5.5Mbps$ and $11Mbps$ for total throughput and per node throughput.
- `run_PacketSize.py`
  First clears and then saves results to `packetSizePerNode.txt` and `packetSizeTotal.txt`. Runs the packet size experiments explained in section 3, *Approach* with packet sizes of 500, 1024 and 1472 bytes for total throughput and per node throughput.
- `run_Distance.py`
  First clears and then saves results to `distanceMovingTotal.txt` and `distanceStandingTotal.txt`. Runs the distance experiments explained in section 3, *Approach* with moving and standing *STA*s at varying distances.
- `plotData_DataRate_Total.py`
  Plots the data rate vs **total** throughput graph.
- `plotData_DataRate_PerNode.py`
  Plots the data rate vs **per node** throughput graph.
- `plotData_PacketSize_Total.py`
  Plots the packet size vs **total** throughput graph.
- `plotData_PacketSize_PerNode.py`
  Plots the packet size vs **per node** throughput graph.
- `plotData_Distance_Standing.py`
  Plots the distance vs total throughput graph with **standing** *STA*s.
- `plotData_Distance_Moving.py`
  Plots the distance vs total throughput graph with **moving** *STA*s.

6 Results

6.1 Data rate vs throughput

As can been seen in figure 1 the throughput stabilises after around 5 *STA*s. For less then 5 *STA*s the data rates $5.5Mbps$ and $11Mbps$ do a better job in keeping the throughput high. Again note that expected throughput, i.e., the data rate of the send data by the *AP* is the same as the data rate of the channel and the *AP* sends every *STA* an even amount of data. The peak at $nWifi = 3$, $dataRateSetting = 2$ is unexpected, you would expect the thoughput to go down when dealing with more *STA*s to send to. The ratio of the difference between the throughput at $nWifi = 10$ and at $nWifi = 1$ is smaller with $dataRateSetting = 1$ then with $dataRateSetting = 1$. A possible explanation could be that there are more packets send when the data rate is higher, therefore increasing the chance of collisions.

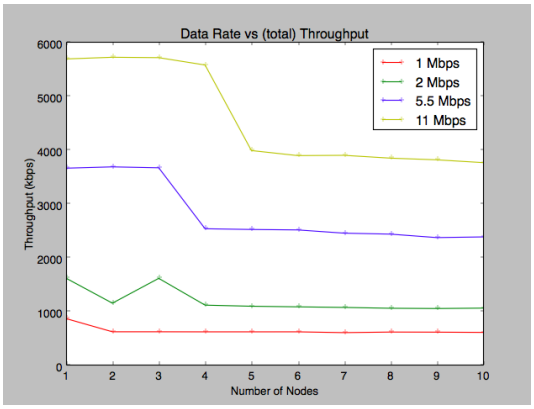

Figure 1: Data rate vs (total) throughput

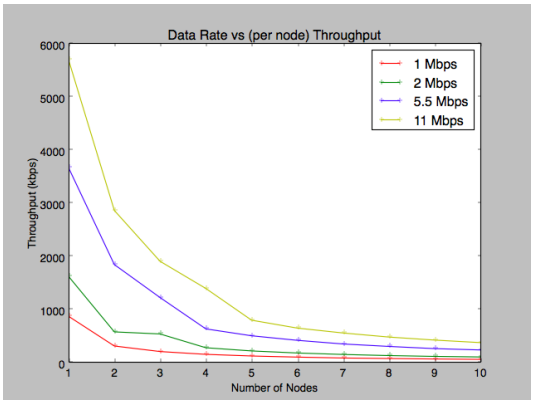

Figure 2: Data rate vs (per node) throughput

6.2 Packet size vs throughput

As shown in figure 3, the throughput decreases as the number of *STA*s increases. For this experiment $dataRateSetting = 11$ is used. The ratio $\frac{\text{throughput at nWifi} = 1}{\text{throughput at nWifi} = 10}$ is the same for all the data rates. Choosing a larger packet size will result (in almost all the cases) in a better throughput. This is because $\frac{\text{data size}}{\text{data size+header size}}$ increases which leads to more data send per frame.

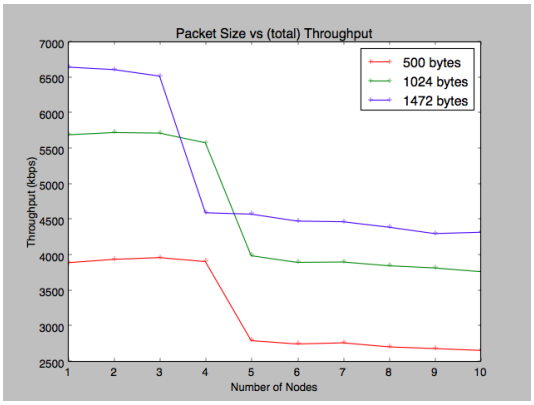

Figure 3: Packet size vs (total) throughput

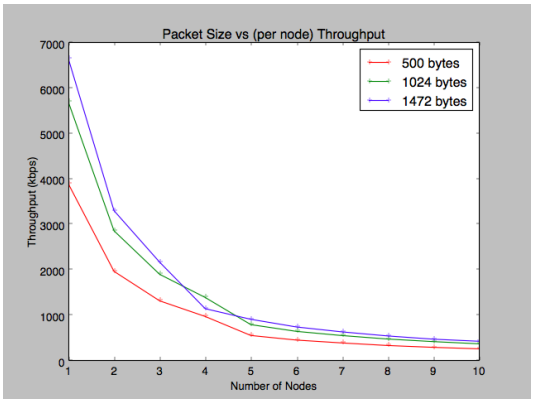

Figure 4: Packet size vs (per node) throughput

6.3 Distance Standing / Moving vs total throughput

See figure 5: as the distance increases the throughput decreases. Note that the throughput increases for $distance = 75$ when the number of *STA*s increases. A cause could be the low chance of a successful

transmission.

If you look at figure 6 and compare it to figure 5 it is clear that moving randomly in this scenario has more advantages when you start closer to the border. There is basically only one way you could go to. When the initial starting point is further away from the center the effect is less visible and the throughput can be either better of worse.

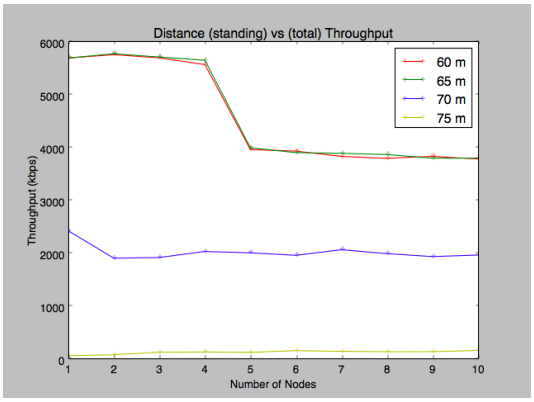

Figure 5: Distance (standing) vs (total) throughput

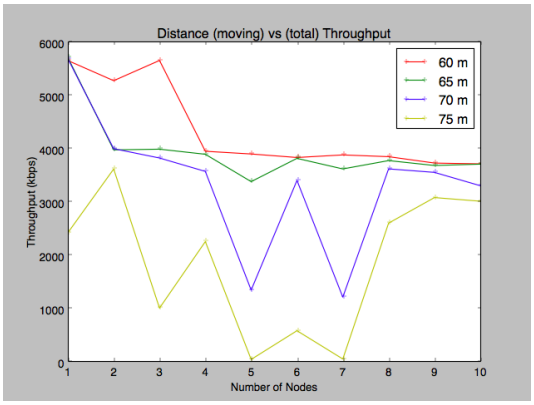

Figure 6: Distance (moving) vs (total) throughput

7 Conclusion

See section 4, *Hypothesis*. Moving *STA*s **sometime** result in a lower throughput. Having a higher data rate for your channel increases the throughput. A higher packet size can yield a better throughput. You could conclude that depending on your application, the needed data rate, the number of *AP*s and the position of the *AP*s the throughput will be different every time. Having some general knowledge can help you make a better decision.

8 References

[1] Webopedia. IEEE 802.11b. Available at *http://www.webopedia.com/TERM/8/802_11b.html*, April 2016.

[2] Techopedia. Direct Sequence Spread Spectrum (DSSS). Available at *https://www.techopedia.com/definition/14804/direct-sequence-spread-spectrum-dsss*, April 2016.

[3] Department of Computer Engineering Faculty of Engineering, Kasetsart University. NS3: Flow Monitor Module. Available at *https://www.cpe.ku.ac.th/~anan/myhomepage/wp-content/uploads/2013/06/ns3-part4-wireless_modified_part2.pdf*, September 2013.

[4] NS3. NS3 tutorial. Available at *https://www.nsnam.org/docs/release/3.18/tutorial/ns-3-tutorial.pdf*, November 2013.

```
1  /* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
2
3  #include "ns3/core-module.h"
4  #include "ns3/network-module.h"
5  #include "ns3/applications-module.h"
6  #include "ns3/wifi-module.h"
7  #include "ns3/mobility-module.h"
8  #include "ns3/csma-module.h"
9  #include "ns3/internet-module.h"
10 #include "ns3/flow-monitor-module.h"
11
12 // Default Network Topology
13 //
14 // Number of wifi nodes can be increased up to 250
15 //                          |
16 // -------------------------
17 //   Wifi 10.1.1.0
18 //                 AP
19 //  *    *    *    *
20 //  |    |    |    |
21 // n5   n6   n7   n0
22 //
23
24
25 using namespace ns3;
26
27 NS_LOG_COMPONENT_DEFINE ("fkastelein");
28
29 int main (int argc, char *argv[])
30 {
31     bool movingNodes = false;
32     bool showNodePositions = false;
33     uint32_t nWifi = 1;
34     uint32_t packetSize = 1024;
35     uint32_t mode = 1;
36     double initialRadius = 40;
37     double maxRadius = 79;
38
39     // Default file to write
40     char fileName[100];
41     sprintf( fileName, "Temp.txt" );
42
43     // Default Datarate
44     double dataRateSetting = 11;
45     StringValue dataRate = StringValue("DsssRate11Mbps");
46
47     // Read command line arguments
48     CommandLine cmd;
49     cmd.AddValue ("nWifi", "Number of wifi STA devices", nWifi);
50     cmd.AddValue ("dataRateSetting", "1 = 1 Mbps, 2 = 2 Mbps, 5.5 = 5.5M bps, 11 = 11
           Mbps", dataRateSetting);
51     cmd.AddValue ("fileName", "File to write result to", fileName);
52     cmd.AddValue ("packetSize", "packetSize of the UDP message", packetSize);
53     cmd.AddValue ("mode", "1 = total througput, 2 = per node througput", mode);
54     cmd.AddValue ("showNodePositions", "show the (initially) node positions",
           showNodePositions);
55     cmd.AddValue ("movingNodes", "true = RandomWalk2dMobilityModel, false =
           ConstantPositionMobilityModel", movingNodes);
56     cmd.AddValue ("initialRadius", "initial radius of all the APs from the origin",
           initialRadius );
57     cmd.Parse (argc,argv);
58
59     std::cout << "/* Start Initialising Data */" << "\n" << std::endl;
60
61     // Set Datarate
62     if ( dataRateSetting == 1 )
63     {
64         std::cout << "Data rate     = DsssRate1Mbps" << std::endl;
65         dataRate = StringValue("DsssRate1Mbps");
66     }
67     else if ( dataRateSetting == 2 )
```

```
68      {
69          std::cout << "Data rate     = DsssRate2Mbps" << std::endl;
70          dataRate = StringValue("DsssRate2Mbps");
71      }
72      else if ( dataRateSetting == 5.5 )
73      {
74          std::cout << "Data rate     = DsssRate5.5Mbps" << std::endl;
75          dataRate = StringValue("DsssRate5_5Mbps");
76      }
77      else if ( dataRateSetting == 11 )
78      {
79          std::cout << "Data rate     = DsssRate11Mbps" << std::endl;
80          dataRate = StringValue("DsssRate11Mbps");
81      }
82
83      // Print fileName to write to
84
85      if ( packetSize <= 1472 )
86      {
87          std::cout << "packetSize    = " << packetSize << " bytes" << std::endl;
88      }
89      else
90      {
91          std::cout << "packetSize    = " << "ERROR, WRONG VALUE: not [0-1472]" << std::endl;
92          return 0;
93      }
94
95      if ( nWifi )
96      {
97           std::cout << "nWifi         = " << nWifi << std::endl;
98      }
99      else
100     {
101         std::cout << "nWifi          = " << "ERROR, WRONG VALUE: not [1-10]" << std::endl;
102         return 0;
103     }
104
105     std::cout << "fileName      = " << fileName << std::endl;
106     std::cout << "movingNodes   = " << movingNodes << std::endl;
107
108
109     if ( initialRadius >= 0 && initialRadius <= maxRadius )
110     {
111         std::cout << "initialRadius = " << initialRadius << std::endl;
112     }
113     else
114     {
115         std::cout << "initialRadius = " << "ERROR, WRONG VALUE: not [0-79]" << std::endl;
116         return 0;
117     }
118
119     // Write mode
120     if ( mode == 1 )
121         std::cout << "mode          = " << "writing (total) throughput" << std::endl;
122     else if (mode == 2 )
123         std::cout << "mode          = " << "writing (per node) throughput" << std::endl;
124     else
125         std::cout << "mode          = " << "ERROR, WRONG VALUE" << std::endl;
126
127     // Check for valid number of csma or wifi nodes
128     if (nWifi > 250 )
129     {
130         std::cout << "Too many wifi nodes, no more than 250 each." << std::endl;
131         return 1;
132     }
133
134     std::cout << "\n" << "/* End Initialising Data */" << std::endl;
135
136     // Create STA and AP node(s)
137     NodeContainer staNodes;
138     NodeContainer apNode;
139     staNodes.Create(nWifi);
```

```
140     apNode.Create(1);
141
142     // Create WiFi channel and link physical and channel
143     YansWifiChannelHelper channel = YansWifiChannelHelper::Default ();
144     YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();
145     phy.SetChannel (channel.Create ());
146
147     WifiHelper wifi = WifiHelper::Default ();
148     wifi.SetStandard (WIFI_PHY_STANDARD_80211b);
149     wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager","DataMode", dataRate,
            "ControlMode", dataRate);
150
151     NqosWifiMacHelper mac = NqosWifiMacHelper::Default ();
152
153     // Install devices on the network
154     Ssid ssid = Ssid ("ns-3-ssid");
155     mac.SetType ("ns3::StaWifiMac",
156                  "Ssid", SsidValue (ssid),
157                  "ActiveProbing", BooleanValue (false));
158
159     NetDeviceContainer staDevices;
160     staDevices = wifi.Install (phy, mac, staNodes);
161
162     mac.SetType ("ns3::ApWifiMac",
163                  "Ssid", SsidValue (ssid));
164
165     NetDeviceContainer apDevice;
166     apDevice = wifi.Install (phy, mac, apNode);
167
168     // Set the mobility and position of the devices
169     MobilityHelper mobility;
170
171     Ptr<ListPositionAllocator> initialAlloc = CreateObject<ListPositionAllocator>();
172     double currentAngle = 0.0;
173     const double pi = 3.1415926535897;
174
175     if ( showNodePositions )
176         std::cout << "/* Start AP Nodes Positions */\n" << std::endl;
177
178     // Create N points evenly spaced around the origin with the same radius
179     for (uint32_t i = 0; i < staNodes.GetN(); i++ )
180     {
181         initialAlloc->Add (Vector (i, cos(currentAngle)*initialRadius, sin(currentAngle)*
                initialRadius ));
182
183         if ( showNodePositions )
184             std::cout << "AP " << i + 1 << ": ( " << cos(currentAngle)*initialRadius << ",
                    " << sin(currentAngle)* initialRadius << " )" << std::endl;
185
186         currentAngle += 2.0*pi/ (double) nWifi;
187     }
188
189     if ( showNodePositions )
190         std::cout << "\n/* Start AP Nodes Positions */" << std::endl;
191
192     initialAlloc->Add( Vector ( staNodes.GetN(), 0.0, 0.0 ) );
193
194     mobility.SetPositionAllocator(initialAlloc);
195
196     if ( movingNodes )
197     {
198         mobility.SetMobilityModel ("ns3::RandomWalk2dMobilityModel",
199                                    "Distance", ns3::DoubleValue (initialRadius/5.0),
200                                    "Bounds", RectangleValue (Rectangle (-maxRadius,
                                        maxRadius, -maxRadius, maxRadius)));
201     }
202     else
203         mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
204
205     mobility.Install (staNodes);
206
207     mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
```

```
208     mobility.Install (apNode);
209
210     InternetStackHelper stack;
211     stack.Install (staNodes);
212     stack.Install (apNode);
213
214     Ipv4AddressHelper address;
215
216     address.SetBase ("10.1.1.0", "255.255.255.0");
217     Ipv4InterfaceContainer apInterface, staInterfaces;
218     staInterfaces = address.Assign (staDevices);
219     apInterface = address.Assign (apDevice);
220
221     UdpEchoServerHelper echoServer (9);
222     ApplicationContainer serverApps = echoServer.Install (apNode.Get (0));
223
224     serverApps.Start (Seconds (1.0));
225     serverApps.Stop (Seconds (7.0));
226
227     double interval = 1.0 / dataRateSetting / 1000.0 / 1000.0 * (double) packetSize *
            (double) nWifi;
228
229     UdpEchoClientHelper echoClient (apInterface.GetAddress (0), 9);
230     echoClient.SetAttribute ("MaxPackets", UintegerValue (10000000));
231     echoClient.SetAttribute ("Interval", TimeValue (Seconds(interval)));
232     echoClient.SetAttribute ("PacketSize", UintegerValue (packetSize));
233
234     ApplicationContainer clientApps = echoClient.Install (staNodes);
235     clientApps.Start (Seconds (2.0));
236     clientApps.Stop (Seconds (7.0));
237
238     Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
239
240     Simulator::Stop (Seconds (7.0));
241
242     // Init Flowmonitor
243     FlowMonitorHelper flowmon;
244     Ptr<FlowMonitor> monitor = flowmon.InstallAll ();
245
246     // Run the simulation
247     Simulator::Run ();
248
249     // Calculate the Throughput
250     monitor->CheckForLostPackets ();
251     Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier>
            (flowmon.GetClassifier());
252     std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats ();
253
254     uint64_t bytesTotal = 0;
255     double lastRxTime=-1;
256     double firstRxTime=-1;
257
258     for (std::map<FlowId,FlowMonitor::FlowStats>::const_iterator
            i=stats.begin();i!=stats.end(); ++i)
259     {
260         if (firstRxTime < 0)
261         {
262             firstRxTime = i->second.timeFirstRxPacket.GetSeconds();
263         }
264         else if (firstRxTime > i->second.timeFirstRxPacket.GetSeconds() )
265         {
266             firstRxTime = i->second.timeFirstRxPacket.GetSeconds();
267         }
268
269         if (lastRxTime < i->second.timeLastRxPacket.GetSeconds() )
270         {
271             lastRxTime = i->second.timeLastRxPacket.GetSeconds();
272         }
273
274         bytesTotal = bytesTotal + i->second.rxBytes - 28;
275     }
276
```

```
277     double throughput = bytesTotal*8/(lastRxTime-firstRxTime) /1000.0;
278
279     std::cout << "/* Start Presenting results */\n" << std::endl;
280
281     std::cout << "expected (total) throughput    = " << 1.0 / interval * packetSize / 1000
            * nWifi << " Kbps" << std::endl;
282     std::cout << "expected (per node) throughput = " << 1.0 / interval * packetSize / 1000
            << " Kbps" << std::endl;
283     std::cout << "real (total) throughput        = " << throughput << " Kbps" << std::endl;
284     std::cout << "real (per node) throughput     = " << throughput / (double) nWifi << "
            Kbps" << std::endl;
285
286     std::cout << "\n/* End Presenting results */" << std::endl;
287
288     // Write result to file
289     std::ofstream outfile;
290
291     char fileNameFinal[100];
292     sprintf( fileNameFinal, "Scripts_Frits/%s", fileName );
293     outfile.open(fileNameFinal, std::ios_base::app);
294
295     if ( mode == 1 ) // Total througput
296         outfile << throughput << std::endl;
297     else if ( mode == 2 ) // Per Node Througput
298         outfile << throughput / (double) nWifi << std::endl;
299     else
300         std::cout << "ERROR: mode flag set wrong" << std::endl;
301
302     Simulator::Destroy ();
303     return 0;
304 }
```

Script.c