DCU School of Electronic Engineering Assignment Submission

Student Name(s):	**Michael Lenehan**
Student Number(s):	**15410402**
Programme:	**B.Eng in Electronic and Computer Engineering**
Module Code:	**EE452**
Lecturer:	**J. McManis**
Project Due Date:	**16/11/2018**

Declaration

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited andacknowledged within the text of my work. I understand thatplagiarism, collusion, and copying is a grave and serious offence in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion, or copying. I have read and understood the Assignment Regulations set out in the module documentation. I have identified and included the source of all facts, ideas, opinions, viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the source cited are identified in the assignment references.

I have not copied or paraphrased an extract of any length from any source without identifying the source and using quotation marks as appropriate. Any images, audio recordings, video or other materials have likewise been originated and produced by me or are fully acknowledged and identified.

This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study. I have read and understood the referencing guidelines found at http://www.library.dcu.ie/citing&refguide08.pdf and/or recommended in the assignment guidelines.

I understand that I may be required to discuss with the module lecturer/s the contents of this submission.

I/me/my incorporates we/us/our in the case of group work, which is signed by all of us.

Signed: ***Michael Lenehan***

Assignment 1

Michael Lenehan

Introduction

The purpose of this assignment is to investigate, using the ns-3 simulation software package, the UDP protocol for packet transfer. The protocol must be analysed for systems of both single source - single ink, and single source - two sink configurations. The analysis focuses on packet loss, delay, throughput, and jitter. The assignment also allows for the student to become familiar with ns3, and NetAnim, Wireshark, and other analysis software packages.

Procedure

2.1 Single Source - Single Sink

For the completion of the Single Source - Single Sink system, the provided code is required to be modified for use in the most recent version of ns-3, ns-3.29. A number of changes had been implemented between ns-3.25 and ns-3.29. This included changes to the default constructor for the WifiHelper, and WifiMacHelper class.

```
// ns-3.25 WifiHelper Constructor
WifiHelper wifi = WifiHelper::Default();

// ns-3.29 WifiHelper Constructor
WifiHelper wifi = WifiHelper();

// ns-3.25 WifiMacHelper Constructor
WifiMacHelper wifiMac = WifiMacHelper::Default();

// ns-3.29 WifiMacHelper Constructor
WifiMacHelper wifiMac = WifiMacHelper();
```

The RAA used within the provided code, the “AARF” RAA also cannot be used in ns-3.29, due to it not supporting HT rates in this version. As such, the compatible RAA’s

were listed, with the MinstrelHT RAA being chosen for this assignment.

2.2 Single Source - Two Sink

In order to modify the code to implement a Single Source - Two Sink model, a third node must be added. This is done by passing a 3 as the argument to the "nodes.Create()" method. This node must have a constant position set, in this case at position (0, 5, 0). A new sink object must then be created. This sink object represents the new node in the system. A second traffic generator object must also be created, however it must be assigned to the same address as the original access point. This will allow for the access point to transmit to both of the nodes.

```
// Added Node
NodeContainer nodes;
nodes.Create(3);

//Added Position Vector
positionAlloc -> add (Vector (0.0, 5.0, 0.0));

// Added Sink
PacketSinkHelper sinkHelper2 ("ns3::UdpSocketFactory",
                InetSocketAddress (Ipv4Address::GetAny(), 9));
ApplicationContainer sinkApp2 = sinkHelper.Install(nodes.Get(2));
sinkApp2.Start(seconds(2.0));
sinkApp2.Stop(seconds(30.0));

//Added Traffic Generator
OnOffHelper onoff2 ("ns3::UdpSocketFactory",
            Address (InetSocketAddress (interfaces.GetAddress(2), 9)));
onoff2.SetAttribute ("OnTime", StringValue
                ("ns3::ConstantRateVariable[Constant=1.0]"));
onoff2.SetAttribute ("OffTime", StringValue
                ("ns3::ConstantRateVariable[Constant=0.0]"));
onoff2.SetAttribute ("PacketSize", StringValue ("1024"));
onoff2.SetAttribute ("DataRate", StringValue ("1000kb/s"));
ApplicationContainer apps2 = onoff2.Install (nodes.Get(0));
apps2.Start(Seconds(2.0));
apps2.Stop(Seconds(30.0));
```

Results

The following results were obtained by executing the code in the Appendix. The output XML files were then loaded into NetAnim for analysis. The simulation runs for a total of 60 seconds in order to correctly calculated the amount of lost packets within the models.

3.1 Single Source - Single Sink

3.1.1 Loss Calculation

The number of lost packets, as shown in NetAnim, is 0. This gives a packet loss ratio of 0, according to the formula:

$$q = \frac{lostPackets}{rxPackets + lostPackets}$$

3.1.2 Delay

The average delay can be calculated using the *delaySum* and *rxPacket* information from NetAnim. The formula given below shows the calculation of mean delay for the Single Source - Single Sink model.

$$\overline{delay} = \frac{delaySum}{rxPackets}$$

$$\cdots = \frac{5.975\,68}{34179}$$

$$\overline{delay} = 0.174\,835 \times 10^{-3}\,\mathrm{s}$$

$$\therefore \overline{delay} = 0.174\,835\,\mathrm{ms}$$

3.1.3 Throughput

Throughput for the model can be calculated using the *rxBytes*, *timeLastRxPacket*, and *timeFirstRxPacket* information from NetAnim. The formula given below shows the calculation of throughput in bits per second, for the Single Source - Single Sink model.

$$Throughput = \frac{rxBytes \times 8}{timeLastRxPacket - timeFirstRxPacket]}$$

$$\cdots = \frac{35956308 \times 8}{29\,999\,467\,197 \times 10^{-9} - 2\,003\,470\,348 \times 10^{-9}}$$

$$Throughput = 10.2747 \times 10^{6}(bps)$$

$$\therefore Throughput = 10.275 Mbps$$

The following figure is a plot of the throughput of the system.

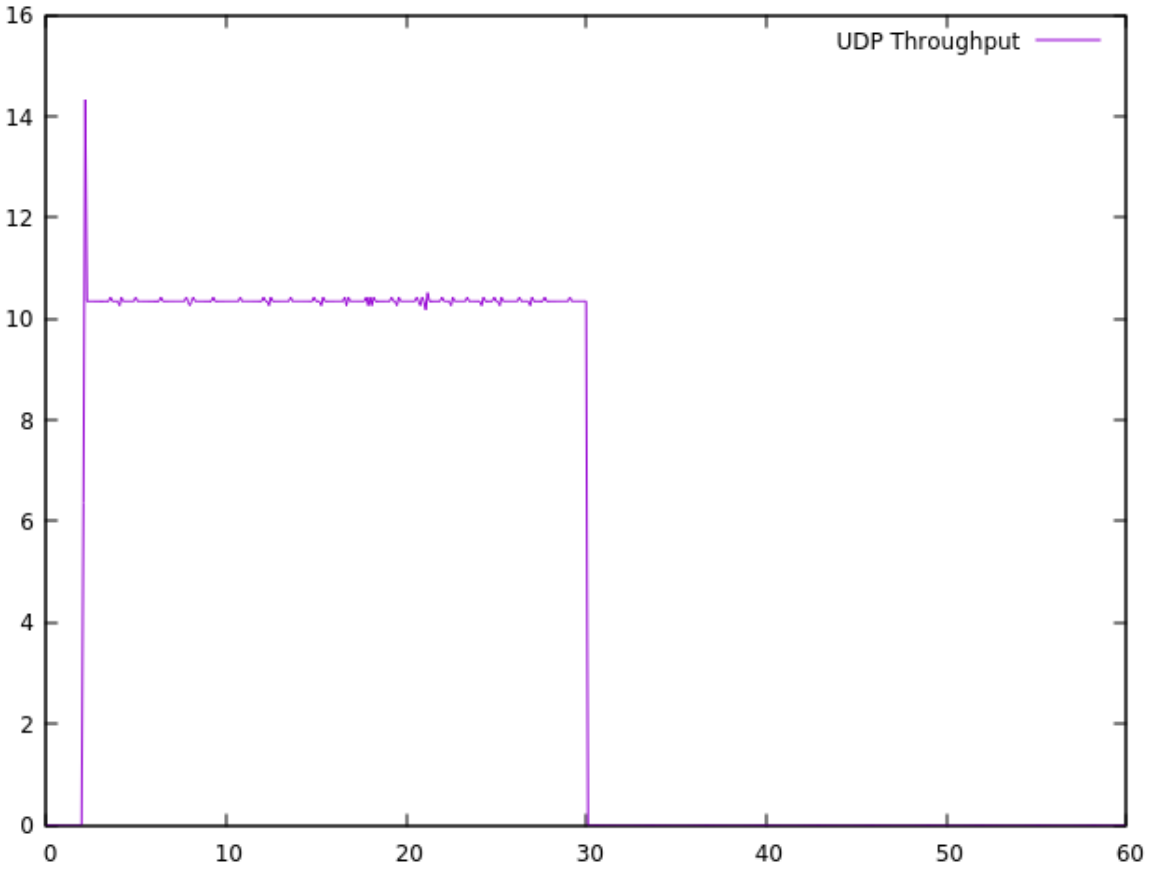

Figure 1: Throughput of the Single Node - Single Access Point Model

3.1.4 Jitter

Average Jitter of the model can be calculated using the *jitterSum* and *rxPackets* information from NetAnim. The formula given below shows the calculation of jitter for the Single Source - Single Sink Model.

$$\overline{jitter} = \frac{jitterSum}{rxPackets - 1}$$

$$\cdots = \frac{2433168191e-9}{34179-9}$$

$$\overline{jitter} = 71.191 \times 10^{-6}$$

3.2 Single Source - Two Sink

3.2.1 Loss Calculation

The number of lost packets for Node 1, as shown in NetAnim, is 3. This gives a packet loss ratio of 0.00877732%. The number of lost packets for Node 2, as shown in NetAnim, is 135. This gives a packet loss ratio of 0.394979%. Packet loss ratios are calculated as follows:

$$q = \frac{lostPackets}{rxPackets+lostPackets}$$

3.2.2 Delay

The average delay is calculated in the same way as above. For Node 1, the average delay is as follows:

$$\overline{delay} = \frac{delaySum}{rxPackets}$$

$$\cdots = \frac{12.933}{34176}$$

$$\overline{delay} = 0.378\,425 \times 10^{-3}\,\mathrm{s}$$

$$\therefore \overline{delay} = 0.378\,425\,\mathrm{ms}$$

For Node 2 the average delay is as follows:

$$\overline{delay} = \frac{delaySum}{rxPackets}$$

$$\cdots = \frac{15.692}{34044}$$

$$\overline{delay} = 0.460\,93 \times 10^{-3}\,\mathrm{s}$$

$$\therefore \overline{delay} = 0.460\,93\,\mathrm{ms}$$

3.2.3 Throughput

$$Throughput = \frac{rxBytes \times 8}{timeLastRxPacket - timeFirstRxPacket}$$

$$\cdots = \frac{35953152 \times 8}{29\,999\,467\,197 \times 10^{-9} - 2\,006\,470\,348 \times 10^{-9}}$$

$$Throughput = 10.2749 \times 10^{6} (bps)$$

$$\therefore Throughput = 10.275 Mbps$$

$$Throughput = \frac{rxBytes \times 8}{timeLastRxPacket - timeFirstRxPacket}$$

$$\cdots = \frac{35814288 \times 8}{29\,999\,627\,229 \times 10^{-9} - 2\,115\,381\,658 \times 10^{-9}}$$

$$Throughput = 10.275\,132 \times 10^{6} (bps)$$

$$\therefore Throughput = 10.275 Mbps$$

The following figure is a plot of the throughput of the system.

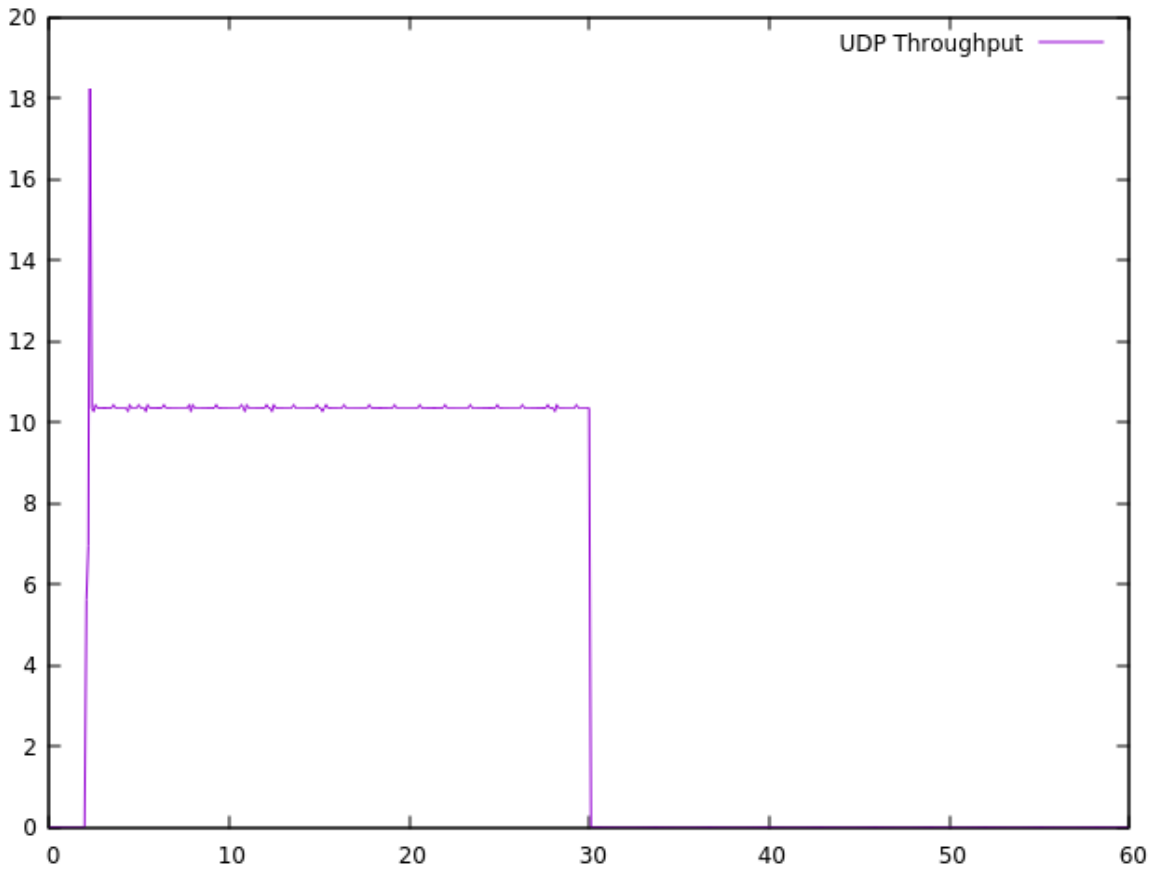

Figure 2: Throughput of the Two Node - Single Access Point Model

3.2.4 Jitter

$$\overline{jitter} = \frac{jitterSum}{rxPackets-1}$$

$$\ldots = \frac{2\,910\,861\,867 \times 10^{-9}}{34176-1}$$

$$\overline{jitter} = 85.175\,182\,65 \times 10^{-6}$$

$$\overline{jitter} = \frac{jitterSum}{rxPackets-1}$$

$$\ldots = \frac{5\,357\,322\,353 \times 10^{-9}}{34044-1}$$

$$\overline{jitter} = 157.369\,278\,6 \times 10^{-6}$$

Conclusion

The ns3 Wifi Model utilised for section one of this assignment consists of two nodes, one acting as a source, and the other as a sink. Section two expands upon this, adding a third node as a sink, with the source now transmitting to both sink nodes.

One of the most significant differences between the TCP and UDP transfer protocols is that UDP is both connectionless and unreliable. This means that UDP begins transferring data to the client node, not requiring acknowledgements for received packets. As such, when packet loss occurs, no retransmission occurs. This can be seen clearly in the Two Node - Single Access Point model, wherein both nodes have an amount of loss occurring.

Using analysis tools such as Wireshark, it can be seen that no handshaking occurs. The transmitting node finds the address of the sink, and begins to transmit data. The optional "checksum" which can be used to detect errors in UDP packet transfers has also not been set. This can also be seen using Wireshark to analyse the separate packets.

Appendix

5.1 Single Source - Single Sink Model

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * This program conducts a simple experiment: It simulates two nodes comunicate
 * over a wireless channel using UDP. One node generates the packets and the
 * other node receives them. The throughput at the received node is calculated
 * every  0.1s and is written to the file 'udpWireless.dat'. ASCII and pcap packet
 * trace are also enable for detail packet exchange analysis.
 *
 * Author: Quang Le-Dang, Feb 2013 for EE452 <quang.ledang2@mail.dcu.ie>
 * Updated for 802.11ac: Anderson Simiscuka <anderson.simiscuka2@mail.dcu.ie>
 *
 *          EE452 Assignment 2018-2019
 *          Default network topology
 *
 *                   802.11ac
 *     Node 0 ------------------- Node 1
 *       |                          |
 *      UDP   ------------------> UDP
 *  traffic source             traffic sink
```

```
 *
 */

#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/mobility-module.h"
#include "ns3/wifi-module.h"
#include "ns3/applications-module.h"
#include <iostream>
#include "ns3/flow-monitor-module.h"


using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("udpWireless");

uint32_t totalBytesReceived(0), mbs (0);
std::ofstream rdTrace;

//------------------------------------------------------------------------
//-- Callback function is called whenever a packet is received successfully.
//-- This function cumulatively add the size of data packet to
//-- totalBytesReceived counter.
//------------------------------------------------------------------------
void
ReceivePacket(std::string context, Ptr <const Packet> p)
{
  totalBytesReceived += p->GetSize();
}

//------------------------------------------------------------------------
//-- Throughput calculating function.
//-- This function calculate the throughput every 0.1s using totalBytesReceived
//-- counter and write the results into a throughput tracefile
//------------------------------------------------------------------------
void
CalculateThroughput()
{
  double mbs = ((totalBytesReceived*8.0)/100000);
  totalBytesReceived =0;
  rdTrace << Simulator::Now ().GetSeconds() << "\t"<< mbs << "\n";
  Simulator::Schedule (Seconds (0.1), &CalculateThroughput);
}
```

```
//-----------------------------------------------------------------------------
// -- main
//-----------------------------------------------------------------------------
int
main (int argc, char *argv[])
{
  //LogComponentEnable ("PacketSink",LOG_LEVEL_INFO);
  //-------------------------------------------------
  //-- Open throughput trace file
  //-------------------------------------------------
  rdTrace.open("udpWireless.dat", std::ios::out);
  rdTrace << "# Time \t Throughput \n";

  //-------------------------------------------------
  //-- Creating nodes for simulation
  //-------------------------------------------------
  NodeContainer nodes;
  nodes.Create (2);

  //-------------------------------------------------
  //-- Creating Wifi channels for created nodes
  //-------------------------------------------------
  // Change in default constructor from ns3.25 to ns3.29
  // For use in ns3.25 uncomment lines conatining "::Default()",
  // comment lines using "()"
  // WifiHelper wifi = WifiHelper::Default ();
  WifiHelper wifi = WifiHelper();
  wifi.WifiHelper::SetStandard (WIFI_PHY_STANDARD_80211ac);
  wifi.SetRemoteStationManager ("ns3::MinstrelHtWifiManager");

  YansWifiChannelHelper channel = YansWifiChannelHelper::Default();
  channel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");
  channel.AddPropagationLoss ("ns3::NakagamiPropagationLossModel");
  YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default();
  wifiPhy.SetChannel (channel.Create());

  //WifiMacHelper wifiMac = WifiMacHelper::Default();
  WifiMacHelper wifiMac = WifiMacHelper();
  wifiMac.SetType ("ns3::AdhocWifiMac");

  NetDeviceContainer devices = wifi.Install (wifiPhy, wifiMac, nodes);

  //-------------------------------------------------
```

```
//-- Creating mobility models for created wireless nodes
//-------------------------------------------------------
MobilityHelper mobility;
Ptr<ListPositionAllocator> positionAlloc = CreateObject<ListPositionAllocator> ();
positionAlloc -> Add (Vector (0.0, 0.0, 0.0));
positionAlloc -> Add (Vector (5.0, 0.0, 0.0));
mobility.SetPositionAllocator (positionAlloc);
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (nodes);

//-------------------------------------------------------
//-- Install TCP/IP stack and assign IP address for nodes to communicate
//-------------------------------------------------------
InternetStackHelper stack;
stack.Install (nodes);

Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");

Ipv4InterfaceContainer interfaces = address.Assign (devices);

//-------------------------------------------------------
//-- Create, setup and install ON/OFF traffic generator
//-------------------------------------------------------
OnOffHelper onoff ("ns3::UdpSocketFactory",
                   Address(InetSocketAddress (interfaces.GetAddress (1), 9)));
onoff.SetAttribute ("OnTime",
          StringValue("ns3::ConstantRandomVariable[Constant=1.0]"));
onoff.SetAttribute ("OffTime",
          StringValue("ns3::ConstantRandomVariable[Constant=0.0]"));
onoff.SetAttribute ("PacketSize", StringValue ("1024"));
onoff.SetAttribute ("DataRate", StringValue ("10000kb/s"));

ApplicationContainer apps = onoff.Install (nodes.Get (0));
apps.Start (Seconds (2.0));
apps.Stop (Seconds (30.0));

//-------------------------------------------------------
//-- Creating a sink to receive the packets from the traffic generator
//-------------------------------------------------------
PacketSinkHelper sinkHelper ("ns3::UdpSocketFactory",
                             InetSocketAddress (Ipv4Address::GetAny (), 9));
ApplicationContainer sinkApp = sinkHelper.Install (nodes.Get(1));
sinkApp.Start (Seconds (2.0));
```

```
  sinkApp.Stop (Seconds (30.0));

  //-----------------------------------------------------
  //-- Connect MacRx event at MAC layer of sink node to ReceivePacket function
  //-- for throughput calculation.
  //-----------------------------------------------------
  Config::Connect ("/NodeList/1/DeviceList/*/$ns3::WifiNetDevice/Mac/MacRx",
           MakeCallback(&ReceivePacket) );

  //-----------------------------------------------------
  //-- Enable ASCII and PCAP tracing
  //-----------------------------------------------------
  AsciiTraceHelper ascii;
  wifiPhy.EnableAsciiAll (ascii.CreateFileStream("udpWireless.tr"));
  wifiPhy.EnablePcapAll("udpWireless");

  //-----------------------------------------------------
  //-- Trigger CalculateThroughput function, it will schedule itself afterwards
  //-----------------------------------------------------
  Simulator::Schedule (Seconds(0.0),&CalculateThroughput);

  //-----------------------------------------------------
  //-- Flow Monitor Setup
  //-----------------------------------------------------
  Ptr<FlowMonitor>flowMonitor;
  FlowMonitorHelper flowHelper;
  flowMonitor = flowHelper.InstallAll();

  //-----------------------------------------------------
  //-- Schedule a stop time at 60s and start the simulation
  //-----------------------------------------------------
  Simulator::Stop (Seconds (60));
  Simulator::Run ();
  flowMonitor->SerializeToXmlFile("TwoNodeData.xml", true, true);
  Simulator::Destroy ();

  return 0;
}
```

5.2 Single Source - Two Sink Model

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * This program conducts a simple experiment: It simulates two nodes comunicate
 * over a wireless channel using UDP. One node generates the packets and the
 * other node receives them. The throughput at the received node is calculated
 * every  0.1s and is written to the file 'udpWireless.dat'. ASCII and pcap packet
 * trace are also enable for detail packet exchange analysis.
 *
 * Author: Quang Le-Dang, Feb 2013 for EE452 <quang.ledang2@mail.dcu.ie>
 * Updated for 802.11ac: Anderson Simiscuka <anderson.simiscuka2@mail.dcu.ie>
 *
 *          EE452 Assignment 2018-2019
 *          Default network topology
 *
 *                   802.11ac
 *     Node 0 -------------------- Node 1
 *       |                           |
 *      UDP    ------------------> UDP
 *  traffic source              traffic sink
 *
 */

/* This Code has been modified to simulate one node generating packets, with two
 * nodes receiving them.
 * Modified by Michael Lenehan, Nov 2018
 */

#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/mobility-module.h"
#include "ns3/wifi-module.h"
#include "ns3/applications-module.h"
#include <iostream>
#include "ns3/flow-monitor-module.h"
#include "ns3/netanim-module.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("udpWireless");
```

```
uint32_t totalBytesReceived(0), mbs (0);
std::ofstream rdTrace;

//-------------------------------------------------------------------------
//-- Callback function is called whenever a packet is received successfully.
//-- This function cumulatively add the size of data packet to
//-- totalBytesReceived counter.
//-------------------------------------------------------------------------
void
ReceivePacket(std::string context, Ptr <const Packet> p)
{
  totalBytesReceived += p->GetSize();
}

//-------------------------------------------------------------------------
//-- Throughput calculating function.
//-- This function calculate the throughput every 0.1s using
//-- totalBytesReceived counter
//-- and write the results into a throughput tracefile
//-------------------------------------------------------------------------
void
CalculateThroughput()
{
  double mbs = ((totalBytesReceived*8.0)/100000);
  totalBytesReceived =0;
  rdTrace << Simulator::Now ().GetSeconds() << "\t"<< mbs << "\n";
  Simulator::Schedule (Seconds (0.1), &CalculateThroughput);
}

//-------------------------------------------------------------------------
// -- main
//-------------------------------------------------------------------------
int
main (int argc, char *argv[])
{
  //LogComponentEnable ("PacketSink",LOG_LEVEL_INFO);
  //-----------------------------------------------------
  //-- Open throughput trace file
  //-----------------------------------------------------
  rdTrace.open("udpWireless.dat", std::ios::out);
  rdTrace << "# Time \t Throughput \n";

  //-----------------------------------------------------
  //-- Creating nodes for simulation
```

```
//------------------------------------------------------
NodeContainer nodes;
nodes.Create (3);

//------------------------------------------------------
//-- Creating Wifi channels for created nodes
//------------------------------------------------------
// Change in default constructor from ns3.25 to ns3.29
// For use in ns3.25 uncomment lines conatining "::Default()", comment
// lines using "()"
// WifiHelper wifi = WifiHelper::Default ();
WifiHelper wifi = WifiHelper();
wifi.WifiHelper::SetStandard (WIFI_PHY_STANDARD_80211ac);
wifi.SetRemoteStationManager ("ns3::MinstrelHtWifiManager");

YansWifiChannelHelper channel = YansWifiChannelHelper::Default();
channel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");
channel.AddPropagationLoss ("ns3::NakagamiPropagationLossModel");
YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default();
wifiPhy.SetChannel (channel.Create());

//WifiMacHelper wifiMac = WifiMacHelper::Default();
WifiMacHelper wifiMac = WifiMacHelper();
wifiMac.SetType ("ns3::AdhocWifiMac");

NetDeviceContainer devices = wifi.Install (wifiPhy, wifiMac, nodes);

//------------------------------------------------------
//-- Creating mobility models for created wireless nodes
//------------------------------------------------------
MobilityHelper mobility;
Ptr<ListPositionAllocator> positionAlloc =
      CreateObject<ListPositionAllocator> ();
positionAlloc -> Add (Vector (0.0, 0.0, 0.0));
positionAlloc -> Add (Vector (5.0, 0.0, 0.0));
positionAlloc -> Add (Vector (0.0, 5.0, 0.0));
mobility.SetPositionAllocator (positionAlloc);
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (nodes);

//------------------------------------------------------
//-- Install TCP/IP stack and assign IP address for nodes to communicate
//------------------------------------------------------
InternetStackHelper stack;
```

```
stack.Install (nodes);

Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");

Ipv4InterfaceContainer interfaces = address.Assign (devices);

//------------------------------------------------------
//-- Create, setup and install ON/OFF traffic generator
//------------------------------------------------------
OnOffHelper onoff ("ns3::UdpSocketFactory",
                   Address (InetSocketAddress
          (interfaces.GetAddress (1), 9)));
onoff.SetAttribute ("OnTime", StringValue
          ("ns3::ConstantRandomVariable[Constant=1.0]"));
onoff.SetAttribute ("OffTime", StringValue
          ("ns3::ConstantRandomVariable[Constant=0.0]"));
onoff.SetAttribute ("PacketSize", StringValue ("1024"));
onoff.SetAttribute ("DataRate", StringValue ("10000kb/s"));

OnOffHelper onoff2 ("ns3::UdpSocketFactory",
                   Address (InetSocketAddress
          (interfaces.GetAddress (2), 9)));
onoff2.SetAttribute ("OnTime", StringValue
          ("ns3::ConstantRandomVariable[Constant=1.0]"));
onoff2.SetAttribute ("OffTime", StringValue
          ("ns3::ConstantRandomVariable[Constant=0.0]"));
onoff2.SetAttribute ("PacketSize", StringValue ("1024"));
onoff2.SetAttribute ("DataRate", StringValue ("10000kb/s"));

ApplicationContainer apps = onoff.Install (nodes.Get (0));
apps.Start (Seconds (2.0));
apps.Stop (Seconds (30.0));

ApplicationContainer apps2 = onoff2.Install (nodes.Get (0));
apps2.Start (Seconds (2.0));
apps2.Stop (Seconds (30.0));

//------------------------------------------------------
//-- Creating a sink to receive the packets from the traffic generator
//------------------------------------------------------
PacketSinkHelper sinkHelper ("ns3::UdpSocketFactory",
                             InetSocketAddress (Ipv4Address::GetAny (), 9));
PacketSinkHelper sinkHelper2 ("ns3::UdpSocketFactory",
```

```
                                InetSocketAddress (Ipv4Address::GetAny (), 9));
ApplicationContainer sinkApp = sinkHelper.Install (nodes.Get(1));
ApplicationContainer sinkApp2 = sinkHelper2.Install (nodes.Get(2));
sinkApp.Start (Seconds (2.0));
sinkApp.Stop (Seconds (30.0));
sinkApp2.Start (Seconds (2.0));
sinkApp2.Stop (Seconds (30));

//-----------------------------------------------------
//-- Connect MacRx event at MAC layer of sink node to ReceivePacket function
//-- for throughput calculation.
//-----------------------------------------------------
Config::Connect ("/NodeList/1/DeviceList/*/$ns3::WifiNetDevice/Mac/MacRx",
          MakeCallback(&ReceivePacket) );
Config::Connect ("/NodeList/2/DeviceList/*/$ns3::WifiNetDevice/Mac/MacRX",
          MakeCallback(&ReceivePacket) );

//-----------------------------------------------------
//-- Enable ASCII and PCAP tracing
//-----------------------------------------------------
AsciiTraceHelper ascii;
wifiPhy.EnableAsciiAll (ascii.CreateFileStream("udpWireless.tr"));
wifiPhy.EnablePcapAll("udpWireless");

//-----------------------------------------------------
//-- Trigger CalculateThroughput function, it will schedule itself afterwards
//-----------------------------------------------------
Simulator::Schedule (Seconds(0.0),&CalculateThroughput);

//-----------------------------------------------------
//-- Flow Monitor Setup
//-----------------------------------------------------
AnimationInterface anim ("animation.xml");
Ptr<FlowMonitor>flowMonitor;
FlowMonitorHelper flowHelper;
flowMonitor = flowHelper.InstallAll();

//-----------------------------------------------------
//-- Schedule a stop time at 60s and start the simulation
//-----------------------------------------------------
Simulator::Stop (Seconds (60));
Simulator::Run ();
flowMonitor->SerializeToXmlFile("ThreeNodeData.xml", true, true);
Simulator::Destroy ();
```

```
  return 0;
}
```