Computer Networks Assignment 3

Harsh Prajapati - 12241300

September 14, 2024

Question 1

This question has 4 subparts.

There is only one code file (code.cc) to understand the whole topology of the question which will give an idea of simulation of all the steps.

To visualize the simulation at all 4 steps/levels ,pass the number in the argument.

Run the code from terminal using,
./ns3 run (path to file in scratch directory) – –question=(question number)

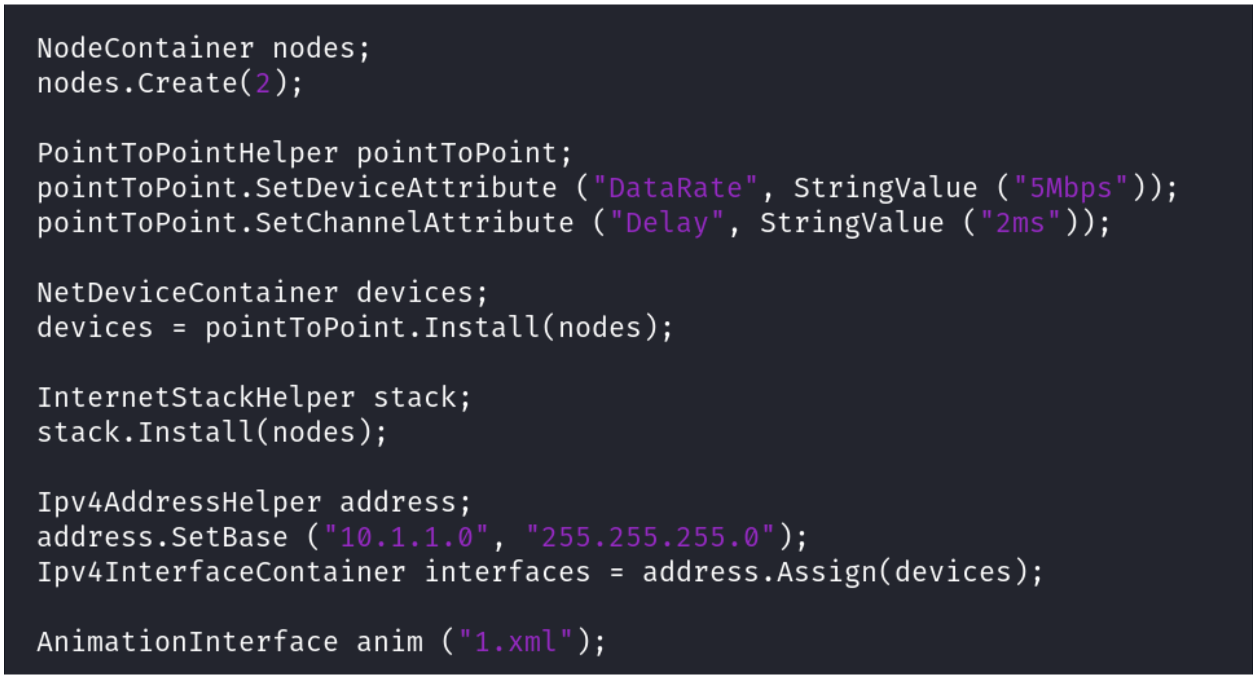
Note: I had to pass one extra pair of double hyphens(–) from the Command line to correctly pass arguments.If it creates any error kindly remove that pair and then run the code again from command line.

It will create a **(question number).xml** file for subpart 1,2,4 and in the third part we are supposed to check throughput of the link for different latency values.

Hence it will generate **animation_(latency value).xml** for 3 different latencies fixed in the code.

In the first part, we have to create a simple topology of two nodes connected via point to point link.

Part 1



```
NodeContainer nodes;
nodes.Create(2);

PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

NetDeviceContainer devices;
devices = pointToPoint.Install(nodes);

InternetStackHelper stack;
stack.Install(nodes);

Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer interfaces = address.Assign(devices);

AnimationInterface anim ("1.xml");
```

Figure 1: Setting up two nodes with point to point link

- In the above image we can see the setup of two nodes connected with a link.
- First, we setup two nodes using **NodeContainer**.
- **PointToPointHelper** class used to setup point-to-point link between the nodes.(With data rate=5 Mbps, Delay=2ms).
- **NetDeviceContainer** set up Network devices on the nodes.

  PointToPoint.Install(nodes) connects two nodes created earlier, simulating a physical link between them.
- **InternetStackHelper** class installs the TCP/IP stack on the nodes.
- Then we assigned IP addresses to the nodes using **Ipv4AddressHelper** class and stored the animation in 1.xml file.

The simulation for the above setup is shown in the image below.

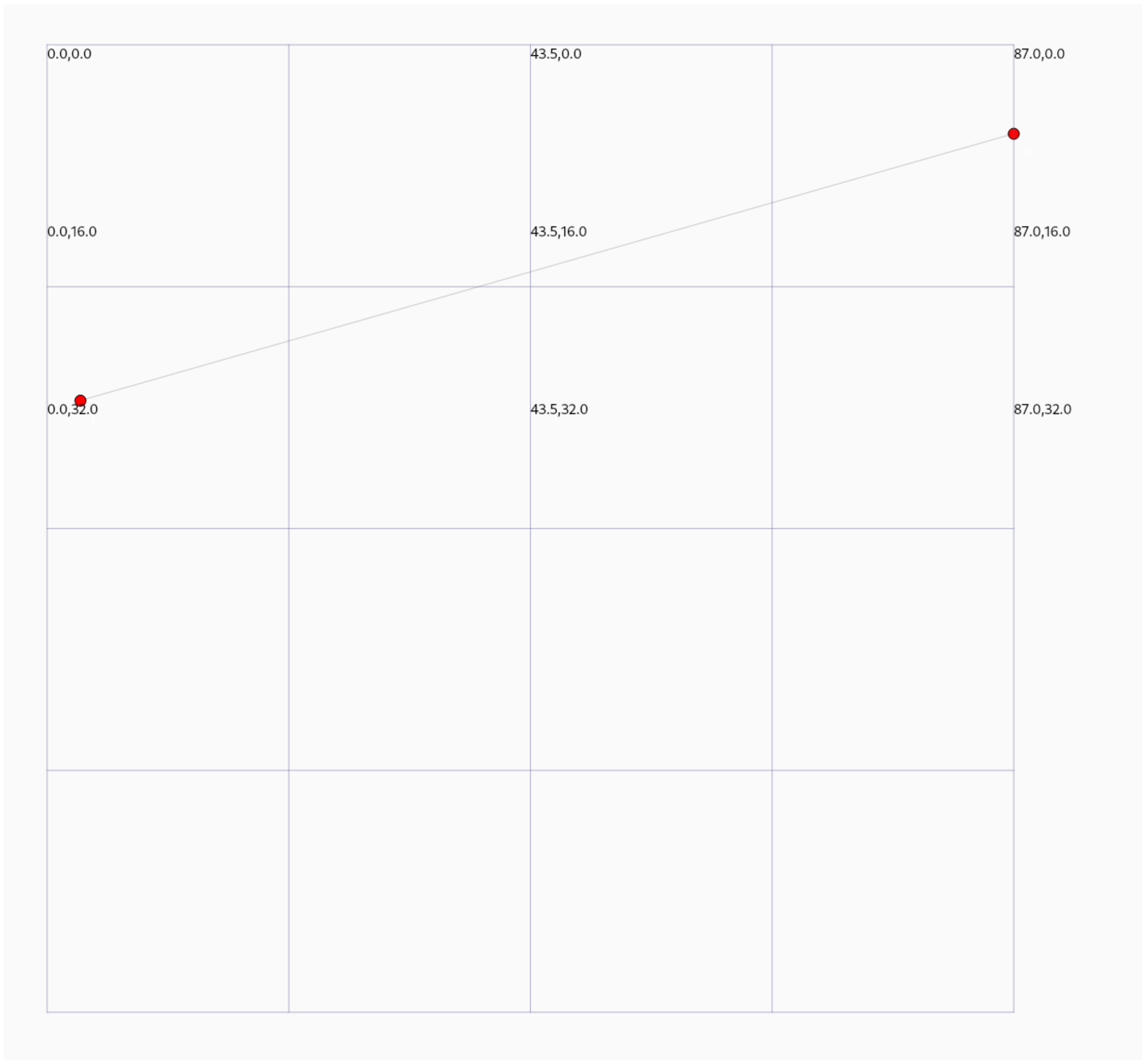
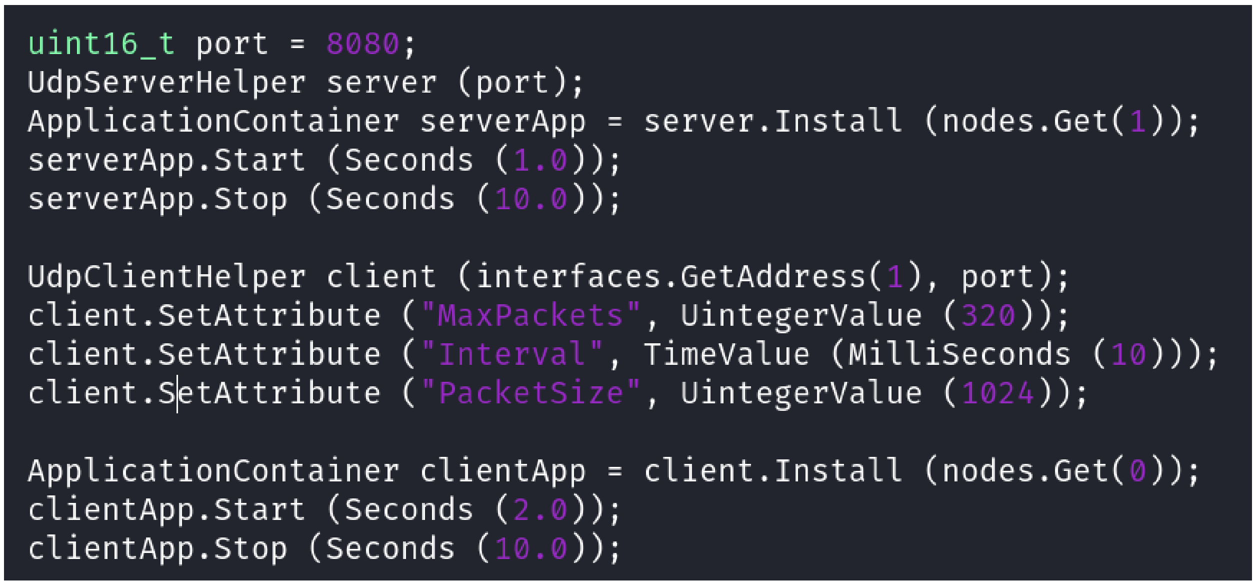


Figure 2: Simulation for two nodes

Part 2



```
uint16_t port = 8080;
UdpServerHelper server (port);
ApplicationContainer serverApp = server.Install (nodes.Get(1));
serverApp.Start (Seconds (1.0));
serverApp.Stop (Seconds (10.0));

UdpClientHelper client (interfaces.GetAddress(1), port);
client.SetAttribute ("MaxPackets", UintegerValue (320));
client.SetAttribute ("Interval", TimeValue (MilliSeconds (10)));
client.SetAttribute ("PacketSize", UintegerValue (1024));

ApplicationContainer clientApp = client.Install (nodes.Get(0));
clientApp.Start (Seconds (2.0));
clientApp.Stop (Seconds (10.0));
```

Figure 3: Setting up UDP server and UDP client

- In the second part, we have to set up UdpClient and UdpServer on node1 and node2 respectively.
- We used the **UdpServerHelper** class, **UdpClientHelper** class and **Application-Container** class for this purpose.
- We set up a max number of packets to be sent by the client at 320 on the interval of 10 milliseconds.
- Client will start sending packets in 2.0 seconds.

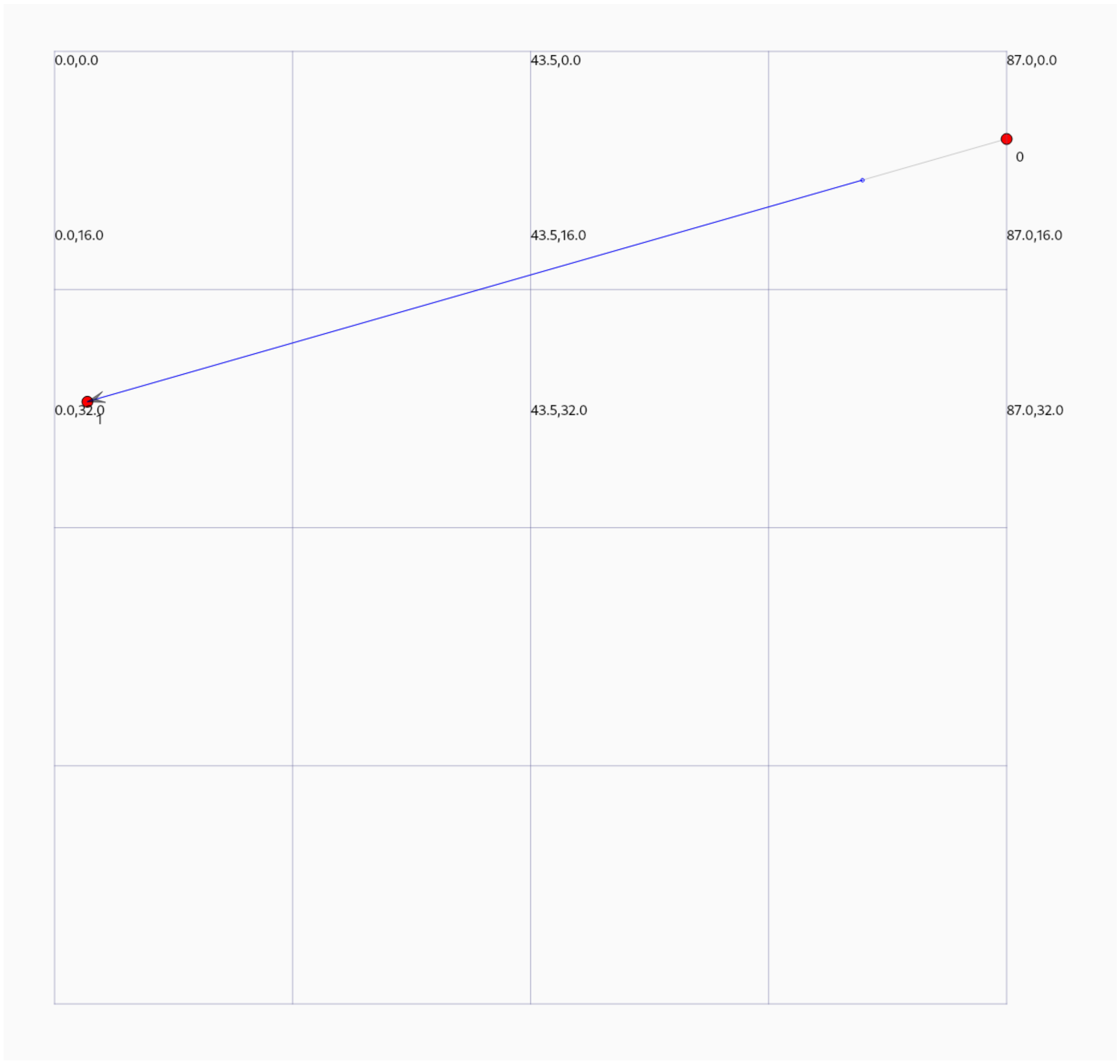
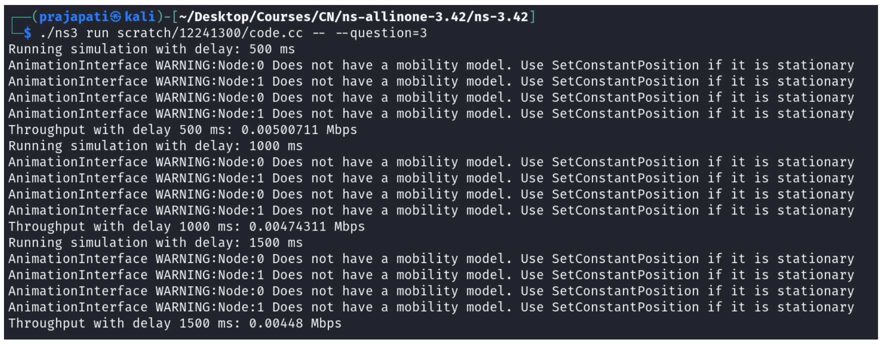


Figure 4: Simulation after UDP setup

Part 3

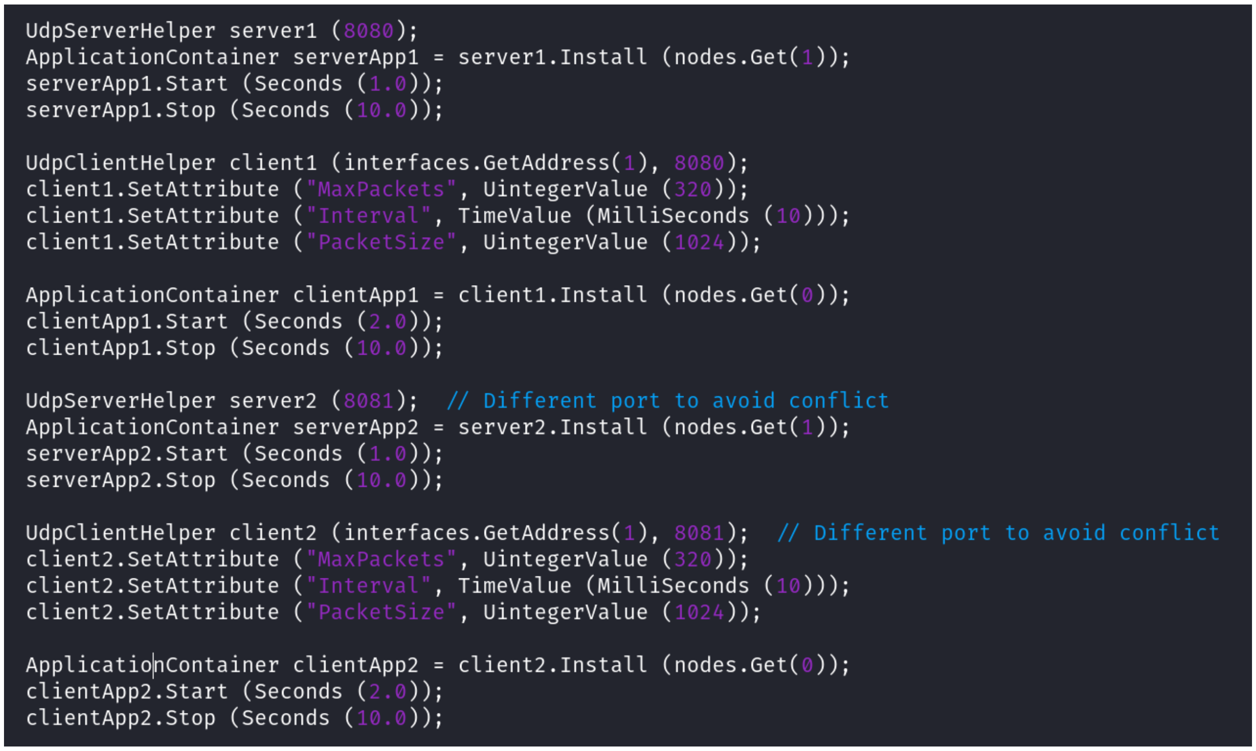


```
┌──(prajapati㉿kali)-[~/Desktop/Courses/CN/ns-allinone-3.42/ns-3.42]
└─$ ./ns3 run scratch/12241300/code.cc -- --question=3
Running simulation with delay: 500 ms
AnimationInterface WARNING:Node:0 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:1 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:0 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:1 Does not have a mobility model. Use SetConstantPosition if it is stationary
Throughput with delay 500 ms: 0.00500711 Mbps
Running simulation with delay: 1000 ms
AnimationInterface WARNING:Node:0 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:1 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:0 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:1 Does not have a mobility model. Use SetConstantPosition if it is stationary
Throughput with delay 1000 ms: 0.00474311 Mbps
Running simulation with delay: 1500 ms
AnimationInterface WARNING:Node:0 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:1 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:0 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:1 Does not have a mobility model. Use SetConstantPosition if it is stationary
Throughput with delay 1500 ms: 0.00448 Mbps
```

Figure 5: Throughput of the links with different delays

- In the 3rd part, we calculated the throughput of the link with the different Delays of **500ms, 1000ms and 1500ms**.
- It is **0.00500 Mbps, 0.00474 Mbps and 0.00448 Mbps** respectively.
- Which shows that increasing the delay of the link will decrease the throughput.
- Simulation can be shown in respective files with names **animation_(delayMs).xml** files.

Part 4



```
UdpServerHelper server1 (8080);
ApplicationContainer serverApp1 = server1.Install (nodes.Get(1));
serverApp1.Start (Seconds (1.0));
serverApp1.Stop (Seconds (10.0));

UdpClientHelper client1 (interfaces.GetAddress(1), 8080);
client1.SetAttribute ("MaxPackets", UintegerValue (320));
client1.SetAttribute ("Interval", TimeValue (MilliSeconds (10)));
client1.SetAttribute ("PacketSize", UintegerValue (1024));

ApplicationContainer clientApp1 = client1.Install (nodes.Get(0));
clientApp1.Start (Seconds (2.0));
clientApp1.Stop (Seconds (10.0));

UdpServerHelper server2 (8081);  // Different port to avoid conflict
ApplicationContainer serverApp2 = server2.Install (nodes.Get(1));
serverApp2.Start (Seconds (1.0));
serverApp2.Stop (Seconds (10.0));

UdpClientHelper client2 (interfaces.GetAddress(1), 8081);  // Different port to avoid conflict
client2.SetAttribute ("MaxPackets", UintegerValue (320));
client2.SetAttribute ("Interval", TimeValue (MilliSeconds (10)));
client2.SetAttribute ("PacketSize", UintegerValue (1024));

ApplicationContainer clientApp2 = client2.Install (nodes.Get(0));
clientApp2.Start (Seconds (2.0));
clientApp2.Stop (Seconds (10.0));
```

Figure 6: Server connected to 2 client applications on ports 8080 and 8081 respectively

- In the above image, we added another application on the client side and connected it with the server but on a different port number to avoid conflicts.
- **ClientApp1 → Server port 8080**
- **ClientApp2 → Server port 8081**
- Simulation for this is shown using a **4.xml** file.

Question 2

Part A

First, we have to create a custom network topology defined in the question as:(node index 3,4,5 are showing router positions)

- **n0 (index = 0), n1 (1)**, and **n2 (2)** are connected to **router0 (3)**.
- **Router0 (3)** is connected to both **router1 (4)** and **router2 (5)**.
- **n3 (6), n4 (7)** and **n5 (8)** are connected to **router1 (4)**.
- **n6 (9)** and **n7 (10)** are connected to **router2 (5)**

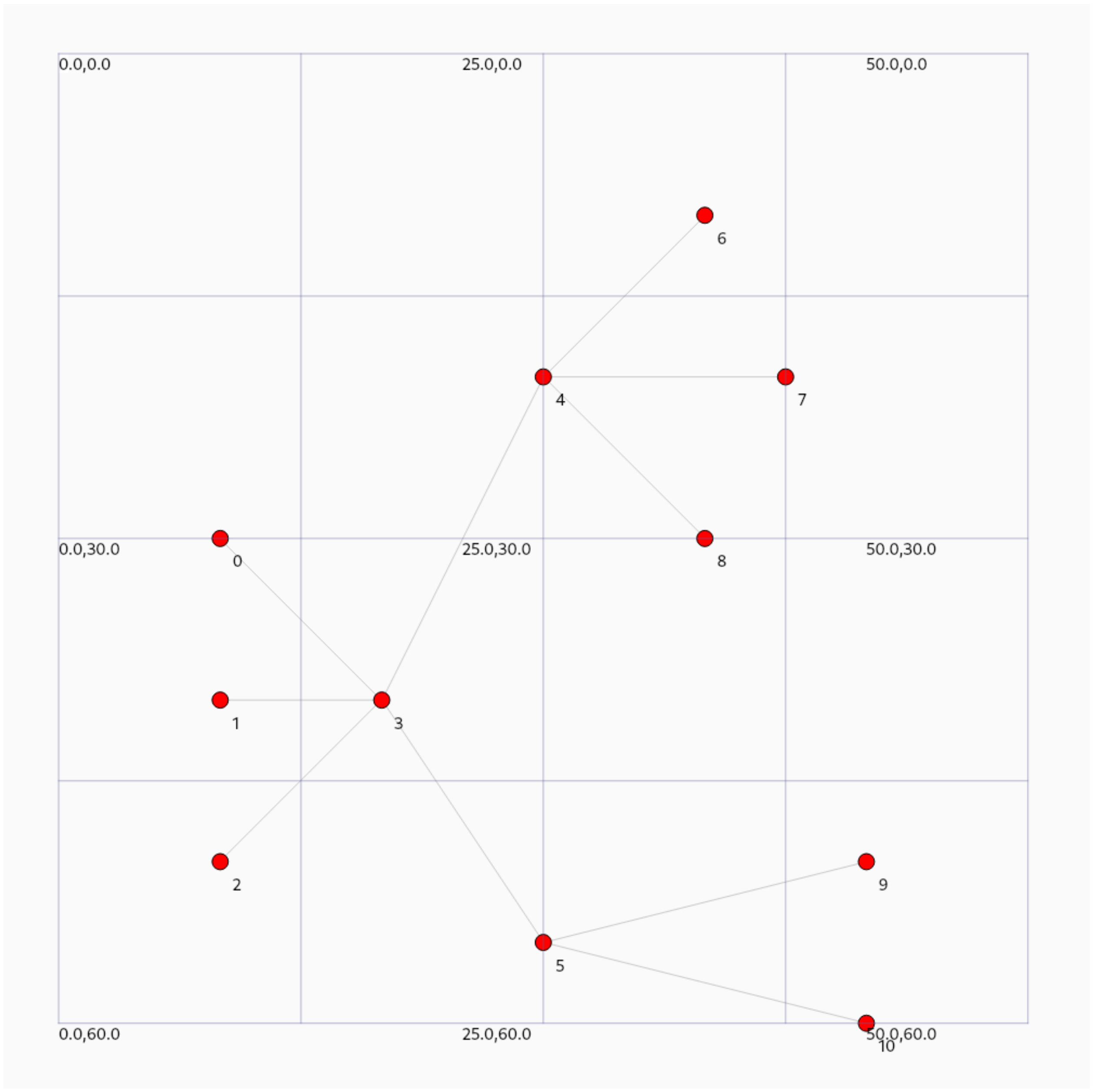


Figure 7: Network Topology setup

Flow between these links is shown in the below figure using one snapshot of the simulation file.

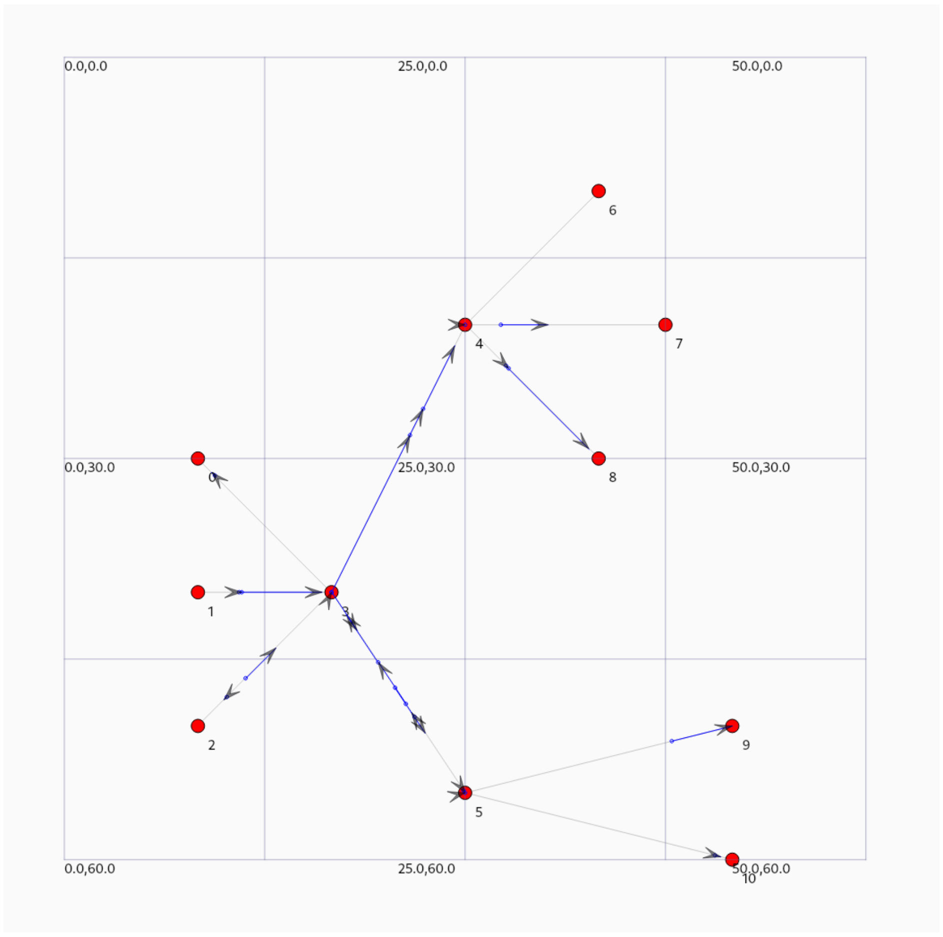
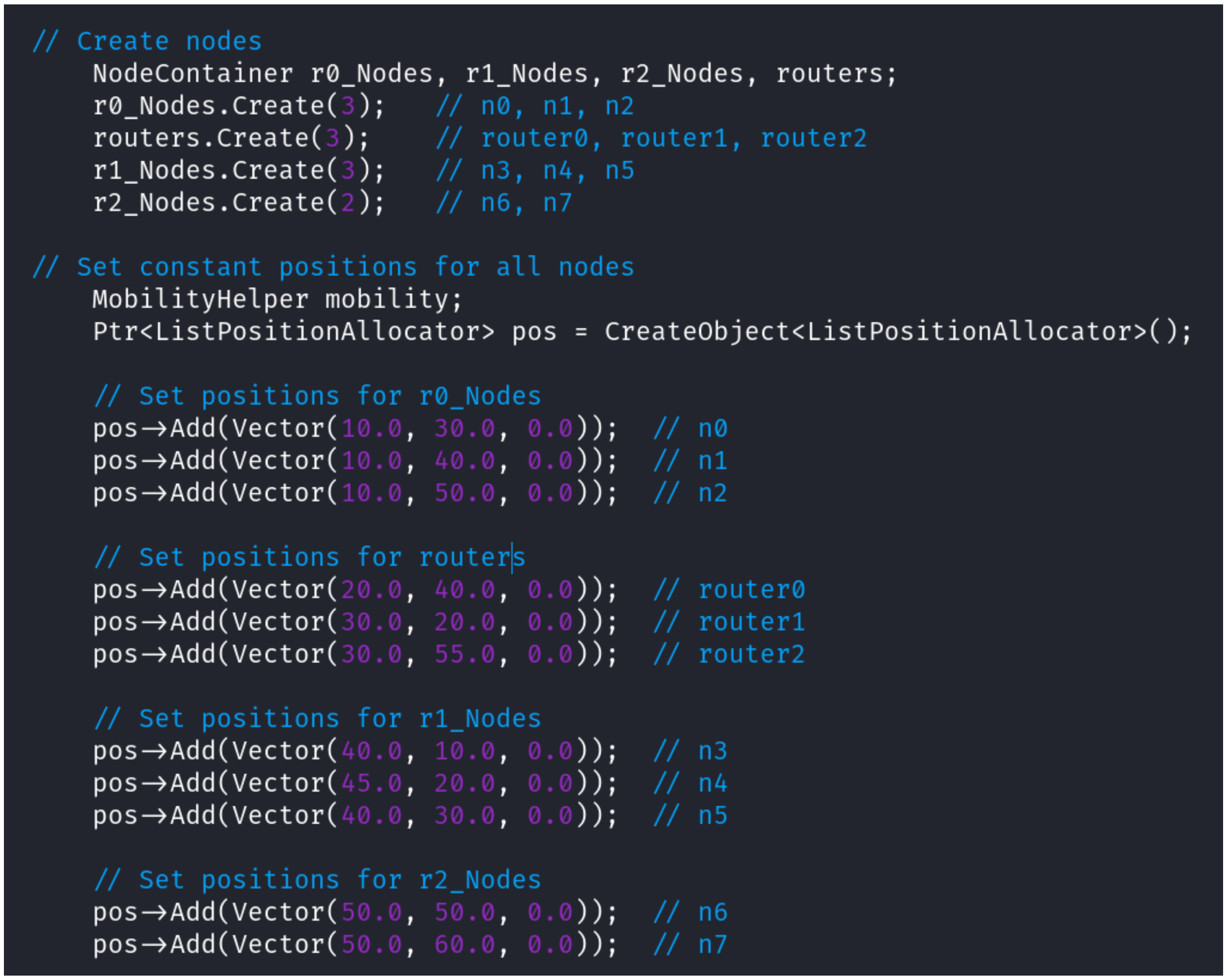


Figure 8: An instace of TCP and UDP connection flow

Now, let's take a look at the code structure for this connection setup.

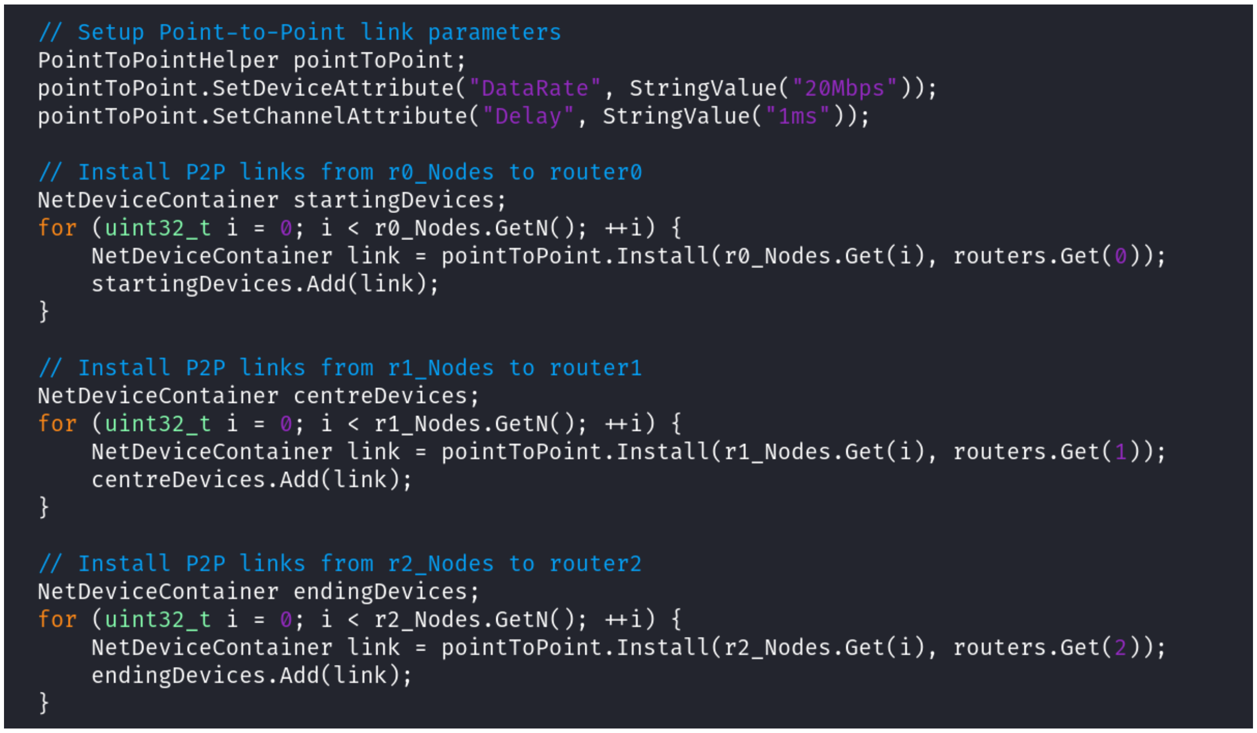


```
// Create nodes
    NodeContainer r0_Nodes, r1_Nodes, r2_Nodes, routers;
    r0_Nodes.Create(3);    // n0, n1, n2
    routers.Create(3);     // router0, router1, router2
    r1_Nodes.Create(3);    // n3, n4, n5
    r2_Nodes.Create(2);    // n6, n7

// Set constant positions for all nodes
    MobilityHelper mobility;
    Ptr<ListPositionAllocator> pos = CreateObject<ListPositionAllocator>();

    // Set positions for r0_Nodes
    pos->Add(Vector(10.0, 30.0, 0.0));  // n0
    pos->Add(Vector(10.0, 40.0, 0.0));  // n1
    pos->Add(Vector(10.0, 50.0, 0.0));  // n2

    // Set positions for routers
    pos->Add(Vector(20.0, 40.0, 0.0));  // router0
    pos->Add(Vector(30.0, 20.0, 0.0));  // router1
    pos->Add(Vector(30.0, 55.0, 0.0));  // router2

    // Set positions for r1_Nodes
    pos->Add(Vector(40.0, 10.0, 0.0));  // n3
    pos->Add(Vector(45.0, 20.0, 0.0));  // n4
    pos->Add(Vector(40.0, 30.0, 0.0));  // n5

    // Set positions for r2_Nodes
    pos->Add(Vector(50.0, 50.0, 0.0));  // n6
    pos->Add(Vector(50.0, 60.0, 0.0));  // n7
```

Figure 9: Node setup code

As shown in the above figure,first, we created the nodes and routers as asked for the topology setup and fixed their position for the simulation.

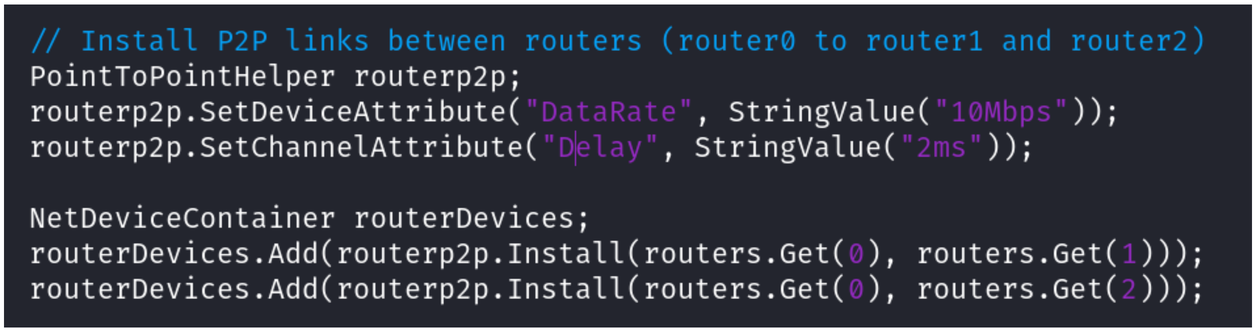
- **r0_Nodes** created for the nodes connected to router0
- **Routers** Container created for the routers in the topology
- **r1_Nodes** created for the nodes connected to router1
- **r2_Nodes** created for the nodes connected to router2



```
// Setup Point-to-Point link parameters
PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute("DataRate", StringValue("20Mbps"));
pointToPoint.SetChannelAttribute("Delay", StringValue("1ms"));

// Install P2P links from r0_Nodes to router0
NetDeviceContainer startingDevices;
for (uint32_t i = 0; i < r0_Nodes.GetN(); ++i) {
    NetDeviceContainer link = pointToPoint.Install(r0_Nodes.Get(i), routers.Get(0));
    startingDevices.Add(link);
}

// Install P2P links from r1_Nodes to router1
NetDeviceContainer centreDevices;
for (uint32_t i = 0; i < r1_Nodes.GetN(); ++i) {
    NetDeviceContainer link = pointToPoint.Install(r1_Nodes.Get(i), routers.Get(1));
    centreDevices.Add(link);
}

// Install P2P links from r2_Nodes to router2
NetDeviceContainer endingDevices;
for (uint32_t i = 0; i < r2_Nodes.GetN(); ++i) {
    NetDeviceContainer link = pointToPoint.Install(r2_Nodes.Get(i), routers.Get(2));
    endingDevices.Add(link);
}
```

Figure 10: P2P link setup between node and router

Then we created a point-to-point link between the nodes and respective routers with **DataRate = 20 Mbps** and **Delay = 1ms** as shown in the above figure.But, we are not yet done with the configuration since we have to connect routers with each other as described in the question.

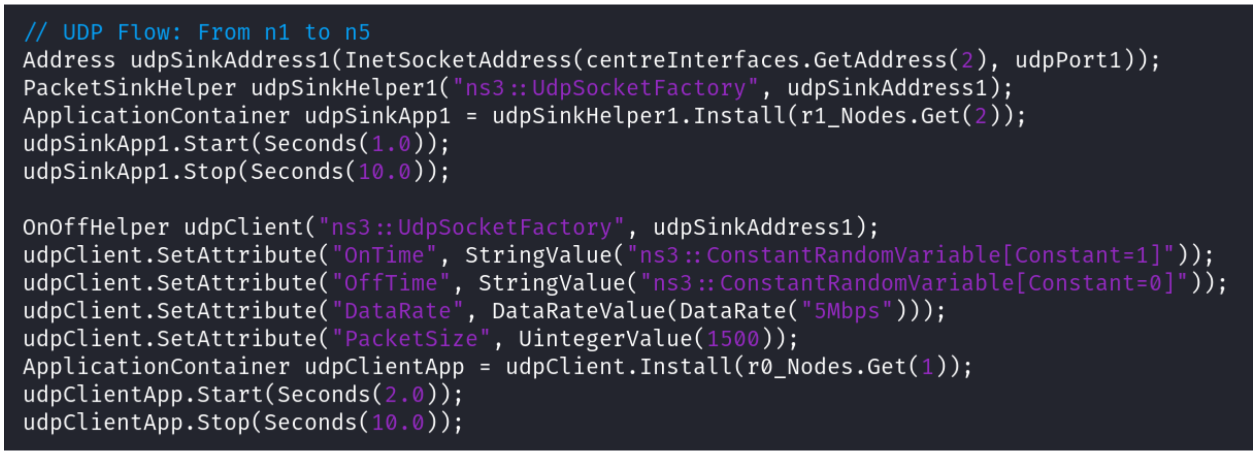


```
// Install P2P links between routers (router0 to router1 and router2)
PointToPointHelper routerp2p;
routerp2p.SetDeviceAttribute("DataRate", StringValue("10Mbps"));
routerp2p.SetChannelAttribute("Delay", StringValue("2ms"));

NetDeviceContainer routerDevices;
routerDevices.Add(routerp2p.Install(routers.Get(0), routers.Get(1)));
routerDevices.Add(routerp2p.Install(routers.Get(0), routers.Get(2)));
```

Figure 11: Link setup between Routers

The above figure shows the connection between routers we did for the setup.This connection has **DataRate = 10 Mbps** and **Delay = 2 ms**.
After this, we assigned IP addresses to all the nodes which was only for simplicity to understand terminal output we will discuss later.

Now, since we were done with creating links between nodes, we set up UDP and TCP connections which were asked in the question.



```
// UDP Flow: From n1 to n5
Address udpSinkAddress1(InetSocketAddress(centreInterfaces.GetAddress(2), udpPort1));
PacketSinkHelper udpSinkHelper1("ns3::UdpSocketFactory", udpSinkAddress1);
ApplicationContainer udpSinkApp1 = udpSinkHelper1.Install(r1_Nodes.Get(2));
udpSinkApp1.Start(Seconds(1.0));
udpSinkApp1.Stop(Seconds(10.0));

OnOffHelper udpClient("ns3::UdpSocketFactory", udpSinkAddress1);
udpClient.SetAttribute("OnTime", StringValue("ns3::ConstantRandomVariable[Constant=1]"));
udpClient.SetAttribute("OffTime", StringValue("ns3::ConstantRandomVariable[Constant=0]"));
udpClient.SetAttribute("DataRate", DataRateValue(DataRate("5Mbps")));
udpClient.SetAttribute("PacketSize", UintegerValue(1500));
ApplicationContainer udpClientApp = udpClient.Install(r0_Nodes.Get(1));
udpClientApp.Start(Seconds(2.0));
udpClientApp.Stop(Seconds(10.0));
```

Figure 12: Installing internet stack

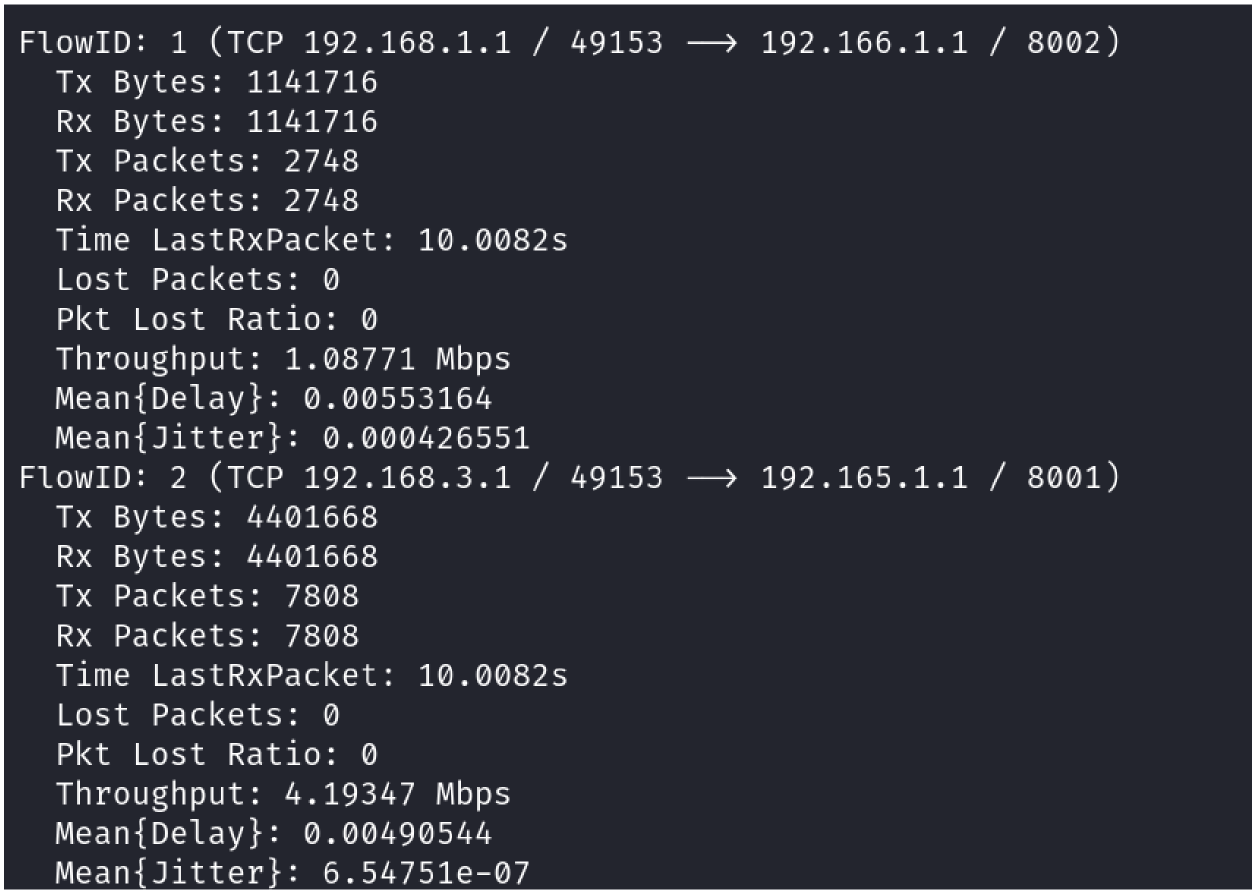
In the above figure, we created **UDP flow between n1 and n5 with DataRate = 5 Mbps and Packet Size = 1500 Bytes.**

After setting up UDP connection, we set up 3 TCP connections with the same method varying packet size and DataRate.

Now, since we were done with the setup we executed the code and showed the stats collected by Flow Monitor on the Terminal Emulator.

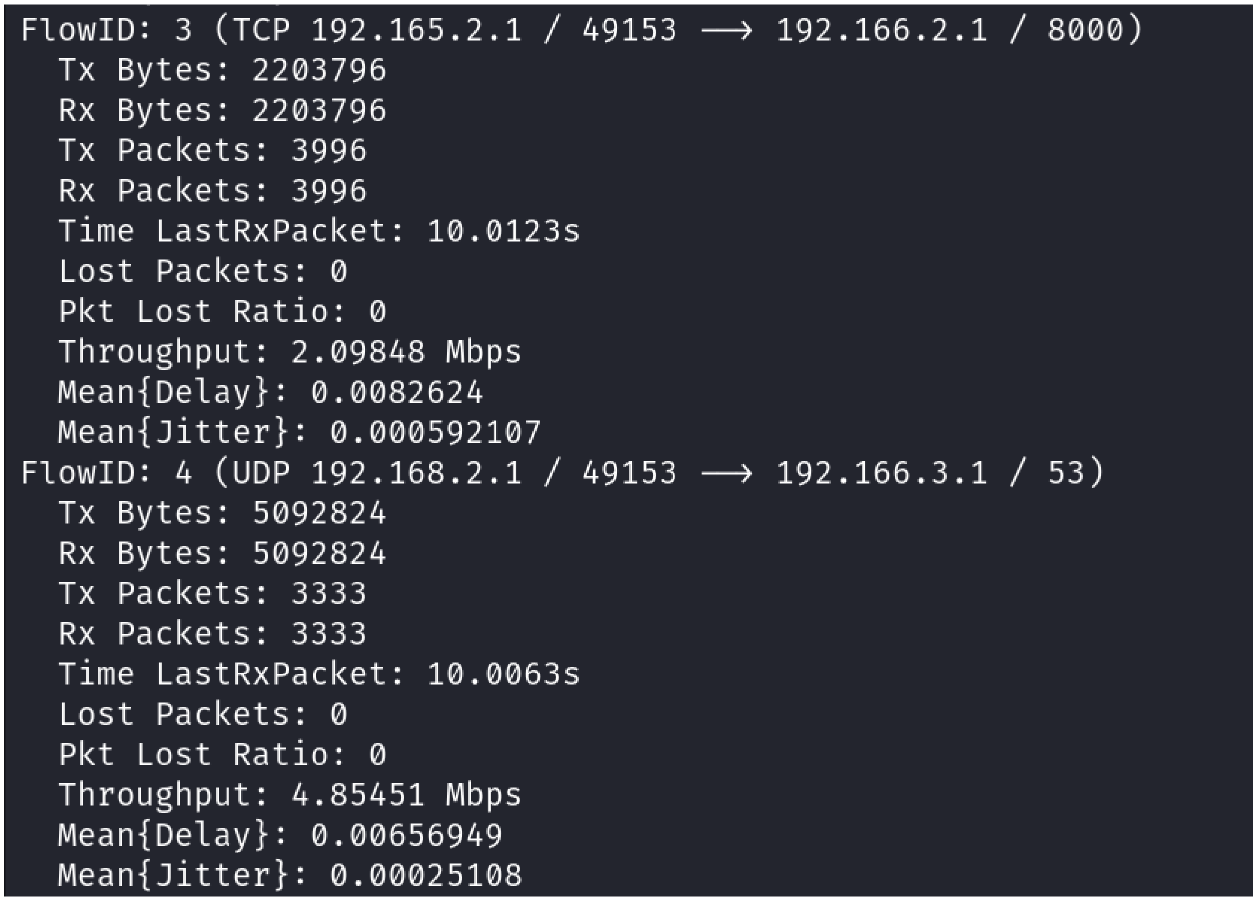
Next 3 figures show the stats of the connection collected by Flow Monitor.

This data includes Flow Direction, Tx and Rx packets information, Number of lost Packets and Throughput of the link and some other stats which we can show in the figures.



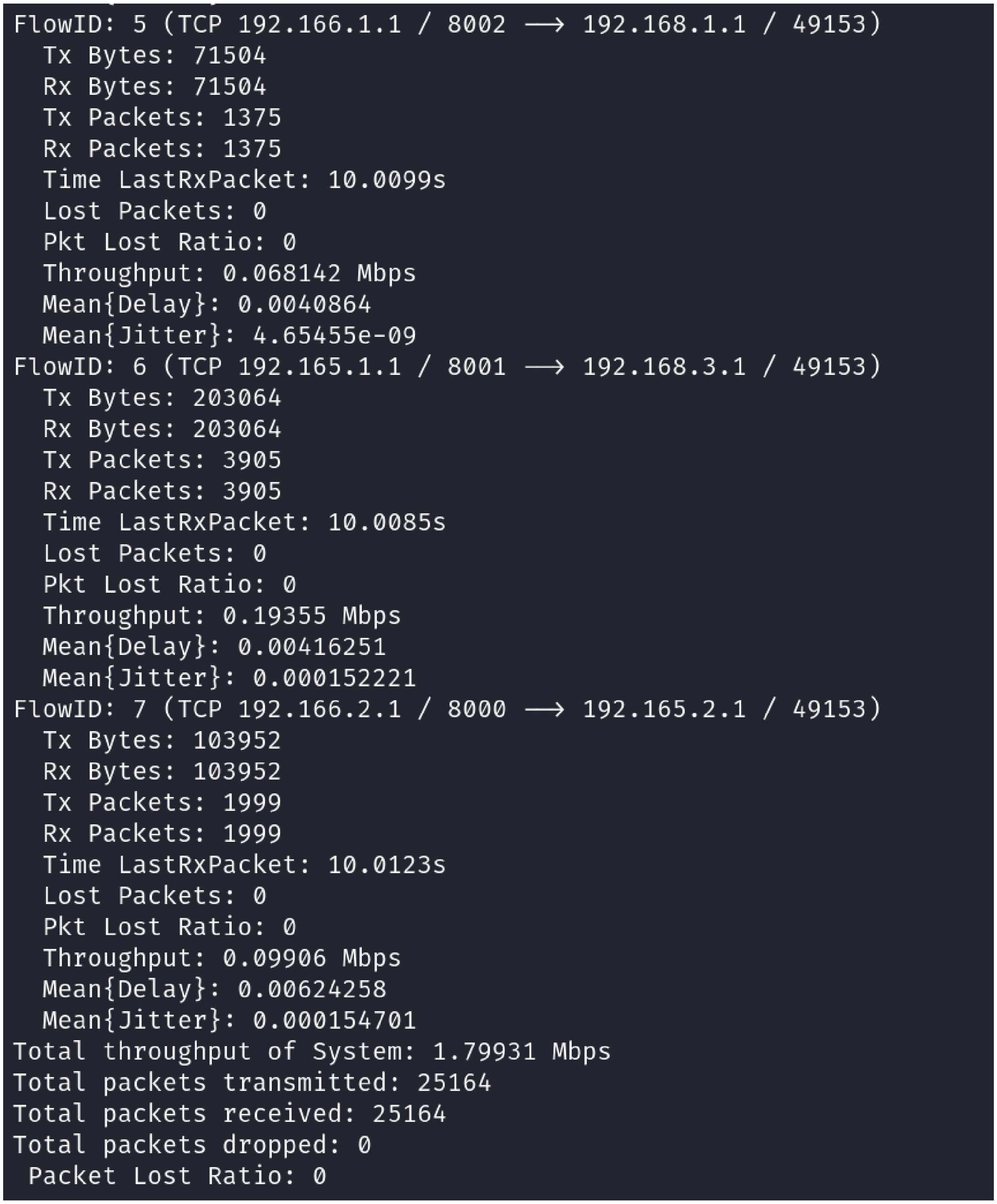
```
FlowID: 1 (TCP 192.168.1.1 / 49153 --> 192.166.1.1 / 8002)
  Tx Bytes: 1141716
  Rx Bytes: 1141716
  Tx Packets: 2748
  Rx Packets: 2748
  Time LastRxPacket: 10.0082s
  Lost Packets: 0
  Pkt Lost Ratio: 0
  Throughput: 1.08771 Mbps
  Mean{Delay}: 0.00553164
  Mean{Jitter}: 0.000426551
FlowID: 2 (TCP 192.168.3.1 / 49153 --> 192.165.1.1 / 8001)
  Tx Bytes: 4401668
  Rx Bytes: 4401668
  Tx Packets: 7808
  Rx Packets: 7808
  Time LastRxPacket: 10.0082s
  Lost Packets: 0
  Pkt Lost Ratio: 0
  Throughput: 4.19347 Mbps
  Mean{Delay}: 0.00490544
  Mean{Jitter}: 6.54751e-07
```

Figure 13: First 2 TCP flows



```
FlowID: 3 (TCP 192.165.2.1 / 49153 ⟶ 192.166.2.1 / 8000)
  Tx Bytes: 2203796
  Rx Bytes: 2203796
  Tx Packets: 3996
  Rx Packets: 3996
  Time LastRxPacket: 10.0123s
  Lost Packets: 0
  Pkt Lost Ratio: 0
  Throughput: 2.09848 Mbps
  Mean{Delay}: 0.0082624
  Mean{Jitter}: 0.000592107
FlowID: 4 (UDP 192.168.2.1 / 49153 ⟶ 192.166.3.1 / 53)
  Tx Bytes: 5092824
  Rx Bytes: 5092824
  Tx Packets: 3333
  Rx Packets: 3333
  Time LastRxPacket: 10.0063s
  Lost Packets: 0
  Pkt Lost Ratio: 0
  Throughput: 4.85451 Mbps
  Mean{Delay}: 0.00656949
  Mean{Jitter}: 0.00025108
```

Figure 14:



```
FlowID: 5 (TCP 192.166.1.1 / 8002 --> 192.168.1.1 / 49153)
  Tx Bytes: 71504
  Rx Bytes: 71504
  Tx Packets: 1375
  Rx Packets: 1375
  Time LastRxPacket: 10.0099s
  Lost Packets: 0
  Pkt Lost Ratio: 0
  Throughput: 0.068142 Mbps
  Mean{Delay}: 0.0040864
  Mean{Jitter}: 4.65455e-09
FlowID: 6 (TCP 192.165.1.1 / 8001 --> 192.168.3.1 / 49153)
  Tx Bytes: 203064
  Rx Bytes: 203064
  Tx Packets: 3905
  Rx Packets: 3905
  Time LastRxPacket: 10.0085s
  Lost Packets: 0
  Pkt Lost Ratio: 0
  Throughput: 0.19355 Mbps
  Mean{Delay}: 0.00416251
  Mean{Jitter}: 0.000152221
FlowID: 7 (TCP 192.166.2.1 / 8000 --> 192.165.2.1 / 49153)
  Tx Bytes: 103952
  Rx Bytes: 103952
  Tx Packets: 1999
  Rx Packets: 1999
  Time LastRxPacket: 10.0123s
  Lost Packets: 0
  Pkt Lost Ratio: 0
  Throughput: 0.09906 Mbps
  Mean{Delay}: 0.00624258
  Mean{Jitter}: 0.000154701
Total throughput of System: 1.79931 Mbps
Total packets transmitted: 25164
Total packets received: 25164
Total packets dropped: 0
 Packet Lost Ratio: 0
```

Figure 15: TCP flows from Router to client

We generated **A_flow_monitor.xml , part_A.xml (netanim file)** and **part_A_routing_tables.txt** files from the code.

We made a python script (**part_A_plots.py**) to plot graphs of the Flow vs Throughput and Flow vs Packet Loss from the stats collected by Flow Monitor.

The **Flow vs Throughput graph** is shown below:

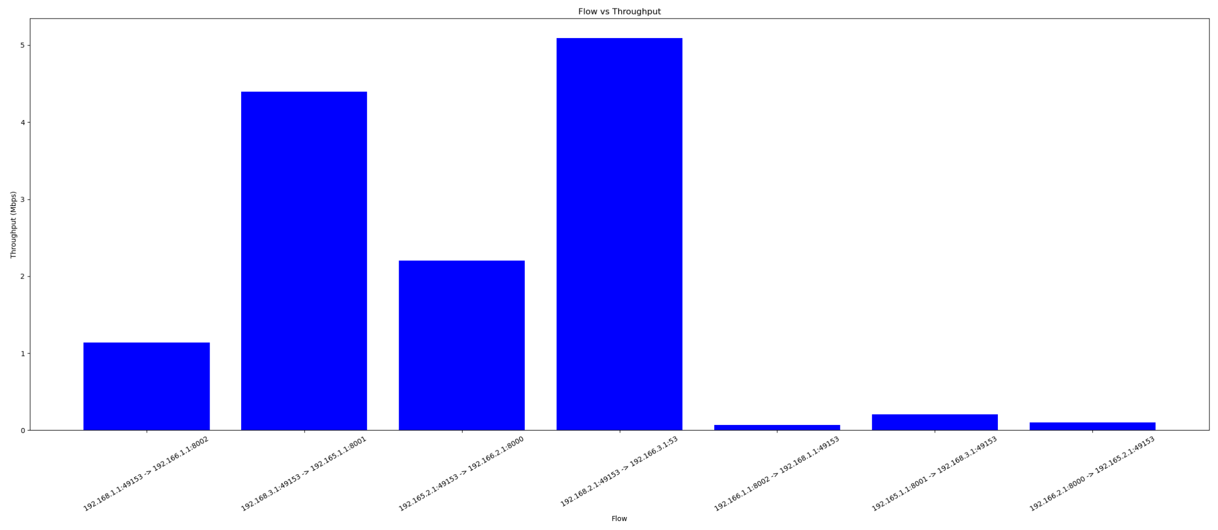


Figure 16: Flow vs Throughput Graph

The above graph shows that the more the DataRate the higher the throughput you can get.

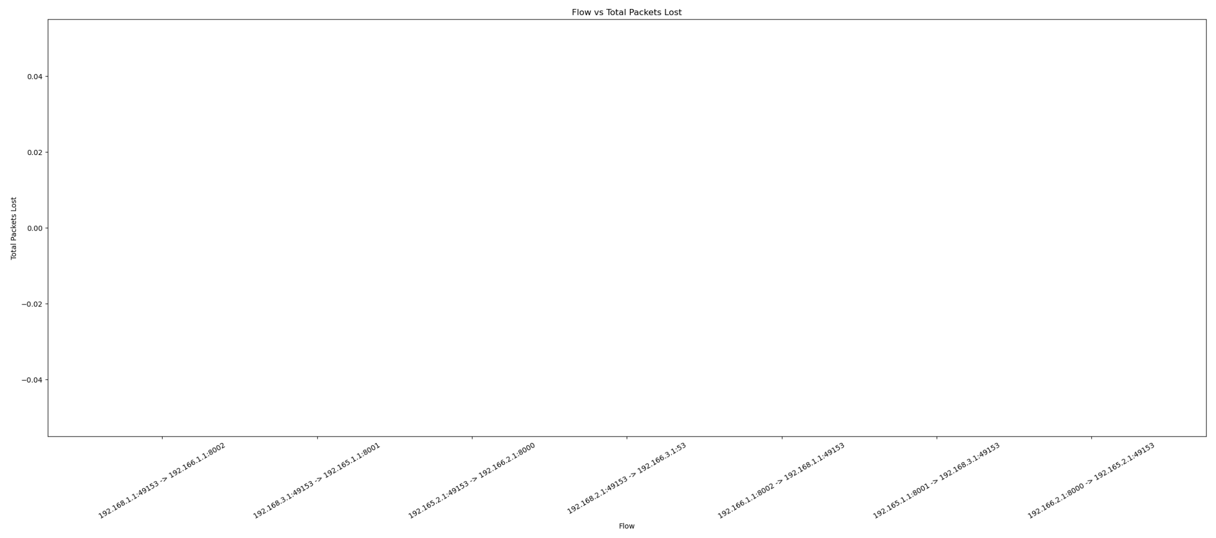
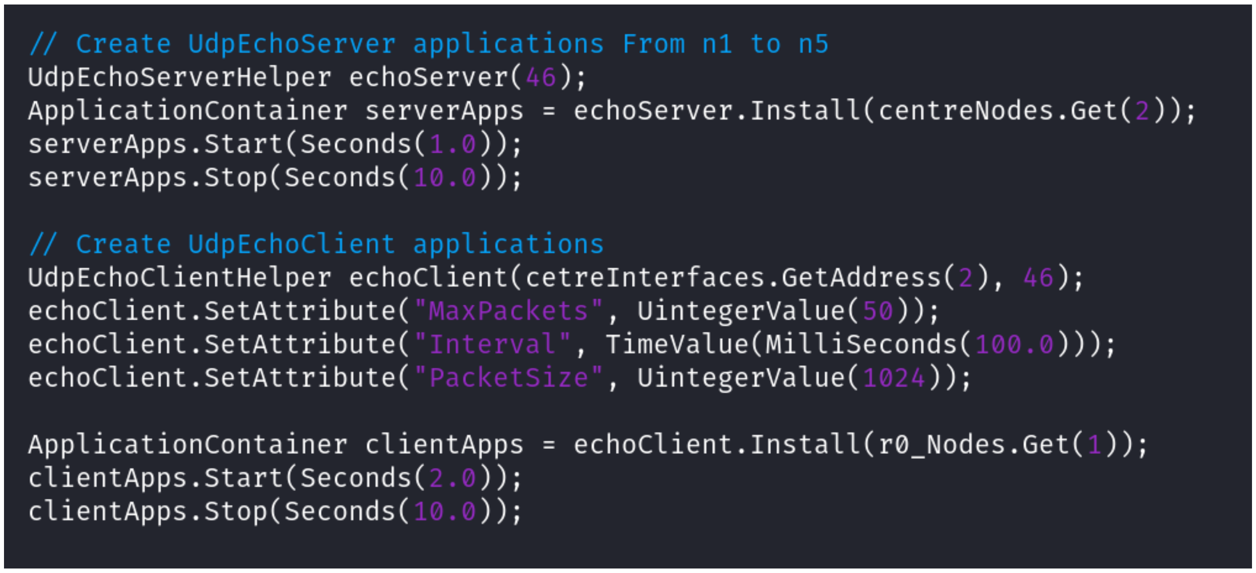


Figure 17: Flow vs Packet Loss Graph

We are not getting any column in the above graph because we can see that there is no packet loss in all the 7 flows. This can also be verified from the terminal output.

Part B

In this question we have to introduce an error model and replace all the flows which were present in the PART-A with udpEchoClient and udpEchoServer flows.



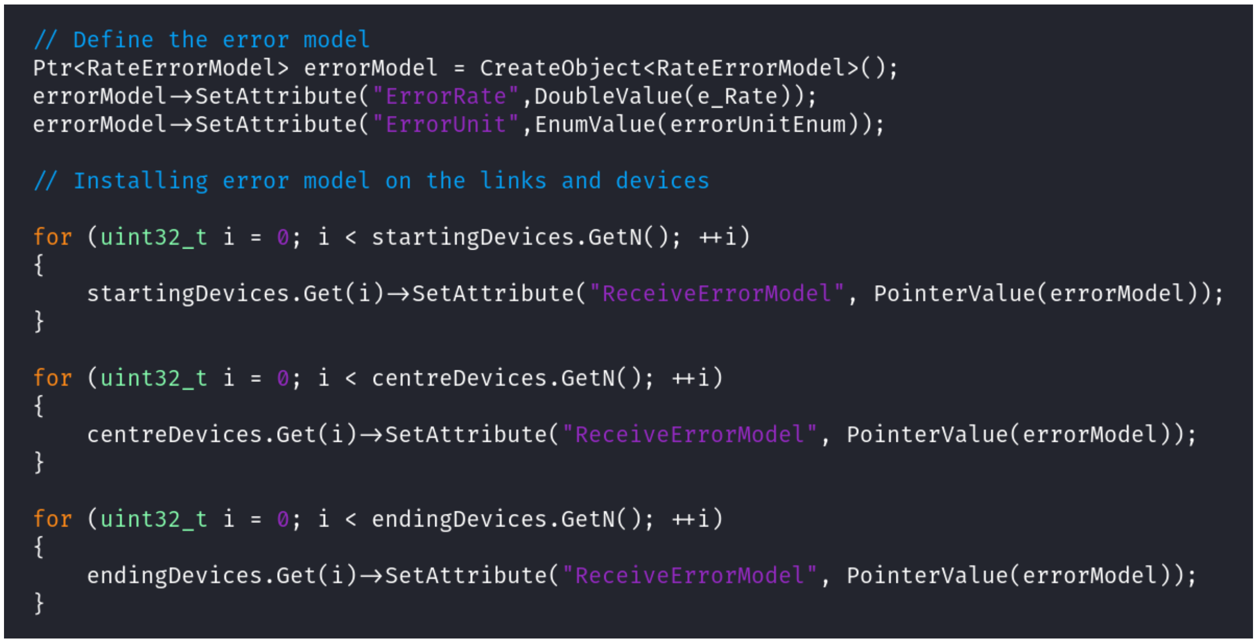
```
// Create UdpEchoServer applications From n1 to n5
UdpEchoServerHelper echoServer(46);
ApplicationContainer serverApps = echoServer.Install(centreNodes.Get(2));
serverApps.Start(Seconds(1.0));
serverApps.Stop(Seconds(10.0));

// Create UdpEchoClient applications
UdpEchoClientHelper echoClient(cetreInterfaces.GetAddress(2), 46);
echoClient.SetAttribute("MaxPackets", UintegerValue(50));
echoClient.SetAttribute("Interval", TimeValue(MilliSeconds(100.0)));
echoClient.SetAttribute("PacketSize", UintegerValue(1024));

ApplicationContainer clientApps = echoClient.Install(r0_Nodes.Get(1));
clientApps.Start(Seconds(2.0));
clientApps.Stop(Seconds(10.0));
```

Figure 18: Setting up UdpEchoServer and UdpEchoClient

The above figure shows how we defined the flow of the udpEchoServer and udpEchoClient for one flow. For the remaining flows we used the same method.

Now, let's take a look at the inclusion of an error model in the code.

We defined RateErrorModel in the code and applied it on all the nodes except routers which can be shown in the below image.



```
// Define the error model
Ptr<RateErrorModel> errorModel = CreateObject<RateErrorModel>();
errorModel->SetAttribute("ErrorRate",DoubleValue(e_Rate));
errorModel->SetAttribute("ErrorUnit",EnumValue(errorUnitEnum));

// Installing error model on the links and devices

for (uint32_t i = 0; i < startingDevices.GetN(); ++i)
{
    startingDevices.Get(i)->SetAttribute("ReceiveErrorModel", PointerValue(errorModel));
}

for (uint32_t i = 0; i < centreDevices.GetN(); ++i)
{
    centreDevices.Get(i)->SetAttribute("ReceiveErrorModel", PointerValue(errorModel));
}

for (uint32_t i = 0; i < endingDevices.GetN(); ++i)
{
    endingDevices.Get(i)->SetAttribute("ReceiveErrorModel", PointerValue(errorModel));
}
```

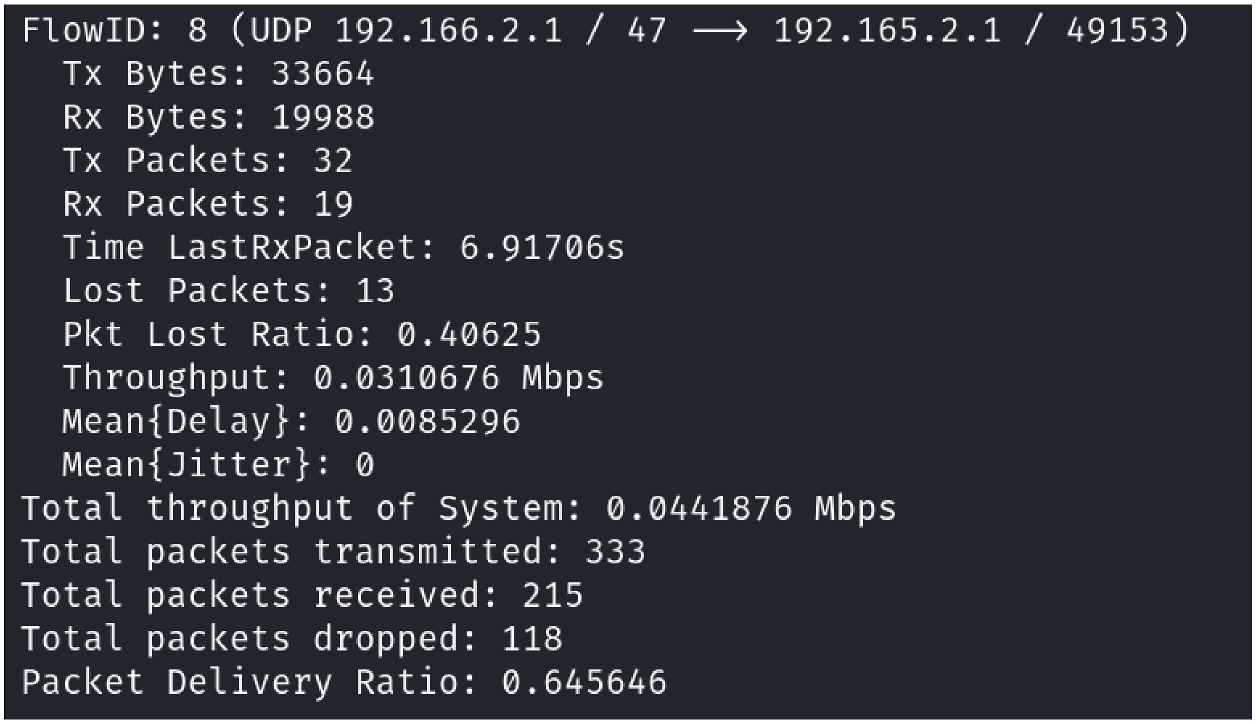
Figure 19: Defining Error Model

We are taking error_rate , error_unit and seed value from the user via command line argument. Hence you can run the 2_B.cc file susing below command,

./ns3 run (file path) – –RngRun=(Seed_value) –errorRate=(error_rate_value) –errorUnit=(error_unit)

Note: I had to pass one extra pair of double hyphens(–) from the Command line to correctly pass Arguments.If it creates any error, kindly remove that pair and then run the code again from command line.

The above command will show the statistics for all the flows corresponding to those particular chosen values passed into arguments and show the calculated **PDR(Packet Delivery Ratio)** on the terminal which is shown in the image below.



```
FlowID: 8 (UDP 192.166.2.1 / 47 ⟶ 192.165.2.1 / 49153)
  Tx Bytes: 33664
  Rx Bytes: 19988
  Tx Packets: 32
  Rx Packets: 19
  Time LastRxPacket: 6.91706s
  Lost Packets: 13
  Pkt Lost Ratio: 0.40625
  Throughput: 0.0310676 Mbps
  Mean{Delay}: 0.0085296
  Mean{Jitter}: 0
Total throughput of System: 0.0441876 Mbps
Total packets transmitted: 333
Total packets received: 215
Total packets dropped: 118
Packet Delivery Ratio: 0.645646
```

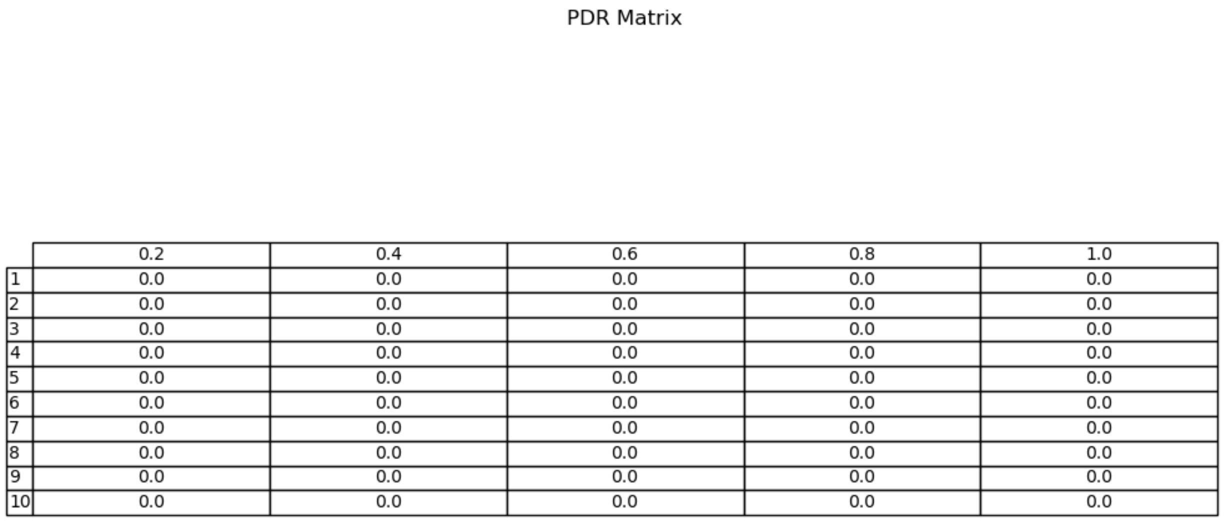
Figure 20: Output of the code on terminal

Now, to automate the task we created one python file which gives 10 consecutive numbers for RngRun value and 5 different values of error rates. The only manual task was to manually change the error unit in the file along with the path to the code file.

The script printed a table for the PDR value for each pair of RngRun values and error rate for the given error unit on the terminal and using gnuplot both.

First we executed python script for error units bit and bytes, But in both the cases, we got the same outputs for the PDR values and Average PDR vs Error Rate graph.

The PDR matrix created using python script is shown below for error unit bit and byte.

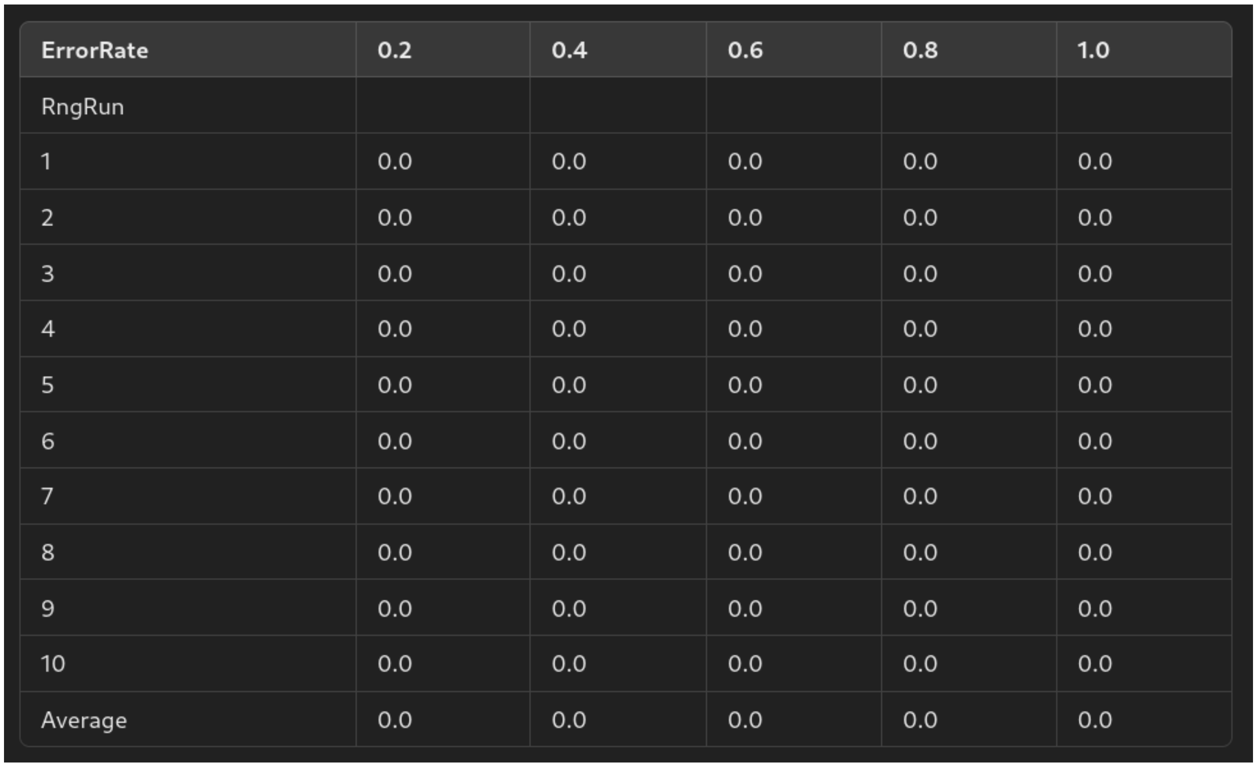

PDR Matrix

	0.2	0.4	0.6	0.8	1.0
1	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0

Figure 21: Table generated by python script

In the above image column numbers show the RngRun values and row headings indicate different error rates (Since my ID number ends with 00, I have chosen 1 to start the RngRun value).

Table with average PDR values calculated in it is below.



ErrorRate	0.2	0.4	0.6	0.8	1.0
RngRun					
1	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0
Average	0.0	0.0	0.0	0.0	0.0

Figure 22: Table of PDR values with error unit bit or byte

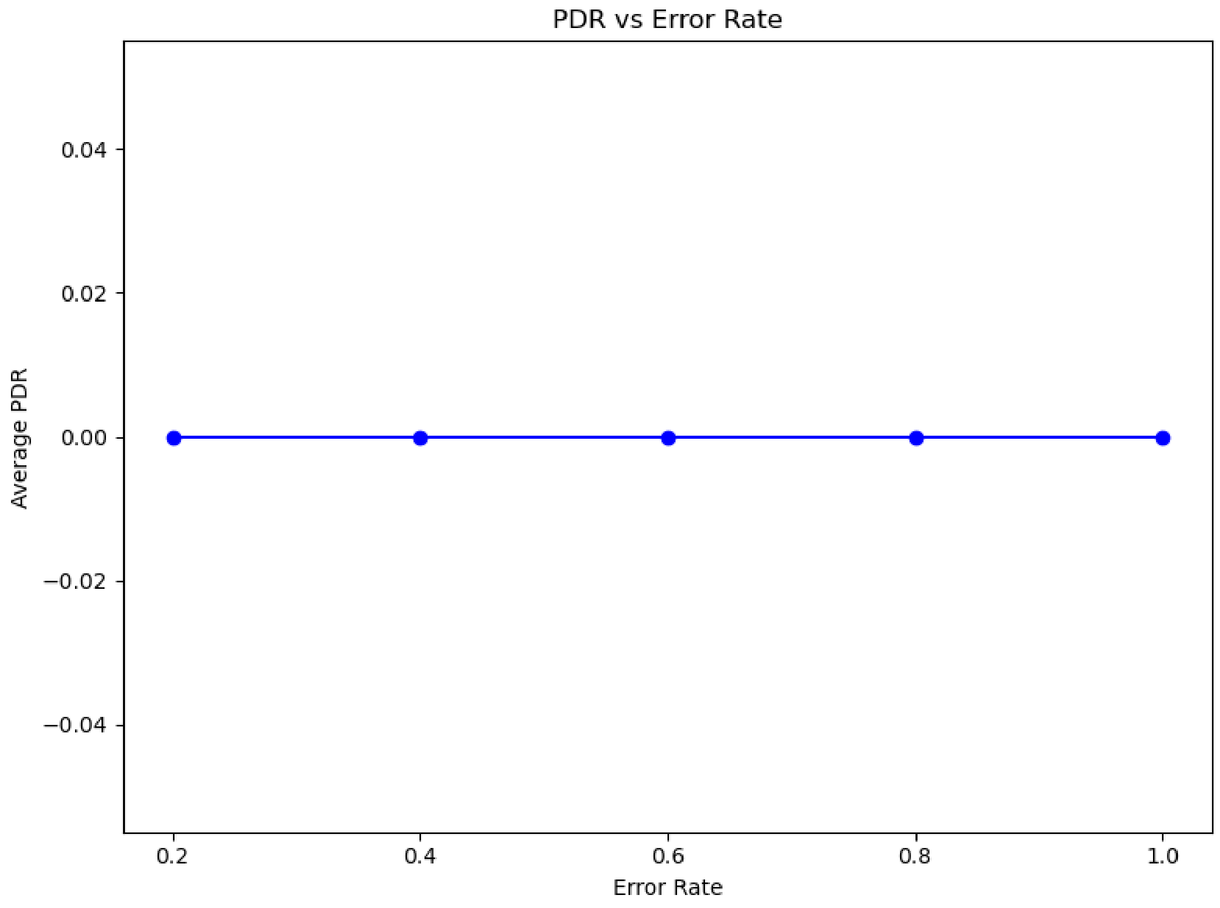
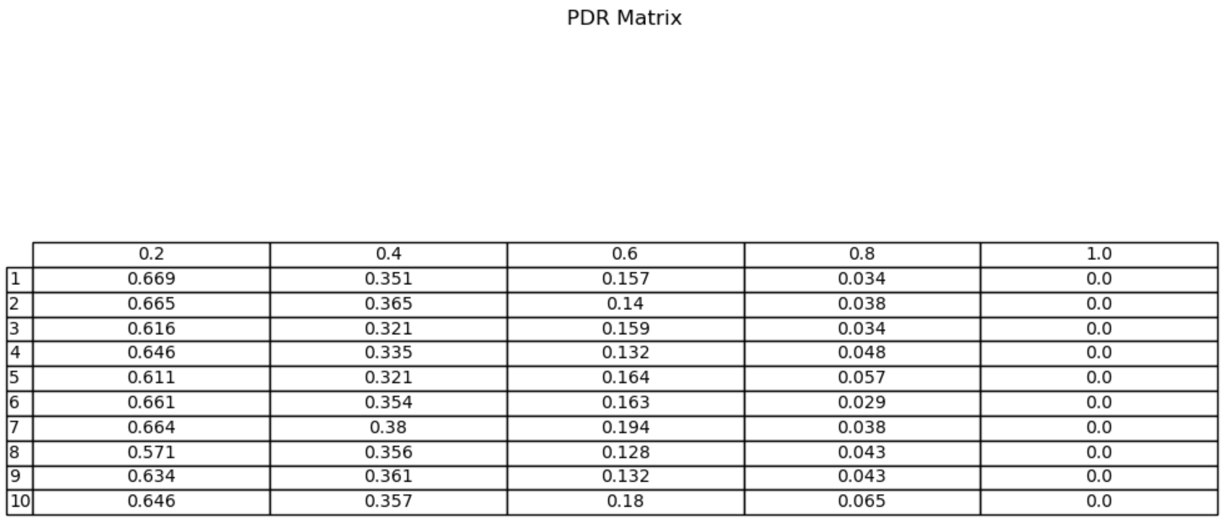


Figure 23: PDR vs Error Rate graph

This same output was shown for the error units bit and bytes both.

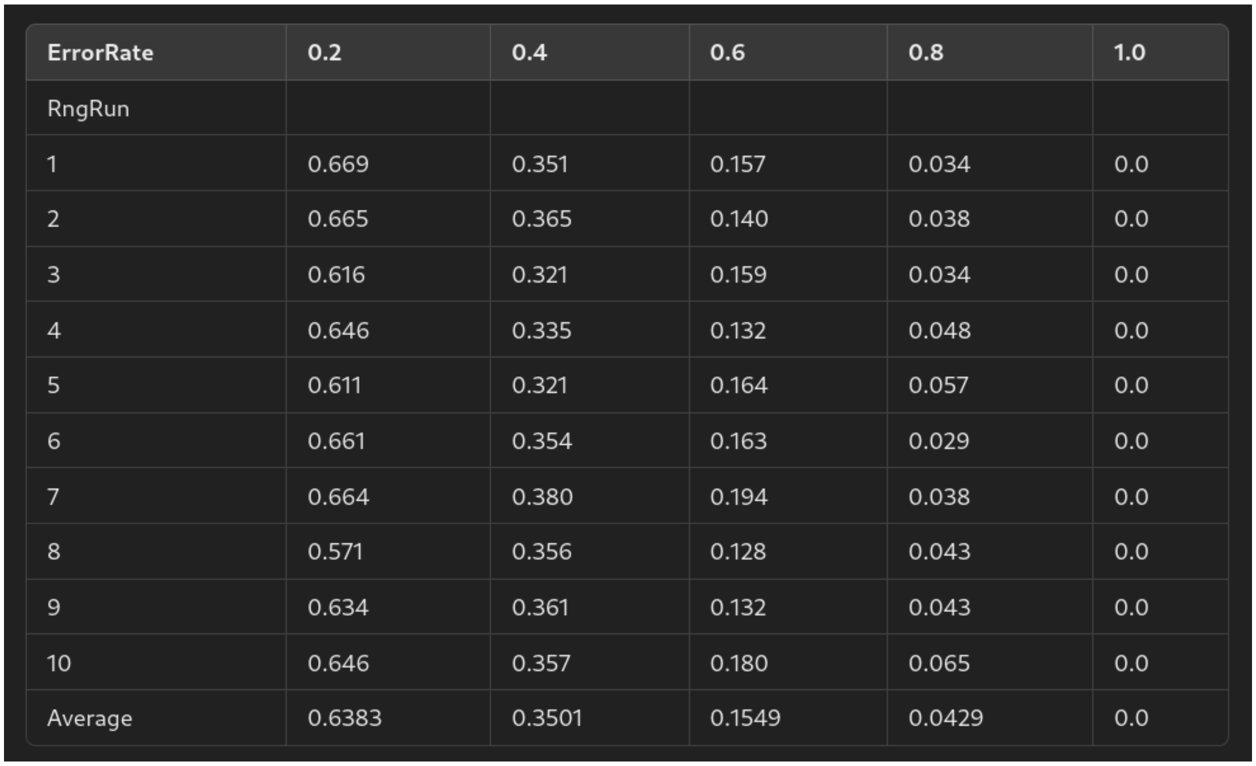
The PDR matrix for the error unit packet created by python script is shown below.



PDR Matrix

	0.2	0.4	0.6	0.8	1.0
1	0.669	0.351	0.157	0.034	0.0
2	0.665	0.365	0.14	0.038	0.0
3	0.616	0.321	0.159	0.034	0.0
4	0.646	0.335	0.132	0.048	0.0
5	0.611	0.321	0.164	0.057	0.0
6	0.661	0.354	0.163	0.029	0.0
7	0.664	0.38	0.194	0.038	0.0
8	0.571	0.356	0.128	0.043	0.0
9	0.634	0.361	0.132	0.043	0.0
10	0.646	0.357	0.18	0.065	0.0

Figure 24: PDR value table generated by python script



ErrorRate	0.2	0.4	0.6	0.8	1.0
RngRun					
1	0.669	0.351	0.157	0.034	0.0
2	0.665	0.365	0.140	0.038	0.0
3	0.616	0.321	0.159	0.034	0.0
4	0.646	0.335	0.132	0.048	0.0
5	0.611	0.321	0.164	0.057	0.0
6	0.661	0.354	0.163	0.029	0.0
7	0.664	0.380	0.194	0.038	0.0
8	0.571	0.356	0.128	0.043	0.0
9	0.634	0.361	0.132	0.043	0.0
10	0.646	0.357	0.180	0.065	0.0
Average	0.6383	0.3501	0.1549	0.0429	0.0

Figure 25: PDR value table for error unit - packet

PDR Matrix with average calculation is shown in the above image.

We can see the change in the PDR values for the error unit packet.

Now, The graph for the PDR vs Error Rate is shown below:

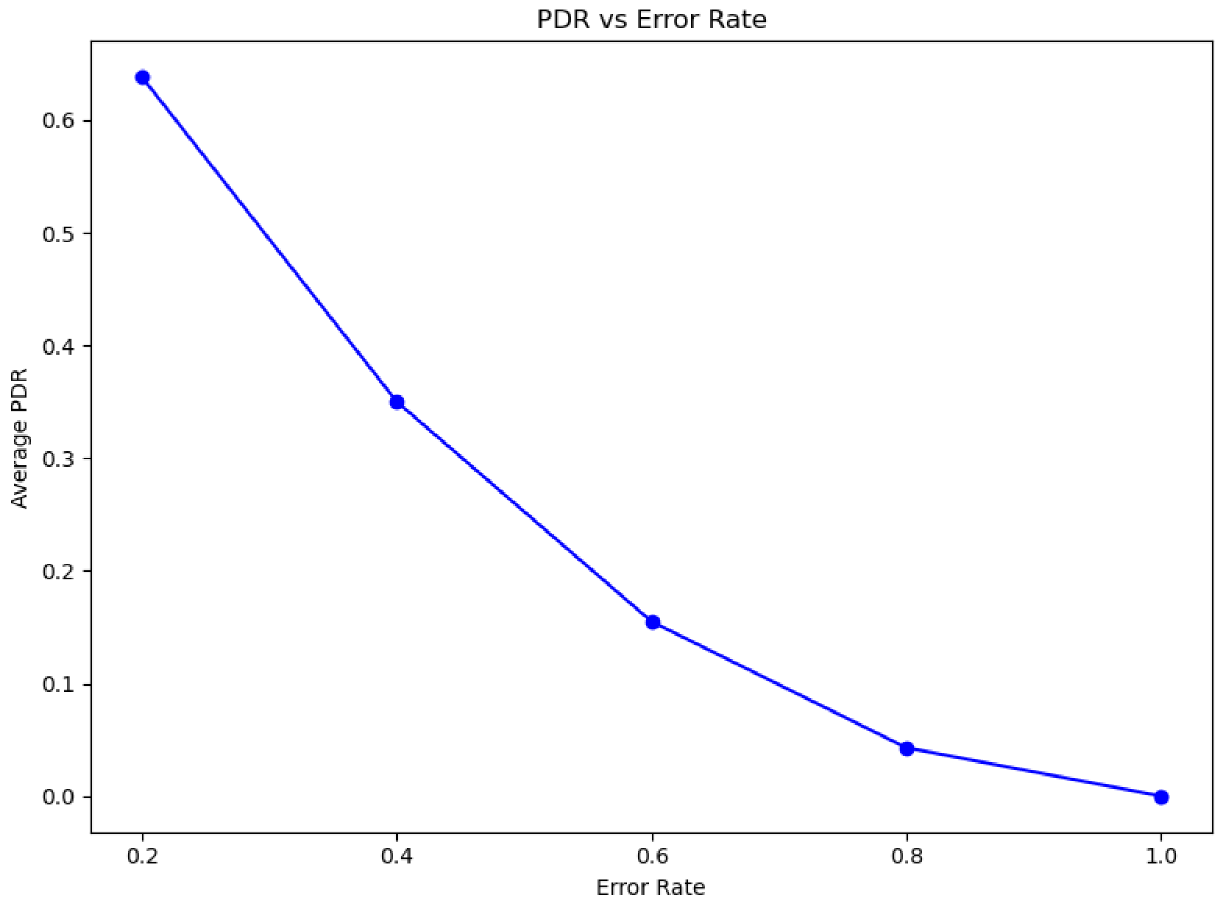


Figure 26: PDR vs Error Rate graph for error unit = packet

The above graph shows that increase in error rate decreases the Average Packet Delivery Ratio (PDR) in the Network.