PROJECT REPORT

EE-353 COMPUTER NETWORK

NAME	CMS ID
Jaweria Manahil	**419118**
Sameed Ilyas	**426125**
Sara Adnan	**411228**
Zainab Athar	**405094**

CLASS + SECTION: BESE 13 B

INSTRUCTOR: DR. HUMA GHAFOOR

Contents

1. Library Imports 3
2. Seed and logging setup: 4
3. Define Number of packets in each burst: 4
4. Creation of a Graph and Its Nodes 5
5.Mobility Configuration: 5
6. WIFI Configuration: 6
7. AODV Routing Protocol: 7
8. IP Address assignment: 7
9. Client UDP Setup: 8
10. Server (Packetsink) setup: 8
11. Animation setup: 9
12. Loop to Perform the Experiment 20 Times 9
13. Flow Moniter: 9
14. Calculate and print total averages 11
15. Complete code: 12
16. Output Screenshots: 16
17. Other Attempts: 25

1. Library Imports

These lines include necessary C++ standard and ns-3 libraries for simulation and networking functionalities. The ns-3 modules are crucial for handling network-related operations in the simulation.

NOTE: This code was written in ns-3.40, it has some slight changes as compared to the previous versions, e.g. it uses ./ns3 run command rather than ./waf to run a script.

1. `#include <iostream>`: Input/output stream library for basic console I/O operations.

2. `#include <fstream>`: File stream library for reading and writing files.

3. `#include <string>`: String manipulation library for handling strings.

4. `#include <cassert>`: Assertion library for debugging purposes.

5. `#include <cstdlib>`: C Standard Library for general-purpose functions.

6. `#include "ns3/core-module.h"`: Core module of ns-3 for fundamental simulation functionalities.

7. `#include "ns3/network-module.h"`: Network module of ns-3 for networking-related functionalities.

8. `#include "ns3/internet-module.h"`: Internet module of ns-3 for internet protocol support.

9. `#include "ns3/applications-module.h"`: Applications module of ns-3 for application layer functionalities.

10. `#include "ns3/aodv-helper.h"`: AODV (Ad hoc On-Demand Distance Vector) helper of ns-3 for AODV routing.

11. `#include "ns3/global-route-manager.h"`: Global route manager of ns-3 for managing global routes.

12. `#include "ns3/mobility-module.h"`: Mobility module of ns-3 for node mobility modeling.

13. `#include "ns3/netanim-module.h"`: NetAnim module of ns-3 for network animation support.

14. `#include "ns3/assert.h"`: Assertion library specific to ns-3 for debugging, was used during testing.

15. `#include "ns3/wifi-module.h"`: Wi-Fi module of ns-3 for wireless communication.

16. `#include <ns3/address.h>`: Address library for ns-3 address manipulation.

2. Seed and logging setup:

```
   LogComponentEnable("ProjectScenario5", LOG_LEVEL_ALL);

double totalAverageDelay = 0.0;

   double totalPdr = 0.0;

    int successfullExp = 0, flowCounter = 0;

   int numExperiments = 20;

    for (int experiment = 0; experiment < numExperiments; ++experiment) {

   RngSeedManager::SetSeed(experiment + 1);

   std::cout << "Iteration Number " << experiment + 1 << " :" <<std::endl;
```

1. `LogComponentEnable("ProjectScenario5", LOG_LEVEL_ALL);`: Enables logging for the component "ProjectScenario5" with all log levels enabled.

2. `double totalAverageDelay = 0.0;`: Initializes a variable for accumulating average delay with an initial value of 0.0.

3. `double totalPdr = 0.0;`: Initializes a variable for accumulating Packet Delivery Ratio (PDR) with an initial value of 0.0.

4. `int successfullExp = 0;`: Initializes a counter for successful experiments with an initial value of 0.

5. 'int flowCounter = 0;': Initializes a counter for number of flows.

6. `int numExperiments = 20;`: Sets the total number of experiments to 20.

7. `for (int experiment = 0; experiment < numExperiments; ++experiment) {`: Begins a loop for running 20 experiments.

8. `RngSeedManager::SetSeed(experiment + 1);`: Sets the random number generator seed for each experiment.

9. `std::cout << "Iteration Number " << experiment + 1 << " :" <<std::endl;`: Prints the current iteration number to the console.

3. Define Number of packets in each burst:

```
const uint32_t numPackets = 5;

const uint32_t packetSize = 1024; // Size of each packet

const DataRate dataRate("500kb/s");
```

This code snippet sets up parameters for a network simulation and creates a container of nodes. Here's a brief explanation of each part:

1. `const uint32_t numPackets = 5;`: Defines the number of packets to be sent in each burst as 5.

2. `const uint32_t packetSize = 1024;`: Defines the size of each packet as 1024 bytes (or 1 kilobyte).

3. `const DataRate dataRate("500kb/s");`: Defines the data rate for transmissions as 500 kilobits per second.

In summary, this code snippet initializes parameters such as the number of packets, packet size, and data rate for a network simulation.

4. Creation of a Graph and Its Nodes

```
NodeContainer nodes;

nodes.Create(NODES);
```

A NodeContainer object named nodes is instantiated. This container is a part of ns-3 and is used to manage and group nodes in the simulation.The Create method of the NodeContainer is called with the argument NODES. This method generates and adds a specified number of nodes to the container. The constant NODES represents the number of nodes to be created in the simulation.The number of nodes is defined by the NODES constant (50 in this case).

5.Mobility Configuration:

```
MobilityHelper mobility;

mobility.SetPositionAllocator("ns3::RandomDiscPositionAllocator",

                    "X", StringValue("300.0"),

                    "Y", StringValue("300.0"),

                    "Rho", StringValue("ns3::UniformRandomVariable[Min=0|Max=150]"));

mobility.SetMobilityModel("ns3::RandomWalk2dMobilityModel",

                  "Bounds", RectangleValue(Rectangle(0, 500, 0, 500)),

                  "Distance", DoubleValue(5.0),

                  "Time", TimeValue(Seconds(1.0)));

mobility.Install(nodes);
```

This code snippet configures mobility for network nodes in a simulation using the NS-3 library. Here's a breakdown:

1. `MobilityHelper mobility;`: Creates a helper object for configuring node mobility.

2. `mobility.SetPositionAllocator("ns3::RandomDiscPositionAllocator", ...);`: Configures the position allocator to randomly allocate positions within a disc-shaped area. The parameters specify the center (`X` and `Y`) and the radius (`Rho`) of the disc.

3. `mobility.SetMobilityModel("ns3::RandomWalk2dMobilityModel", ...);`: Configures the mobility model for nodes to follow a 2D random walk within specified bounds. The parameters define the rectangular bounds within which nodes can move, the distance traveled in each step, and the time interval between steps.

4. `mobility.Install(nodes);`: Installs the configured mobility model on the nodes in the `nodes` container, enabling them to move according to the specified random walk pattern within the defined bounds.

In summary, this code snippet sets up a random disc position allocator and a 2D random walk mobility model for the network nodes, allowing them to move randomly within a specified area during the simulation.

6. WIFI Configuration:

```
WifiHelper wifi;

YansWifiPhyHelper wifiPhy;


YansWifiChannelHelper wifiChannel = YansWifiChannelHelper::Default();

wifiPhy.SetChannel(wifiChannel.Create());


WifiMacHelper wifiMac;

wifiMac.SetType("ns3::AdhocWifiMac");

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

1. `WifiHelper wifi;`: Creates a helper object for managing Wi-Fi configurations.

2. `YansWifiPhyHelper wifiPhy;`: Creates a helper object for Wi-Fi physical layer configurations.

3. `YansWifiChannelHelper wifiChannel = YansWifiChannelHelper::Default();`: Sets up default Wi-Fi channel properties.

4. `wifiPhy.SetChannel(wifiChannel.Create());`: Associates the Wi-Fi physical layer with the created channel.

5. `WifiMacHelper wifiMac;`: Creates a helper object for Wi-Fi MAC layer configurations.

6. `wifiMac.SetType("ns3::AdhocWifiMac");`: Specifies the type of MAC layer to be used as ad-hoc.

7. `NetDeviceContainer wifiDevices = wifi.Install(wifiPhy, wifiMac, nodes);`: Installs Wi-Fi devices on specified nodes using the configured PHY and MAC settings.

7. AODV Routing Protocol:

```
AodvHelper aodv;
  InternetStackHelper stack;
  stack.SetRoutingHelper(aodv);
  stack.Install(nodes);
```

1. `AodvHelper aodv;`: Creates a helper object for the Ad-hoc On-demand Distance Vector (AODV) routing protocol.

2. `InternetStackHelper stack;`: Creates a helper object for managing Internet protocol stack configurations.

3. `stack.SetRoutingHelper(aodv);`: Configures the routing helper to use AODV, instead of OLSR routing, the reason will be mentioned later in the report.

4. `stack.Install(nodes);`: Installs the Internet protocol stack with AODV routing on the specified nodes.

8. IP Address assignment:

```
  Ipv4AddressHelper address;
  address.SetBase("10.0.0.0", "255.255.255.0");
  Ipv4InterfaceContainer interfaces = address.Assign(wifiDevices);
```

This code snippet configures IP address assignment for network interfaces in a simulation using the NS-3 library. Here's a breakdown:

1. `Ipv4AddressHelper address;`: Creates a helper object for configuring IPv4 addresses.

2. `address.SetBase("10.0.0.0", "255.255.255.0");`: Sets the base IPv4 address and subnet mask for address assignment. In this example, the base address is set to "10.0.0.0" with a subnet mask of "255.255.255.0", defining a Class C private network.

3. `Ipv4InterfaceContainer interfaces = address.Assign(wifiDevices);`: Assigns IPv4 addresses to the network interfaces represented by the `wifiDevices` container using the configured address helper. The assigned addresses are stored in the `Ipv4InterfaceContainer` object named `interfaces`.

9. Client UDP Setup:

```
UdpClientHelper client(interfaces.GetAddress(49), port); // The address of the destination node

client.SetAttribute("MaxPackets", UintegerValue(numPackets));

client.SetAttribute("Interval", TimeValue(Seconds(1.0))); // Interval between packets

client.SetAttribute("PacketSize", UintegerValue(packetSize));


ApplicationContainer clientApps = client.Install(nodes.Get(0)); // Install client on the source node

clientApps.Start(Seconds(1.0));

clientApps.Stop(Seconds(ANIMATION_TIME));
```

1. `UdpClientHelper client(interfaces.GetAddress(49), port);`: Creates a UDP client helper with the destination address set to the address of the node at index 49 and using a specified port.

2. `client.SetAttribute("MaxPackets", UintegerValue(numPackets));`: Sets the maximum number of packets the client will send to `numPackets`.

3. `client.SetAttribute("Interval", TimeValue(Seconds(1.0)));`: Sets the time interval between sending consecutive packets to 1 second.

4. `client.SetAttribute("PacketSize", UintegerValue(packetSize));`: Sets the size of each packet to `packetSize`.

5. `ApplicationContainer clientApps = client.Install(nodes.Get(0));`: Installs the UDP client application on the first node (index 0) in the `nodes` container.

6. `clientApps.Start(Seconds(1.0));`: Starts the client application after a delay of 1 second.

7. `clientApps.Stop(Seconds(ANIMATION_TIME));`: Stops the client application after a specified animation time duration.

10. Server (Packetsink) setup:

```
UdpServerHelper server(port);

ApplicationContainer serverApps = server.Install(nodes.Get(49)); // Install server on the destination node

serverApps.Start(Seconds(1.0));

serverApps.Stop(Seconds(ANIMATION_TIME));
```

1. `UdpServerHelper server(port);`: Creates a UDP server helper configured to listen on the specified `port`.

2. `ApplicationContainer serverApps = server.Install(nodes.Get(49));`: Installs the UDP server application on the node at index 49 in the `nodes` container.

3. `serverApps.Start(Seconds(1.0));`: Starts the server application after a delay of 1 second.

4. `serverApps.Stop(Seconds(ANIMATION_TIME));`: Stops the server application after a specified animation time duration.

11. Animation setup:

```
std::string animFilename = "project-scenario5-" + std::to_string(experiment) + ".xml";

    AnimationInterface anim(animFilename.c_str());

    anim.EnablePacketMetadata(true); // Enable packet metadata for NetAnim

    anim.EnableIpv4L3ProtocolCounters(Seconds(0), Seconds(10));
```

This code generates a unique filename for the NetAnim animation interface based on the experiment number, initializes the animation interface using this filename, and configures it to enable packet metadata display. The animation interface is set to visualize IPv4 layer-3 protocol counters for the initial 10 seconds of the simulation. The filename is created by concatenating "project-scenario5-" with the experiment number converted to a string, ensuring each experiment generates a distinct XML file for NetAnim visualization.

12. Loop to Perform the Experiment 20 Times

```
Simulator::Stop(Seconds(ANIMATION_TIME));

Simulator::Run();
```

The simulation is executed using the ns-3 Simulator. This statement runs the simulation, allowing nodes to move, send and receive packets, and perform other defined actions.

13. Flow Moniter:

```
//FlowMonitor Setup

 FlowMonitorHelper flowmon;

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



//FlowMonitor Statistics Calculation

 Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier>(flowmon.GetClassifier());

std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats();
```

```
for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator i = stats.begin(); i != stats.end(); ++i) {
   Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow(i->first);
   std::cout << "Flow " << i->first << " (" << t.sourceAddress << " -> " << t.destinationAddress << ")\n";
   std::cout << "  Tx Packets: " << i->second.txPackets << "\n";
   std::cout << "  Rx Packets: " << i->second.rxPackets << "\n";
   flowCounter++;
   double throughput = i->second.rxBytes * 8.0 / (i->second.timeLastRxPacket.GetSeconds() - i->second.timeFirstTxPacket.GetSeconds()) / 1024 / 1024;
   std::cout << "  Throughput: " << throughput << " Mbps\n";

   if (i->second.rxPackets > 0) {
      double averageDelay = i->second.delaySum.GetSeconds() / i->second.rxPackets;
      double pdr = static_cast<double>(i->second.rxPackets) / i->second.txPackets;
      std::cout << "  Average Delay: " << averageDelay << " seconds\n";
      std::cout << "  Packet Delivery Ratio: " << pdr << "\n";
      totalAverageDelay += averageDelay;
      totalPdr += pdr;
      successfullExp++;

   } else {
      std::cout << "  Average Delay: N/A\n";
      std::cout << "  Packet Delivery Ratio: N/A\n";
   }
```

This code snippet is related to network simulation and uses the NS-3 library to monitor and analyze flows in a network. Here's a brief explanation of each part:

1. `FlowMonitorHelper flowmon;`: Creates a helper object for monitoring flows in the network.

2. `Ptr<FlowMonitor> monitor = flowmon.InstallAll();`: Installs the flow monitor on all network interfaces and returns a pointer to the monitor object.

3. `Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier>(flowmon.GetClassifier());`: Retrieves the classifier for IPv4 flows from the flow monitor helper.

4. `std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats();`: Retrieves statistics for all flows monitored by the flow monitor.

5. The subsequent `for` loop iterates through each flow, prints flow details such as source and destination addresses, and calculates and prints various metrics like throughput, average delay, and packet delivery ratio for each flow.

6. `totalAverageDelay += averageDelay;`: Accumulates the average delay for successful flows.

7. `totalPdr += pdr;`: Accumulates the packet delivery ratio for successful flows.

8. `successfullExp++;`: Increments the counter for successful experiments.

9.’ flowCounter++;’: Increments the counter for number of flows.

Overall, this code is used to monitor and analyze the performance of flows in a simulated network, computing metrics like throughput, average delay, and packet delivery ratio for each flow and accumulating these metrics for further analysis or reporting.

14. Calculate and print total averages

```
   double finalAverageDelay = totalAverageDelay / successfullExp;

   double finalPdr = totalPdr / flowCounter;

   std::cout << "Total Average Delay over " << numExperiments << " experiments: " << finalAverageDelay
<< " seconds" << std::endl;

   std::cout << "Total Packet Delivery Ratio over " << numExperiments << " experiments: " << finalPdr <<
std::endl;
```

1. `double finalAverageDelay = totalAverageDelay / successfullExp;`: Calculates the final average delay by dividing the accumulated total average delay (`totalAverageDelay`) by the number of successful experiments (`successfullExp`).

2. `double finalPdr = totalPdr / successfullExp;`: Calculates the final packet delivery ratio (PDR) by dividing the accumulated total PDR (`totalPdr`) by the flowCounter.

3. `std::cout << "Total Average Delay over " << numExperiments << " experiments: " << finalAverageDelay << " seconds" << std::endl;`: Prints the total average delay over all experiments along with the number of experiments and the unit of measurement (seconds).

4. `std::cout << "Total Packet Delivery Ratio over " << numExperiments << " experiments: " << finalPdr << std::endl;`: Prints the total packet delivery ratio over all experiments along with the number of experiments.

15. Complete code:

```
#include <iostream>
#include <fstream>
#include <string>
#include <cassert>
#include <cstdlib>

#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/applications-module.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/aodv-helper.h"
#include "ns3/global-route-manager.h"
#include "ns3/mobility-module.h"
#include "ns3/netanim-module.h"
#include "ns3/assert.h"
#include "ns3/wifi-module.h"
#include <ns3/ptr.h>
#include <ns3/nstime.h>
#include <ns3/packet.h>
#include <ns3/address.h>
#include "ns3/flow-monitor-module.h"

//MADE USING NS-3.40 some methods or classes will be implemented differently on
previous versions

#define NODES 50
#define ANIMATION_TIME 10

using namespace ns3;

NS_LOG_COMPONENT_DEFINE("ProjectScenario5");


int main(int argc, char *argv[])
{
    // Seed and logging setup
    LogComponentEnable("ProjectScenario5", LOG_LEVEL_ALL);

double totalAverageDelay = 0.0;
    double totalPdr = 0.0;
    int successfullExp = 0, flowCounter = 0;
    int numExperiments = 20;
```

```
     for (int experiment = 0; experiment < numExperiments; ++experiment) {


    RngSeedManager::SetSeed(experiment + 1);

    std::cout << "Iteration Number " << experiment + 1 << " :" <<std::endl;

    // Define the number of packets to send in each burst
const uint32_t numPackets = 5;
const uint32_t packetSize = 1024; // Size of each packet
const DataRate dataRate("500kb/s");

      // Create nodes

    NodeContainer nodes;
    nodes.Create(NODES);

       // Mobility configuration
    MobilityHelper mobility;
    mobility.SetPositionAllocator("ns3::RandomDiscPositionAllocator",
                                  "X", StringValue("300.0"),
                                  "Y", StringValue("300.0"),
                                  "Rho",
StringValue("ns3::UniformRandomVariable[Min=0|Max=150]"));
    mobility.SetMobilityModel("ns3::RandomWalk2dMobilityModel",
                              "Bounds", RectangleValue(Rectangle(0, 500, 0,
500)),
                              "Distance", DoubleValue(5.0),
                              "Time", TimeValue(Seconds(1.0)));
    mobility.Install(nodes);


// Wi-Fi configuration
WifiHelper wifi;
YansWifiPhyHelper wifiPhy;

YansWifiChannelHelper wifiChannel = YansWifiChannelHelper::Default();
wifiPhy.SetChannel(wifiChannel.Create());

WifiMacHelper wifiMac;
wifiMac.SetType("ns3::AdhocWifiMac");
NetDeviceContainer wifiDevices = wifi.Install(wifiPhy, wifiMac, nodes);
```

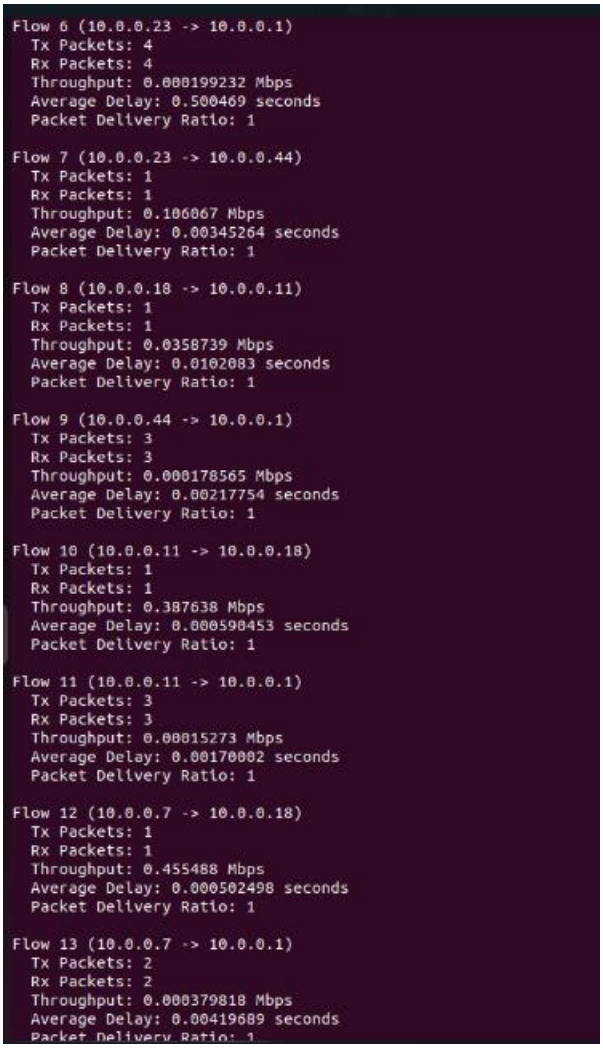
```
    //AODV Routing Protocol
    AodvHelper aodv;
    InternetStackHelper stack;
    stack.SetRoutingHelper(aodv);
    stack.Install(nodes);

            // IP address assignment
    Ipv4AddressHelper address;
    address.SetBase("10.0.0.0", "255.255.255.0");
    Ipv4InterfaceContainer interfaces = address.Assign(wifiDevices);


// Server (PacketSink) setup
uint16_t port = 9;  // Use any port number

// Client (UdpClient) setup
UdpClientHelper client(interfaces.GetAddress(49), port); // The address of the
destination node
client.SetAttribute("MaxPackets", UintegerValue(numPackets));
client.SetAttribute("Interval", TimeValue(Seconds(1.0))); // Interval between
packets
client.SetAttribute("PacketSize", UintegerValue(packetSize));

ApplicationContainer clientApps = client.Install(nodes.Get(0)); // Install client
on the source node
clientApps.Start(Seconds(1.0));
clientApps.Stop(Seconds(ANIMATION_TIME));


// Server (PacketSink) setup
UdpServerHelper server(port);
ApplicationContainer serverApps = server.Install(nodes.Get(49)); // Install
server on the destination node
serverApps.Start(Seconds(1.0));
serverApps.Stop(Seconds(ANIMATION_TIME));

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


    // Animation setup
    std::string animFilename = "project-scenario5-" + std::to_string(experiment)
+ ".xml";
    AnimationInterface anim(animFilename.c_str());
```

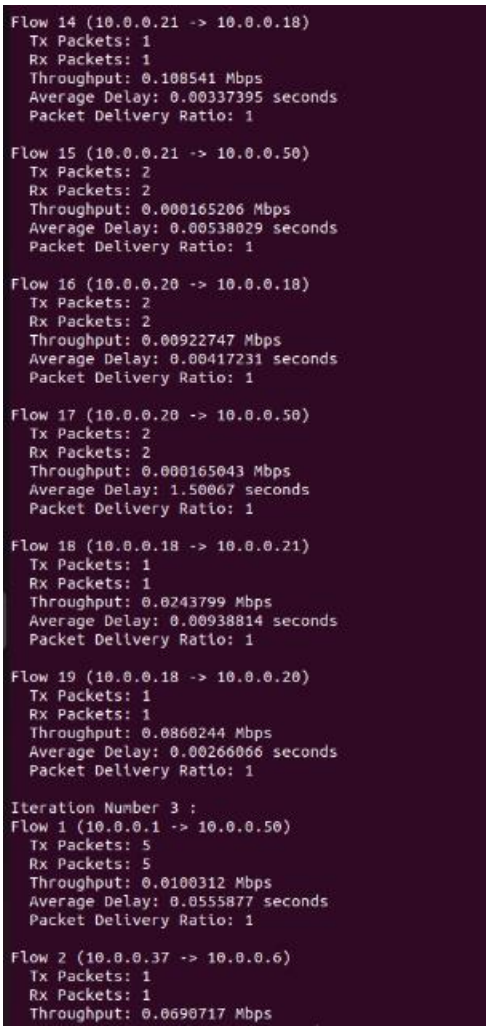
```
    anim.EnablePacketMetadata(true); // Enable packet metadata for NetAnim
    anim.EnableIpv4L3ProtocolCounters(Seconds(0), Seconds(10));


        // Run the simulation
        Simulator::Stop(Seconds(ANIMATION_TIME));
        Simulator::Run();

        //FlowMonitor Statistics Calculation
 Ptr<Ipv4FlowClassifier> classifier =
DynamicCast<Ipv4FlowClassifier>(flowmon.GetClassifier());
std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats();

for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator i = stats.begin();
i != stats.end(); ++i) {
    Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow(i->first);
    std::cout << "Flow " << i->first << " (" << t.sourceAddress << " -> " <<
t.destinationAddress << ")\n";
    std::cout << "  Tx Packets: " << i->second.txPackets << "\n";
    std::cout << "  Rx Packets: " << i->second.rxPackets << "\n";
    flowCounter++;
    double throughput = i->second.rxBytes * 8.0 / (i-
>second.timeLastRxPacket.GetSeconds() - i->second.timeFirstTxPacket.GetSeconds())
/ 1024 / 1024;
    std::cout << "  Throughput: " << throughput << " Mbps\n";

    if (i->second.rxPackets > 0) {
        double averageDelay = i->second.delaySum.GetSeconds() / i-
>second.rxPackets;
        double pdr = static_cast<double>(i->second.rxPackets) / i-
>second.txPackets;
        std::cout << "  Average Delay: " << averageDelay << " seconds\n";
        std::cout << "  Packet Delivery Ratio: " << pdr << "\n";
        totalAverageDelay += averageDelay;
        totalPdr += pdr;
        successfullExp++;

    } else {
        std::cout << "  Average Delay: N/A\n";
        std::cout << "  Packet Delivery Ratio: N/A\n";
    }

    std::cout << "\n";
}
```

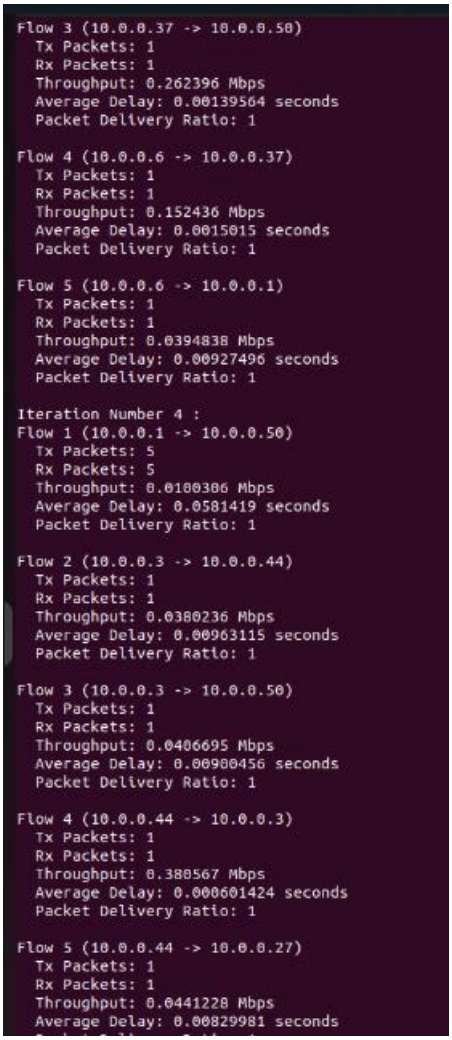
```
    Simulator::Destroy();
    }
 // Calculate and print total averages
    double finalAverageDelay = totalAverageDelay / successfullExp;
    double finalPdr = totalPdr / flowCounter;

    std::cout << "Total Average Delay over " << numExperiments << " experiments:
" << finalAverageDelay << " seconds" << std::endl;
    std::cout << "Total Packet Delivery Ratio over " << numExperiments << "
experiments: " << finalPdr << std::endl;
    return 0;
}
```

16. Output Screenshots:

```
[  0%] Building CXX object scratch/CMakeFiles/scratch_checking.dir/checking.cc.o
[  0%] Linking CXX executable ../../build/scratch/ns3.40-checking-default
Iteration Number 1 :
Flow 1 (10.0.0.1 -> 10.0.0.50)
  Tx Packets: 5
  Rx Packets: 0
  Throughput: -0 Mbps
  Average Delay: N/A
  Packet Delivery Ratio: N/A

Iteration Number 2 :
Flow 1 (10.0.0.1 -> 10.0.0.50)
  Tx Packets: 5
  Rx Packets: 4
  Throughput: 0.00797727 Mbps
  Average Delay: 0.371593 seconds
  Packet Delivery Ratio: 0.8

Flow 2 (10.0.0.18 -> 10.0.0.7)
  Tx Packets: 1
  Rx Packets: 1
  Throughput: 0.000365769 Mbps
  Average Delay: 1.00121 seconds
  Packet Delivery Ratio: 1

Flow 3 (10.0.0.18 -> 10.0.0.50)
  Tx Packets: 4
  Rx Packets: 4
  Throughput: 0.00153823 Mbps
  Average Delay: 0.000837241 seconds
  Packet Delivery Ratio: 1

Flow 4 (10.0.0.18 -> 10.0.0.23)
  Tx Packets: 3
  Rx Packets: 3
  Throughput: 0.000326137 Mbps
  Average Delay: 0.00239337 seconds
  Packet Delivery Ratio: 1

Flow 5 (10.0.0.23 -> 10.0.0.18)
  Tx Packets: 2
  Rx Packets: 2
  Throughput: 0.00108535 Mbps
  Average Delay: 0.00179578 seconds
  Packet Delivery Ratio: 1
```


Flow 6 (10.0.0.23 -> 10.0.0.1)
Tx Packets: 4
Rx Packets: 4
Throughput: 0.000199232 Mbps
Average Delay: 0.500469 seconds
Packet Delivery Ratio: 1
Flow 7 (10.0.0.23 -> 10.0.0.44)
Tx Packets: 1
Rx Packets: 1
Throughput: 0.106067 Mbps
Average Delay: 0.00345264 seconds
Packet Delivery Ratio: 1
Flow 8 (10.0.0.18 -> 10.0.0.11)
Tx Packets: 1
Rx Packets: 1
Throughput: 0.0358739 Mbps
Average Delay: 0.0102083 seconds
Packet Delivery Ratio: 1
Flow 9 (10.0.0.44 -> 10.0.0.1)
Tx Packets: 3
Rx Packets: 3
Throughput: 0.000178565 Mbps
Average Delay: 0.00217754 seconds
Packet Delivery Ratio: 1
Flow 10 (10.0.0.11 -> 10.0.0.18)
Tx Packets: 1
Rx Packets: 1
Throughput: 0.387638 Mbps
Average Delay: 0.000598453 seconds
Packet Delivery Ratio: 1
Flow 11 (10.0.0.11 -> 10.0.0.1)
Tx Packets: 3
Rx Packets: 3
Throughput: 0.00015273 Mbps
Average Delay: 0.00170002 seconds
Packet Delivery Ratio: 1
Flow 12 (10.0.0.7 -> 10.0.0.18)
Tx Packets: 1
Rx Packets: 1
Throughput: 0.455488 Mbps
Average Delay: 0.000502498 seconds
Packet Delivery Ratio: 1
Flow 13 (10.0.0.7 -> 10.0.0.1)
Tx Packets: 2
Rx Packets: 2
Throughput: 0.000379818 Mbps
Average Delay: 0.00419689 seconds
Packet Delivery Ratio: 1


Flow 14 (10.0.0.21 -> 10.0.0.18)
Tx Packets: 1
Rx Packets: 1
Throughput: 0.108541 Mbps
Average Delay: 0.00337395 seconds
Packet Delivery Ratio: 1
Flow 15 (10.0.0.21 -> 10.0.0.50)
Tx Packets: 2
Rx Packets: 2
Throughput: 0.000165206 Mbps
Average Delay: 0.00538029 seconds
Packet Delivery Ratio: 1
Flow 16 (10.0.0.20 -> 10.0.0.18)
Tx Packets: 2
Rx Packets: 2
Throughput: 0.00922747 Mbps
Average Delay: 0.00417231 seconds
Packet Delivery Ratio: 1
Flow 17 (10.0.0.20 -> 10.0.0.50)
Tx Packets: 2
Rx Packets: 2
Throughput: 0.000165043 Mbps
Average Delay: 1.50067 seconds
Packet Delivery Ratio: 1
Flow 18 (10.0.0.18 -> 10.0.0.21)
Tx Packets: 1
Rx Packets: 1
Throughput: 0.0243799 Mbps
Average Delay: 0.00938814 seconds
Packet Delivery Ratio: 1
Flow 19 (10.0.0.18 -> 10.0.0.20)
Tx Packets: 1
Rx Packets: 1
Throughput: 0.0860244 Mbps
Average Delay: 0.00266066 seconds
Packet Delivery Ratio: 1
Iteration Number 3 :
Flow 1 (10.0.0.1 -> 10.0.0.50)
Tx Packets: 5
Rx Packets: 5
Throughput: 0.0100312 Mbps
Average Delay: 0.0555877 seconds
Packet Delivery Ratio: 1
Flow 2 (10.0.0.37 -> 10.0.0.6)
Tx Packets: 1
Rx Packets: 1
Throughput: 0.0690717 Mbps

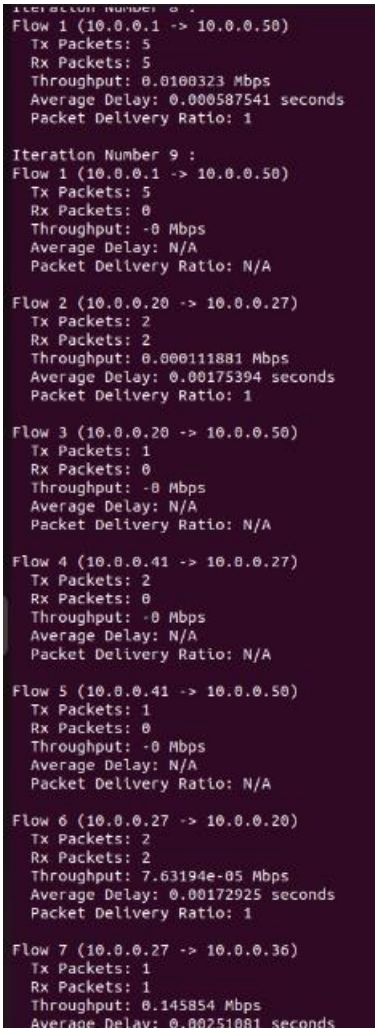

Flow 3 (10.0.0.37 -> 10.0.0.50)
Tx Packets: 1
Rx Packets: 1
Throughput: 0.262396 Mbps
Average Delay: 0.00139564 seconds
Packet Delivery Ratio: 1
Flow 4 (10.0.0.6 -> 10.0.0.37)
Tx Packets: 1
Rx Packets: 1
Throughput: 0.152436 Mbps
Average Delay: 0.0015015 seconds
Packet Delivery Ratio: 1
Flow 5 (10.0.0.6 -> 10.0.0.1)
Tx Packets: 1
Rx Packets: 1
Throughput: 0.0394838 Mbps
Average Delay: 0.00927496 seconds
Packet Delivery Ratio: 1
Iteration Number 4 :
Flow 1 (10.0.0.1 -> 10.0.0.50)
Tx Packets: 5
Rx Packets: 5
Throughput: 0.0100386 Mbps
Average Delay: 0.0581419 seconds
Packet Delivery Ratio: 1
Flow 2 (10.0.0.3 -> 10.0.0.44)
Tx Packets: 1
Rx Packets: 1
Throughput: 0.0380236 Mbps
Average Delay: 0.00963115 seconds
Packet Delivery Ratio: 1
Flow 3 (10.0.0.3 -> 10.0.0.50)
Tx Packets: 1
Rx Packets: 1
Throughput: 0.0406695 Mbps
Average Delay: 0.00900456 seconds
Packet Delivery Ratio: 1
Flow 4 (10.0.0.44 -> 10.0.0.3)
Tx Packets: 1
Rx Packets: 1
Throughput: 0.380567 Mbps
Average Delay: 0.000601424 seconds
Packet Delivery Ratio: 1
Flow 5 (10.0.0.44 -> 10.0.0.27)
Tx Packets: 1
Rx Packets: 1
Throughput: 0.0441228 Mbps
Average Delay: 0.00829981 seconds

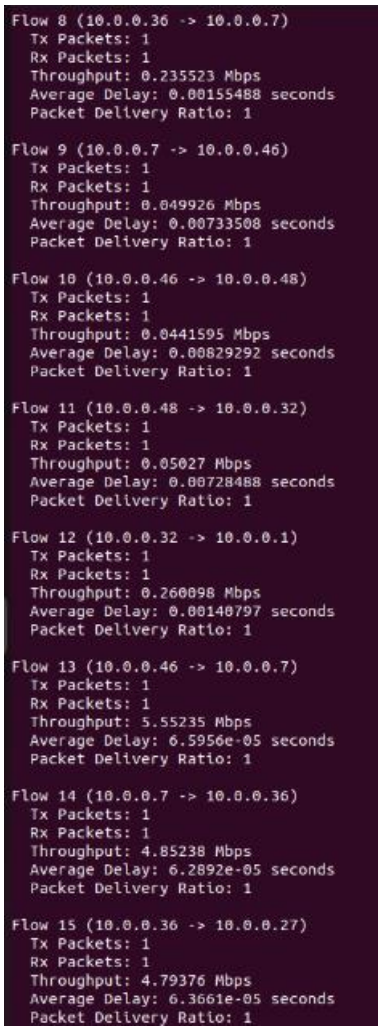
```
Flow 6 (10.0.0.27 -> 10.0.0.1)
  Tx Packets: 1
  Rx Packets: 1
  Throughput: 0.0350642 Mbps
  Average Delay: 0.010444 seconds
  Packet Delivery Ratio: 1

Iteration Number 5 :
Flow 1 (10.0.0.1 -> 10.0.0.50)
  Tx Packets: 5
  Rx Packets: 2
  Throughput: 0.00266867 Mbps
  Average Delay: 1.0187 seconds
  Packet Delivery Ratio: 0.4

Flow 2 (10.0.0.6 -> 10.0.0.1)
  Tx Packets: 1
  Rx Packets: 1
  Throughput: 0.0355169 Mbps
  Average Delay: 0.0103109 seconds
  Packet Delivery Ratio: 1

Flow 3 (10.0.0.6 -> 10.0.0.50)
  Tx Packets: 3
  Rx Packets: 3
  Throughput: 0.000182729 Mbps
  Average Delay: 0.00248563 seconds
  Packet Delivery Ratio: 1

Flow 4 (10.0.0.1 -> 10.0.0.6)
  Tx Packets: 1
  Rx Packets: 1
  Throughput: 0.153071 Mbps
  Average Delay: 0.00149526 seconds
  Packet Delivery Ratio: 1

Flow 5 (10.0.0.6 -> 10.0.0.7)
  Tx Packets: 1
  Rx Packets: 1
  Throughput: 0.000182835 Mbps
  Average Delay: 2.00296 seconds
  Packet Delivery Ratio: 1

Flow 6 (10.0.0.28 -> 10.0.0.7)
  Tx Packets: 1
  Rx Packets: 1
  Throughput: 0.0390463 Mbps
  Average Delay: 0.00937889 seconds
  Packet Delivery Ratio: 1

Flow 7 (10.0.0.28 -> 10.0.0.50)
  Tx Packets: 1
  Rx Packets: 1
  Throughput: 0.000182879 Mbps
  Average Delay: 2.00248 seconds
```

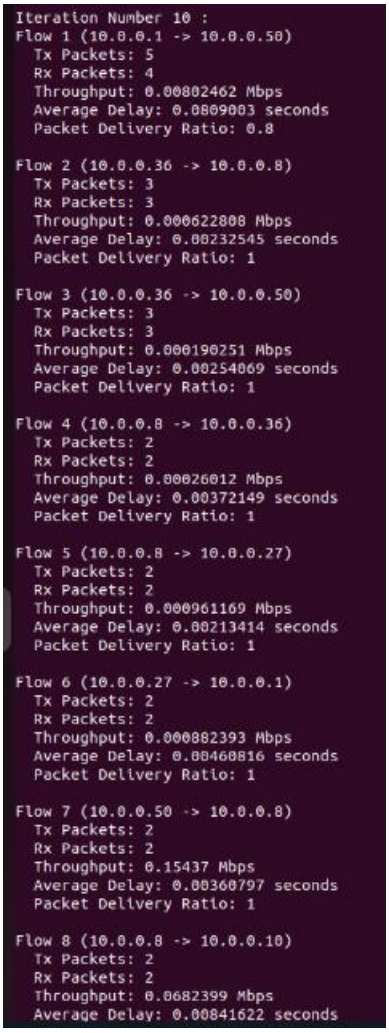
```
Flow 8 (10.0.0.10 -> 10.0.0.25)
  Tx Packets: 1
  Rx Packets: 0
  Throughput: -0 Mbps
  Average Delay: N/A
  Packet Delivery Ratio: N/A

Flow 9 (10.0.0.10 -> 10.0.0.50)
  Tx Packets: 2
  Rx Packets: 2
  Throughput: 0.000361731 Mbps
  Average Delay: 0.00517193 seconds
  Packet Delivery Ratio: 1

Flow 10 (10.0.0.7 -> 10.0.0.28)
  Tx Packets: 1
  Rx Packets: 1
  Throughput: 0.0250687 Mbps
  Average Delay: 0.0091302 seconds
  Packet Delivery Ratio: 1

Flow 11 (10.0.0.7 -> 10.0.0.47)
  Tx Packets: 2
  Rx Packets: 2
  Throughput: 0.000367352 Mbps
  Average Delay: 0.00437586 seconds
  Packet Delivery Ratio: 1

Flow 12 (10.0.0.47 -> 10.0.0.1)
  Tx Packets: 2
  Rx Packets: 2
  Throughput: 0.000368906 Mbps
  Average Delay: 0.00325172 seconds
  Packet Delivery Ratio: 1

Flow 13 (10.0.0.7 -> 10.0.0.6)
  Tx Packets: 1
  Rx Packets: 1
  Throughput: 0.0271187 Mbps
  Average Delay: 0.00843999 seconds
  Packet Delivery Ratio: 1

Iteration Number 6 :
Flow 1 (10.0.0.1 -> 10.0.0.50)
  Tx Packets: 5
  Rx Packets: 5
  Throughput: 0.0100315 Mbps
  Average Delay: 0.0539644 seconds
  Packet Delivery Ratio: 1

Flow 2 (10.0.0.19 -> 10.0.0.39)
  Tx Packets: 1
  Rx Packets: 1
  Throughput: 0.0442262 Mbps
  Average Delay: 0.0082804 seconds
```

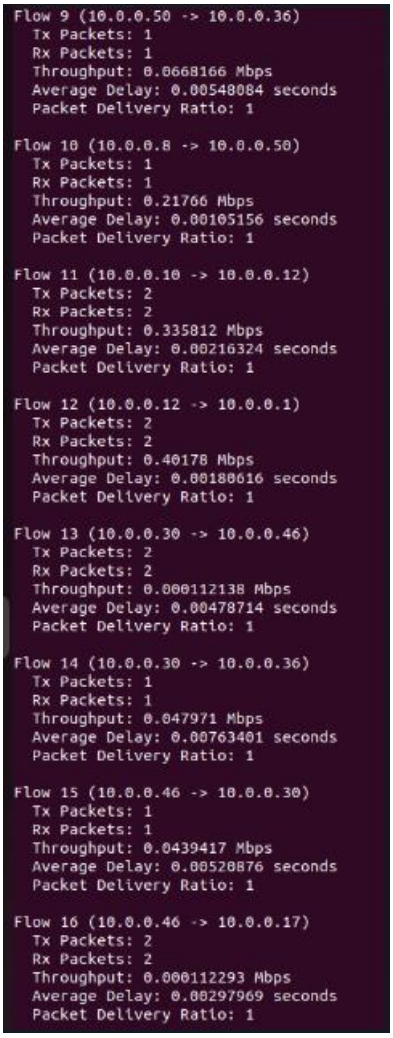
```
Flow 3 (10.0.0.19 -> 10.0.0.50)
  Tx Packets: 1
  Rx Packets: 1
  Throughput: 0.0571452 Mbps
  Average Delay: 0.00640843 seconds
  Packet Delivery Ratio: 1

Flow 4 (10.0.0.39 -> 10.0.0.19)
  Tx Packets: 1
  Rx Packets: 1
  Throughput: 0.166412 Mbps
  Average Delay: 0.00137539 seconds
  Packet Delivery Ratio: 1

Flow 5 (10.0.0.39 -> 10.0.0.1)
  Tx Packets: 1
  Rx Packets: 1
  Throughput: 0.0689943 Mbps
  Average Delay: 0.00530784 seconds
  Packet Delivery Ratio: 1

Iteration Number 7 :
Flow 1 (10.0.0.1 -> 10.0.0.50)
  Tx Packets: 5
  Rx Packets: 3
  Throughput: 0.00503465 Mbps
  Average Delay: 3.78238 seconds
  Packet Delivery Ratio: 0.6

Flow 2 (10.0.0.40 -> 10.0.0.32)
  Tx Packets: 2
  Rx Packets: 0
  Throughput: -0 Mbps
  Average Delay: N/A
  Packet Delivery Ratio: N/A

Flow 3 (10.0.0.40 -> 10.0.0.50)
  Tx Packets: 2
  Rx Packets: 2
  Throughput: 0.00103221 Mbps
  Average Delay: 0.00479974 seconds
  Packet Delivery Ratio: 1

Flow 4 (10.0.0.22 -> 10.0.0.32)
  Tx Packets: 2
  Rx Packets: 0
  Throughput: -0 Mbps
  Average Delay: N/A
  Packet Delivery Ratio: N/A

Flow 5 (10.0.0.22 -> 10.0.0.50)
  Tx Packets: 3
  Rx Packets: 3
  Throughput: 0.000283935 Mbps
  Average Delay: 0.000497928 seconds
```

```
Flow 6 (10.0.0.32 -> 10.0.0.37)
  Tx Packets: 1
  Rx Packets: 1
  Throughput: 0.0646045 Mbps
  Average Delay: 0.00566851 seconds
  Packet Delivery Ratio: 1

Flow 7 (10.0.0.32 -> 10.0.0.50)
  Tx Packets: 2
  Rx Packets: 2
  Throughput: 0.00022851 Mbps
  Average Delay: 0.00172832 seconds
  Packet Delivery Ratio: 1

Flow 8 (10.0.0.37 -> 10.0.0.32)
  Tx Packets: 1
  Rx Packets: 1
  Throughput: 0.0358879 Mbps
  Average Delay: 0.00637769 seconds
  Packet Delivery Ratio: 1

Flow 9 (10.0.0.37 -> 10.0.0.41)
  Tx Packets: 2
  Rx Packets: 2
  Throughput: 0.000182943 Mbps
  Average Delay: 0.00518832 seconds
  Packet Delivery Ratio: 1

Flow 10 (10.0.0.41 -> 10.0.0.18)
  Tx Packets: 2
  Rx Packets: 2
  Throughput: 0.000167793 Mbps
  Average Delay: 0.00467498 seconds
  Packet Delivery Ratio: 1

Flow 11 (10.0.0.18 -> 10.0.0.2)
  Tx Packets: 2
  Rx Packets: 2
  Throughput: 0.000167886 Mbps
  Average Delay: 0.00170803 seconds
  Packet Delivery Ratio: 1

Flow 12 (10.0.0.2 -> 10.0.0.1)
  Tx Packets: 2
  Rx Packets: 2
  Throughput: 0.00016802 Mbps
  Average Delay: 0.00323217 seconds
  Packet Delivery Ratio: 1

Iteration Number 8 :
Flow 1 (10.0.0.1 -> 10.0.0.50)
  Tx Packets: 5
  Rx Packets: 5
  Throughput: 0.0100323 Mbps
```


Flow 1 (10.0.0.1 -> 10.0.0.50)
Tx Packets: 5
Rx Packets: 5
Throughput: 0.0100323 Mbps
Average Delay: 0.000587541 seconds
Packet Delivery Ratio: 1
Iteration Number 9 :
Flow 1 (10.0.0.1 -> 10.0.0.50)
Tx Packets: 5
Rx Packets: 0
Throughput: -0 Mbps
Average Delay: N/A
Packet Delivery Ratio: N/A
Flow 2 (10.0.0.20 -> 10.0.0.27)
Tx Packets: 2
Rx Packets: 2
Throughput: 0.000111881 Mbps
Average Delay: 0.00175394 seconds
Packet Delivery Ratio: 1
Flow 3 (10.0.0.20 -> 10.0.0.50)
Tx Packets: 1
Rx Packets: 0
Throughput: -0 Mbps
Average Delay: N/A
Packet Delivery Ratio: N/A
Flow 4 (10.0.0.41 -> 10.0.0.27)
Tx Packets: 2
Rx Packets: 0
Throughput: -0 Mbps
Average Delay: N/A
Packet Delivery Ratio: N/A
Flow 5 (10.0.0.41 -> 10.0.0.50)
Tx Packets: 1
Rx Packets: 0
Throughput: -0 Mbps
Average Delay: N/A
Packet Delivery Ratio: N/A
Flow 6 (10.0.0.27 -> 10.0.0.20)
Tx Packets: 2
Rx Packets: 2
Throughput: 7.63194e-05 Mbps
Average Delay: 0.00172925 seconds
Packet Delivery Ratio: 1
Flow 7 (10.0.0.27 -> 10.0.0.36)
Tx Packets: 1
Rx Packets: 1
Throughput: 0.145854 Mbps

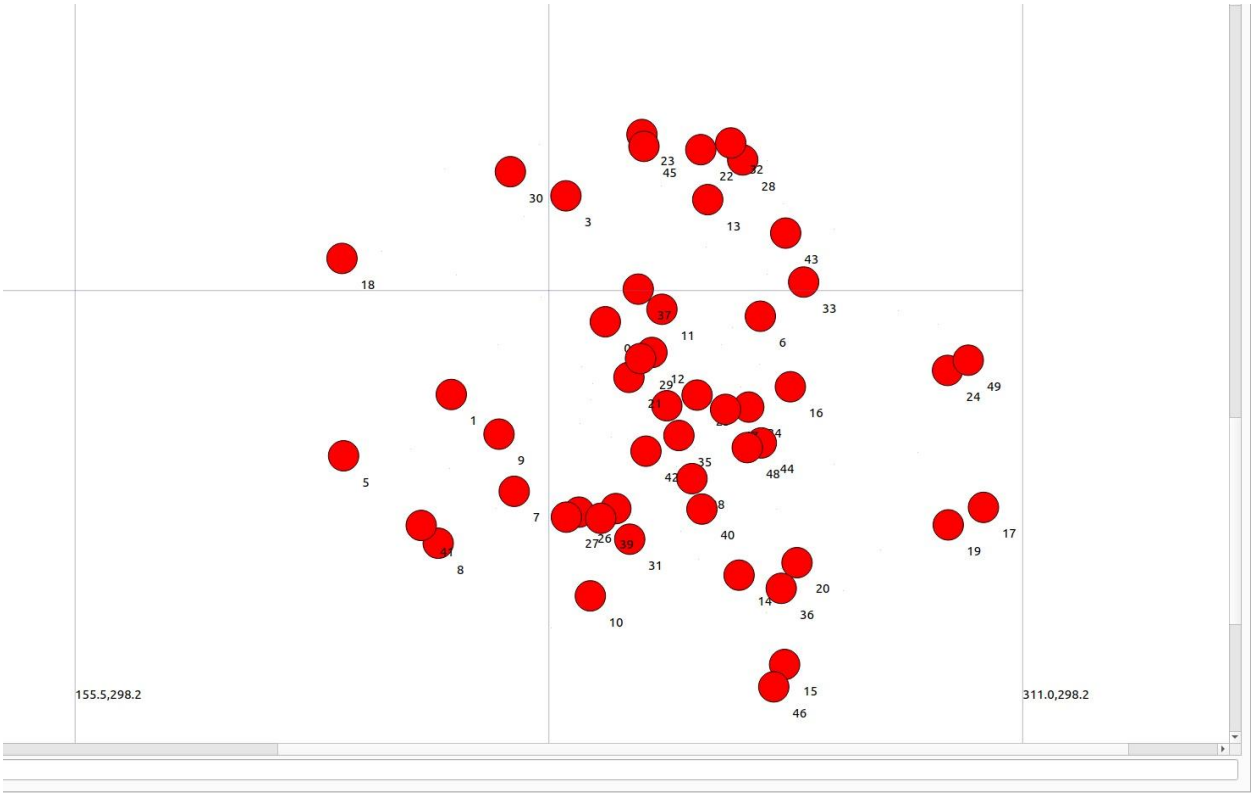

Flow 8 (10.0.0.36 -> 10.0.0.7)
Tx Packets: 1
Rx Packets: 1
Throughput: 0.235523 Mbps
Average Delay: 0.00155488 seconds
Packet Delivery Ratio: 1
Flow 9 (10.0.0.7 -> 10.0.0.46)
Tx Packets: 1
Rx Packets: 1
Throughput: 0.049926 Mbps
Average Delay: 0.00733508 seconds
Packet Delivery Ratio: 1
Flow 10 (10.0.0.46 -> 10.0.0.48)
Tx Packets: 1
Rx Packets: 1
Throughput: 0.0441595 Mbps
Average Delay: 0.00829292 seconds
Packet Delivery Ratio: 1
Flow 11 (10.0.0.48 -> 10.0.0.32)
Tx Packets: 1
Rx Packets: 1
Throughput: 0.05027 Mbps
Average Delay: 0.00728488 seconds
Packet Delivery Ratio: 1
Flow 12 (10.0.0.32 -> 10.0.0.1)
Tx Packets: 1
Rx Packets: 1
Throughput: 0.260098 Mbps
Average Delay: 0.00140797 seconds
Packet Delivery Ratio: 1
Flow 13 (10.0.0.46 -> 10.0.0.7)
Tx Packets: 1
Rx Packets: 1
Throughput: 5.55235 Mbps
Average Delay: 6.5956e-05 seconds
Packet Delivery Ratio: 1
Flow 14 (10.0.0.7 -> 10.0.0.36)
Tx Packets: 1
Rx Packets: 1
Throughput: 4.85238 Mbps
Average Delay: 6.2892e-05 seconds
Packet Delivery Ratio: 1
Flow 15 (10.0.0.36 -> 10.0.0.27)
Tx Packets: 1
Rx Packets: 1
Throughput: 4.79376 Mbps
Average Delay: 6.3661e-05 seconds
Packet Delivery Ratio: 1


Iteration Number 10 :
Flow 1 (10.0.0.1 -> 10.0.0.50)
Tx Packets: 5
Rx Packets: 4
Throughput: 0.00802462 Mbps
Average Delay: 0.0809003 seconds
Packet Delivery Ratio: 0.8
Flow 2 (10.0.0.36 -> 10.0.0.8)
Tx Packets: 3
Rx Packets: 3
Throughput: 0.000622808 Mbps
Average Delay: 0.00232545 seconds
Packet Delivery Ratio: 1
Flow 3 (10.0.0.36 -> 10.0.0.50)
Tx Packets: 3
Rx Packets: 3
Throughput: 0.000190251 Mbps
Average Delay: 0.00254069 seconds
Packet Delivery Ratio: 1
Flow 4 (10.0.0.8 -> 10.0.0.36)
Tx Packets: 2
Rx Packets: 2
Throughput: 0.00026012 Mbps
Average Delay: 0.00372149 seconds
Packet Delivery Ratio: 1
Flow 5 (10.0.0.8 -> 10.0.0.27)
Tx Packets: 2
Rx Packets: 2
Throughput: 0.000961169 Mbps
Average Delay: 0.00213414 seconds
Packet Delivery Ratio: 1
Flow 6 (10.0.0.27 -> 10.0.0.1)
Tx Packets: 2
Rx Packets: 2
Throughput: 0.000882393 Mbps
Average Delay: 0.00460816 seconds
Packet Delivery Ratio: 1
Flow 7 (10.0.0.50 -> 10.0.0.8)
Tx Packets: 2
Rx Packets: 2
Throughput: 0.15437 Mbps
Average Delay: 0.00360797 seconds
Packet Delivery Ratio: 1
Flow 8 (10.0.0.8 -> 10.0.0.10)
Tx Packets: 2
Rx Packets: 2
Throughput: 0.0682399 Mbps
Average Delay: 0.00841622 seconds


Flow 9 (10.0.0.50 -> 10.0.0.36)
Tx Packets: 1
Rx Packets: 1
Throughput: 0.0668166 Mbps
Average Delay: 0.00548084 seconds
Packet Delivery Ratio: 1
Flow 10 (10.0.0.8 -> 10.0.0.50)
Tx Packets: 1
Rx Packets: 1
Throughput: 0.21766 Mbps
Average Delay: 0.00105156 seconds
Packet Delivery Ratio: 1
Flow 11 (10.0.0.10 -> 10.0.0.12)
Tx Packets: 2
Rx Packets: 2
Throughput: 0.335812 Mbps
Average Delay: 0.00216324 seconds
Packet Delivery Ratio: 1
Flow 12 (10.0.0.12 -> 10.0.0.1)
Tx Packets: 2
Rx Packets: 2
Throughput: 0.40178 Mbps
Average Delay: 0.00180616 seconds
Packet Delivery Ratio: 1
Flow 13 (10.0.0.30 -> 10.0.0.46)
Tx Packets: 2
Rx Packets: 2
Throughput: 0.000112138 Mbps
Average Delay: 0.00478714 seconds
Packet Delivery Ratio: 1
Flow 14 (10.0.0.30 -> 10.0.0.36)
Tx Packets: 1
Rx Packets: 1
Throughput: 0.047971 Mbps
Average Delay: 0.00763401 seconds
Packet Delivery Ratio: 1
Flow 15 (10.0.0.46 -> 10.0.0.30)
Tx Packets: 1
Rx Packets: 1
Throughput: 0.0439417 Mbps
Average Delay: 0.00520876 seconds
Packet Delivery Ratio: 1
Flow 16 (10.0.0.46 -> 10.0.0.17)
Tx Packets: 2
Rx Packets: 2
Throughput: 0.000112293 Mbps
Average Delay: 0.00297969 seconds
Packet Delivery Ratio: 1

```
Flow 17 (10.0.0.17 -> 10.0.0.8)
  Tx Packets: 1
  Rx Packets: 1
  Throughput: 0.0857599 Mbps
  Average Delay: 0.00427019 seconds
  Packet Delivery Ratio: 1

Iteration Number 11 :
Flow 1 (10.0.0.1 -> 10.0.0.50)
  Tx Packets: 5
  Rx Packets: 0
  Throughput: -0 Mbps
  Average Delay: N/A
  Packet Delivery Ratio: N/A

Iteration Number 12 :
Flow 1 (10.0.0.1 -> 10.0.0.50)
  Tx Packets: 5
  Rx Packets: 4
  Throughput: 0.0106983 Mbps
  Average Delay: 0.169321 seconds
  Packet Delivery Ratio: 0.8

Flow 2 (10.0.0.35 -> 10.0.0.34)
  Tx Packets: 2
  Rx Packets: 2
  Throughput: 0.000199848 Mbps
  Average Delay: 0.0021974 seconds
  Packet Delivery Ratio: 1

Flow 3 (10.0.0.35 -> 10.0.0.50)
  Tx Packets: 1
  Rx Packets: 1
  Throughput: 0.160256 Mbps
  Average Delay: 0.00228517 seconds
  Packet Delivery Ratio: 1

Flow 4 (10.0.0.34 -> 10.0.0.35)
  Tx Packets: 2
  Rx Packets: 2
  Throughput: 8.50022e-05 Mbps
  Average Delay: 0.00137773 seconds
  Packet Delivery Ratio: 1

Flow 5 (10.0.0.34 -> 10.0.0.27)
  Tx Packets: 2
  Rx Packets: 2
  Throughput: 0.000199507 Mbps
  Average Delay: 0.00107773 seconds
  Packet Delivery Ratio: 1

Flow 6 (10.0.0.27 -> 10.0.0.1)
  Tx Packets: 3
  Rx Packets: 3
  Throughput: 0.000129605 Mbps
```

```
  Average Delay: 0.00181114 seconds
  Packet Delivery Ratio: 1

Flow 26 (10.0.0.42 -> 10.0.0.26)
  Tx Packets: 1
  Rx Packets: 1
  Throughput: 0.000365688 Mbps
  Average Delay: 1.00143 seconds
  Packet Delivery Ratio: 1

Flow 27 (10.0.0.42 -> 10.0.0.50)
  Tx Packets: 1
  Rx Packets: 1
  Throughput: 0.0684119 Mbps
  Average Delay: 0.00535303 seconds
  Packet Delivery Ratio: 1

Flow 28 (10.0.0.26 -> 10.0.0.24)
  Tx Packets: 2
  Rx Packets: 2
  Throughput: 0.000176571 Mbps
  Average Delay: 0.00277718 seconds
  Packet Delivery Ratio: 1

Flow 29 (10.0.0.26 -> 10.0.0.35)
  Tx Packets: 2
  Rx Packets: 2
  Throughput: 0.000732571 Mbps
  Average Delay: 0.00522291 seconds
  Packet Delivery Ratio: 1

Flow 30 (10.0.0.35 -> 10.0.0.5)
  Tx Packets: 2
  Rx Packets: 2
  Throughput: 0.000740059 Mbps
  Average Delay: 0.0042682 seconds
  Packet Delivery Ratio: 1

Flow 31 (10.0.0.26 -> 10.0.0.42)
  Tx Packets: 1
  Rx Packets: 1
  Throughput: 0.212023 Mbps
  Average Delay: 0.00107951 seconds
  Packet Delivery Ratio: 1

Flow 32 (10.0.0.35 -> 10.0.0.26)
  Tx Packets: 1
  Rx Packets: 1
  Throughput: 0.0407602 Mbps
  Average Delay: 0.00748711 seconds
  Packet Delivery Ratio: 1

Total Average Delay over 20 experiments: 0.113705 seconds
Total Packet Delivery Ratio over 20 experiments: 0.96772
scorpius@jarvis:~/ns-allinone-3.40/ns-3.40$
```



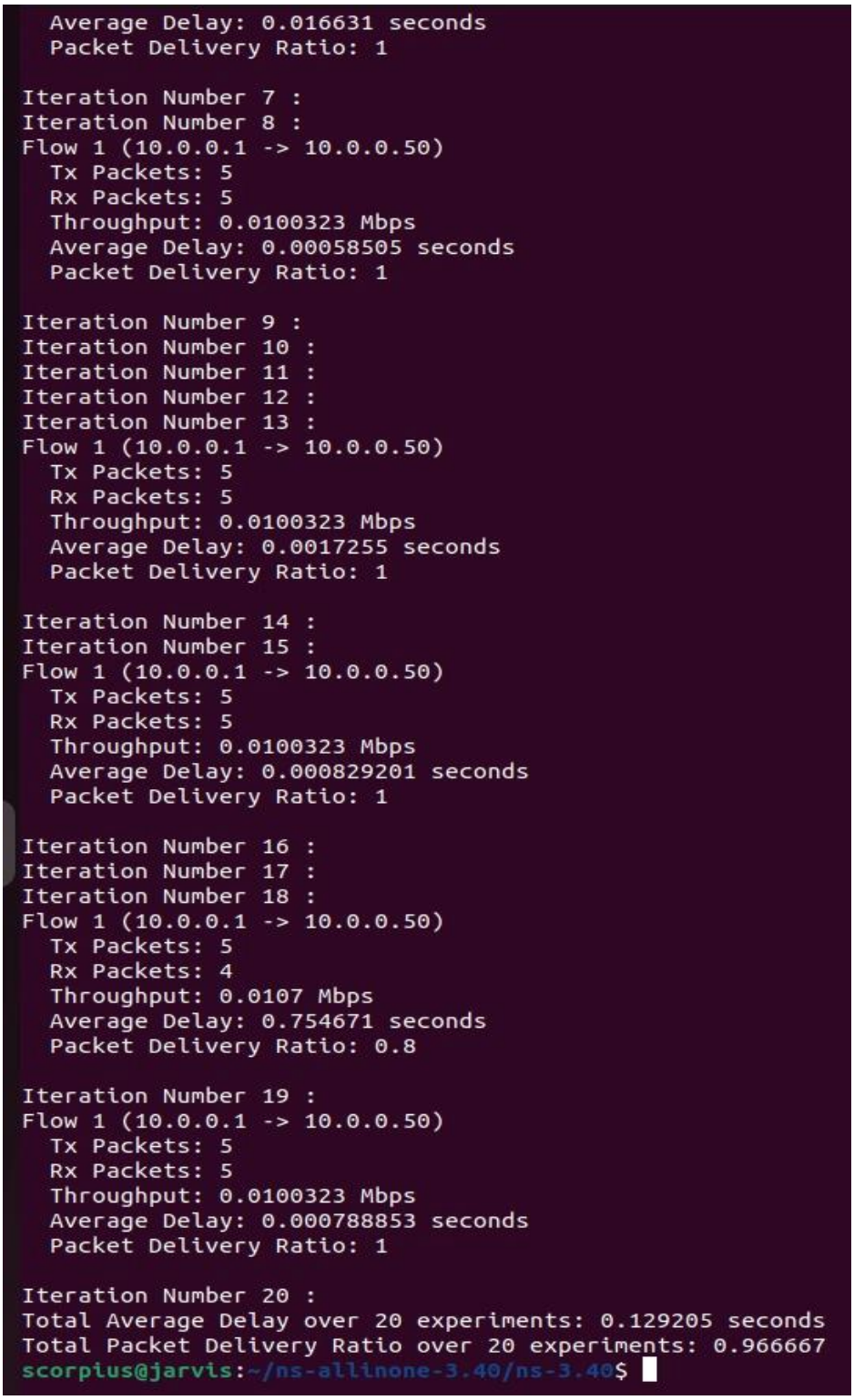
NOTE: This was configured on ns-3.40 for netanim-3.109, it may not show appropriate dynamic model on previous versions without a few tweaking.

XML FILES:

NOTE: Total 20 are made, ranging from 0 to 19.

17. Other Attempts:

1. **USING OLSR ROUTING:**



OLSR wasn't able to cope with the dynamic nature of the model, and had to recalculate the whole routing table and in most iterations didn't send any packets, so AODV was used as an alternative.

2. **TimeProbe Class and OnOff Helpers:**

A custom class named TimeProbe was made that used tracebacks from the Server and Client nodes to calculate Delay, received packets, sent packets, and PDR of the iteration. However, we needed to display the paths of the packets as well, and then FlowMonitor Module was discovered that had all of these capabilities, so this class became redundant and removed, UDP Server and Client Helpers worked better with the OnOff scheduling method, so that was replaced as well.

Relevant Code Portion:

```
class TimeProbe: public ns3::Object {
public:
    void PacketSent(Ptr<const Packet> p) {
    sendTimes[p->GetUid()] = Simulator::Now();
    sentPackets++;
  }

  void PacketReceived(Ptr<const Packet> p, const Address& address) {
    uint32_t packetId = p->GetUid();
    if (sendTimes.find(packetId) != sendTimes.end()) {
      Time sentTime = sendTimes[packetId];
      Time receivedTime = Simulator::Now();
      totalDelay += (receivedTime - sentTime);
      receivedPackets++;
    }
  }

  double GetAverageDelay() {
    if (receivedPackets > 0)
      return totalDelay.GetSeconds() / receivedPackets;
    else
      return 0.0;
  }

  uint32_t GetReceivedPackets(){
  return receivedPackets;
  }

  uint32_t GetSentPackets(){
  return sentPackets;}

private:
```

```
   std::map<uint32_t, Time> sendTimes;
   Time totalDelay;
   uint32_t receivedPackets = 0, sentPackets = 0;
};
// Timestamp when packets are sent (at the application layer)
   Ptr<TimeProbe> timeProbe = CreateObject<TimeProbe>();


   // Calculate time to send five packets
Time onTime = Seconds(interPacketInterval.GetSeconds());

// Set a fixed OffTime, for example, 1 second
Time offTime = Seconds(1.0);


// Configure OnOffHelper
OnOffHelper onoff("ns3::UdpSocketFactory",
Address(InetSocketAddress(interfaces.GetAddress(49), port)));
onoff.SetAttribute("PacketSize", UintegerValue(packetSize));
onoff.SetAttribute("DataRate", DataRateValue(dataRate));
// Configure OnOffHelper with the new OnTime and OffTime
onoff.SetAttribute("OnTime", StringValue("ns3::ConstantRandomVariable[Constant=" +
std::to_string(onTime.GetSeconds()) + "]"));
onoff.SetAttribute("OffTime", StringValue("ns3::ConstantRandomVariable[Constant=" +
std::to_string(offTime.GetSeconds()) + "]"));


   // ...

ApplicationContainer sourceApps = onoff.Install(nodes.Get(0));
sourceApps.Start(Seconds(2.0));
sourceApps.Stop(Seconds(ANIMATION_TIME));
sourceApps.Get(0)->TraceConnectWithoutContext("Tx", MakeCallback(&TimeProbe::PacketSent,
timeProbe));

PacketSinkHelper sink("ns3::UdpSocketFactory",
                Address(InetSocketAddress(Ipv4Address::GetAny(), port)));
ApplicationContainer sinkApps = sink.Install(nodes.Get(49));
sinkApps.Start(Seconds(0.0));
sinkApps.Stop(Seconds(ANIMATION_TIME));
sinkApps.Get(0)->TraceConnectWithoutContext("Rx",
MakeCallback(&TimeProbe::PacketReceived, timeProbe));
// Calculate and print delay and PDR
   double averageDelay = timeProbe->GetAverageDelay();
```

```
double pdr = static_cast<double>(timeProbe->GetReceivedPackets()) / (timeProbe->GetSentPackets());

std::cout << "Average Delay: " << averageDelay << " seconds" << std::endl;
std::cout << "Packet Delivery Ratio: " << pdr << std::endl;
std::cout << "Recieved Packets: " << (timeProbe->GetReceivedPackets()) << std::endl;
std::cout << "Number of Packets Sent: " << timeProbe->GetSentPackets() << std::endl<<std::endl;
```

Sample Output:

```
Packet Delivery Ratio: 0.714286
Recieved Packets: 5
Number of Packets Sent: 7

Iteration Number 13 :
Average Delay: 0.0010209 seconds
Packet Delivery Ratio: 1
Recieved Packets: 7
Number of Packets Sent: 7

Iteration Number 14 :
Average Delay: 0.000969356 seconds
Packet Delivery Ratio: 1
Recieved Packets: 7
Number of Packets Sent: 7

Iteration Number 15 :
Max Packets per trace file exceeded
Average Delay: 0.00119747 seconds
Packet Delivery Ratio: 0.428571
Recieved Packets: 3
Number of Packets Sent: 7

Iteration Number 16 :
Average Delay: 0.00155534 seconds
Packet Delivery Ratio: 1
Recieved Packets: 7
Number of Packets Sent: 7

Iteration Number 17 :
Average Delay: 0.00593191 seconds
Packet Delivery Ratio: 1
Recieved Packets: 5
Number of Packets Sent: 5

Iteration Number 18 :
Average Delay: 0.000950382 seconds
Packet Delivery Ratio: 0.857143
Recieved Packets: 6
Number of Packets Sent: 7

Iteration Number 19 :
Average Delay: 0.00313834 seconds
Packet Delivery Ratio: 0.666667
Recieved Packets: 4
Number of Packets Sent: 6

Iteration Number 20 :
Average Delay: 0.000639567 seconds
Packet Delivery Ratio: 0.857143
Recieved Packets: 6
Number of Packets Sent: 7

Total Average Delay over 20 experiments: 0.0022763 seconds
Total Packet Delivery Ratio over 20 experiments: 0.846429
```

3. **Miscellaneous:**

   CSMA, P2P were also implemented to no avail, Trace files for each node and sending of numerous packets were also done to select a best suited approach, an example output of a high volume of packets with Log of the packets is attached: