**Practical No 16**

**Aim: -** Mini Project - Real time problem solving.

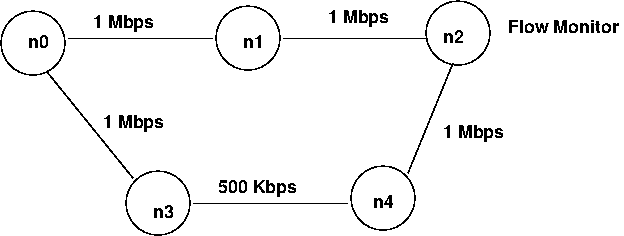
**Objective: -**

To Learn and implement the real time throughput & congestion analysis.

**Theory: -**

**TASK -1: PROBLEM DEFINITION OF MINI PROJECT**

Create a simple topology of two nodes – Node1 and Node2, separated by a point-to-point link. Setup a UdpClient on Node1 and UdpServer on Node2. Start the client application, and measure end to end throughput while varying the latency of the link. Now add another client application to Node1 and a server instance to Node2. What you need to configure to ensure that there is no conflict? Measure end-to-end throughput with the extra client and server application instances. Show screenshots of pcap traces which indicate that delivery is made to the appropriate server instance.  
Make the following simulations in NS-3

Make a topology of 4 nodes in the following way.

n0 starts CBR traffic at time 1.0 of rate 900 Kbps destined for n2. n0 starts another CBR traffic at time 1.5 of rate 300 Kbps destined for node n1. At time 2.0, link from n0 to n1 goes down. Use a dynamic routing protocol so that path n0-n3-n2 is used now At time 2.7, link n0-n1 comes up again. At time 3.0, CBR traffic destined for node n1 stops. CBR destined for n2 stops at time 3.5. Use a Flow monitor to monitor losses at n2. Draw a graph of percentage loss as a function of time for the duration of simulation.

Give an explanation for results you find.

**Program: -**

#include "ns3/core-module.h"

#include "ns3/network-module.h"

#include "ns3/internet-module.h"

#include "ns3/point-to-point-module.h"

#include "ns3/applications-module.h"

#include <fstream>

#include "ns3/flow-monitor-module.h"

#include "ns3/netanim-module.h"

#include "ns3/mobility-module.h"

using namespace ns3;

using namespace std;

NS\_LOG\_COMPONENT\_DEFINE ("Lab-4-1");

int

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

{

  double delay = 2;

  Time::SetResolution (Time::NS);

  LogComponentEnable ("UdpEchoClientApplication", LOG\_LEVEL\_INFO);

  LogComponentEnable ("UdpEchoServerApplication", LOG\_LEVEL\_INFO);

  CommandLine cmd;

  cmd.AddValue("delay", "P2P delay /latency in ms ", delay);

  cmd.Parse(argc,argv);

  NodeContainer nodes;

  nodes.Create (2);

  PointToPointHelper pointToPoint;

  pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));

  pointToPoint.SetChannelAttribute ("Delay", TimeValue (MilliSeconds(delay)));

  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);

  UdpEchoServerHelper echoServer1 (9000);

  UdpEchoServerHelper echoServer2 (9001);

  ApplicationContainer serverApps = echoServer1.Install (nodes.Get (1));

  serverApps.Start (Seconds (1.0));

  serverApps.Stop (Seconds (10.0));

  serverApps = echoServer2.Install (nodes.Get (1));

  serverApps.Start (Seconds (1.0));

  serverApps.Stop (Seconds (10.0));

  UdpEchoClientHelper echoClient1 (interfaces.GetAddress (1), 9000);

  echoClient1.SetAttribute ("MaxPackets", UintegerValue (1));

  echoClient1.SetAttribute ("Interval", TimeValue (Seconds (1.0)));

  echoClient1.SetAttribute ("PacketSize", UintegerValue (1024));

  ApplicationContainer clientApps = echoClient1.Install (nodes.Get (0));

  clientApps.Start (Seconds (2.0));

  clientApps.Stop (Seconds (10.0));

  UdpEchoClientHelper echoClient2 (interfaces.GetAddress (1), 9001);

  echoClient2.SetAttribute ("MaxPackets", UintegerValue (1));

  echoClient2.SetAttribute ("Interval", TimeValue (Seconds (1.0)));

  echoClient2.SetAttribute ("PacketSize", UintegerValue (1024));

  clientApps = echoClient2.Install (nodes.Get (0));

  clientApps.Start (Seconds (2.0));

  clientApps.Stop (Seconds (10.0));

  //

  // Tracing

  //

    AsciiTraceHelper ascii;

    pointToPoint.EnableAscii(ascii.CreateFileStream ("lab-4-1.tr"), devices);

    pointToPoint.EnablePcap("lab-4-1",devices, false);

  //

  // Calculate Throughput using Flowmonitor

  //

    FlowMonitorHelper flowmon;

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

  //

  // Now, do the actual simulation.

  //

    NS\_LOG\_INFO ("Run Simulation.");

    AnimationInterface anim("MiniProject-First.xml");

    Simulator::Stop (Seconds(11.0));

    Simulator::Run ();

    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);

        if ((t.sourceAddress=="10.1.1.1" && t.destinationAddress == "10.1.1.2"))

        {

            std::cout << "Flow " << i->first  << " (" << t.sourceAddress<<":"<<t.sourcePort << " -> " << t.destinationAddress <<":"<<t.destinationPort<< ")\n";

            std::cout << "  Tx Bytes:   " << i->second.txBytes << "\n";

            std::cout << "  Rx Bytes:   " << i->second.rxBytes << "\n";

            std::cout << "  Throughput: " << i->second.rxBytes \* 8.0 / (i->second.timeLastRxPacket.GetSeconds() - i->second.timeFirstTxPacket.GetSeconds())/1024/1024  << " Mbps\n";

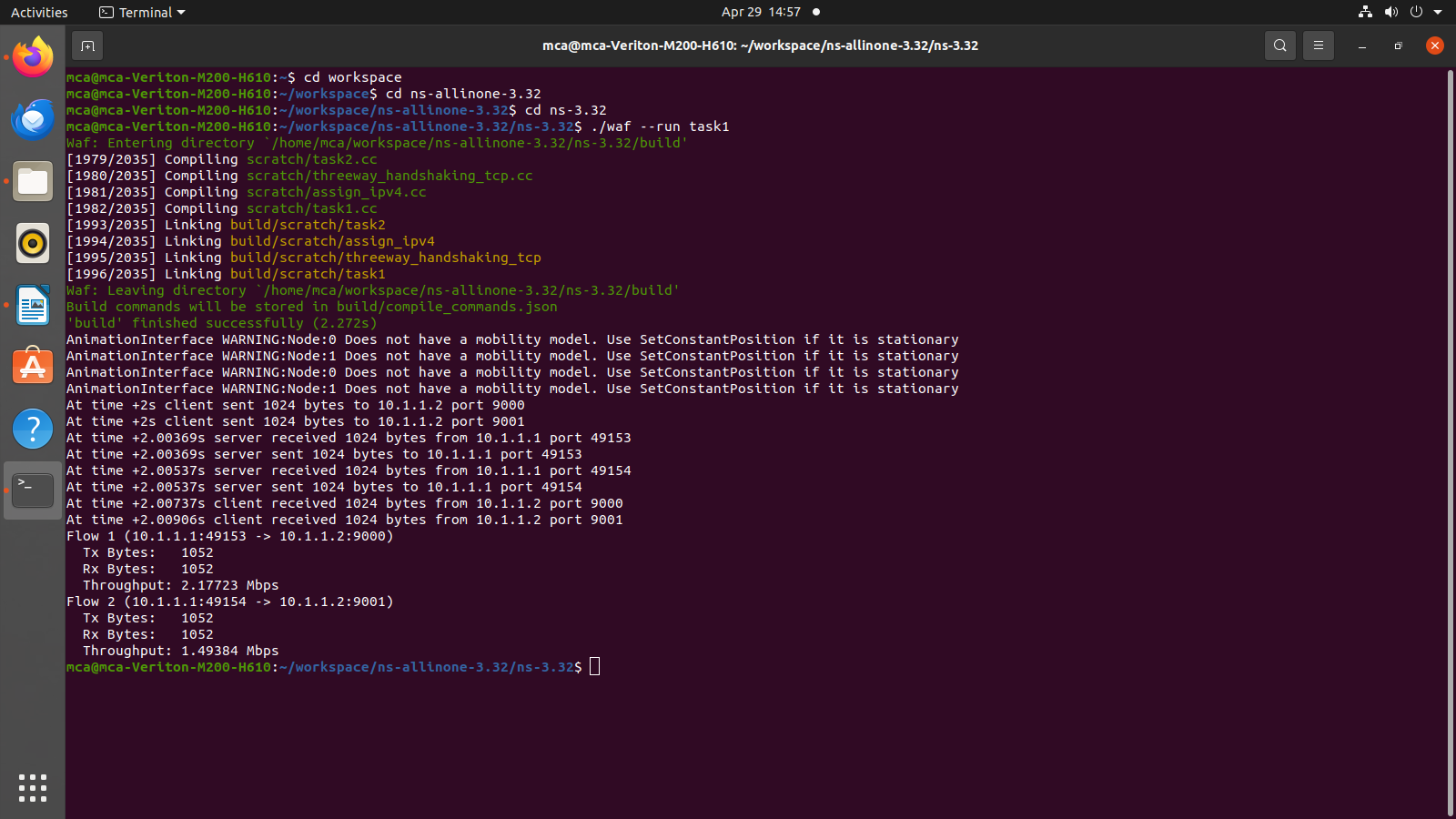
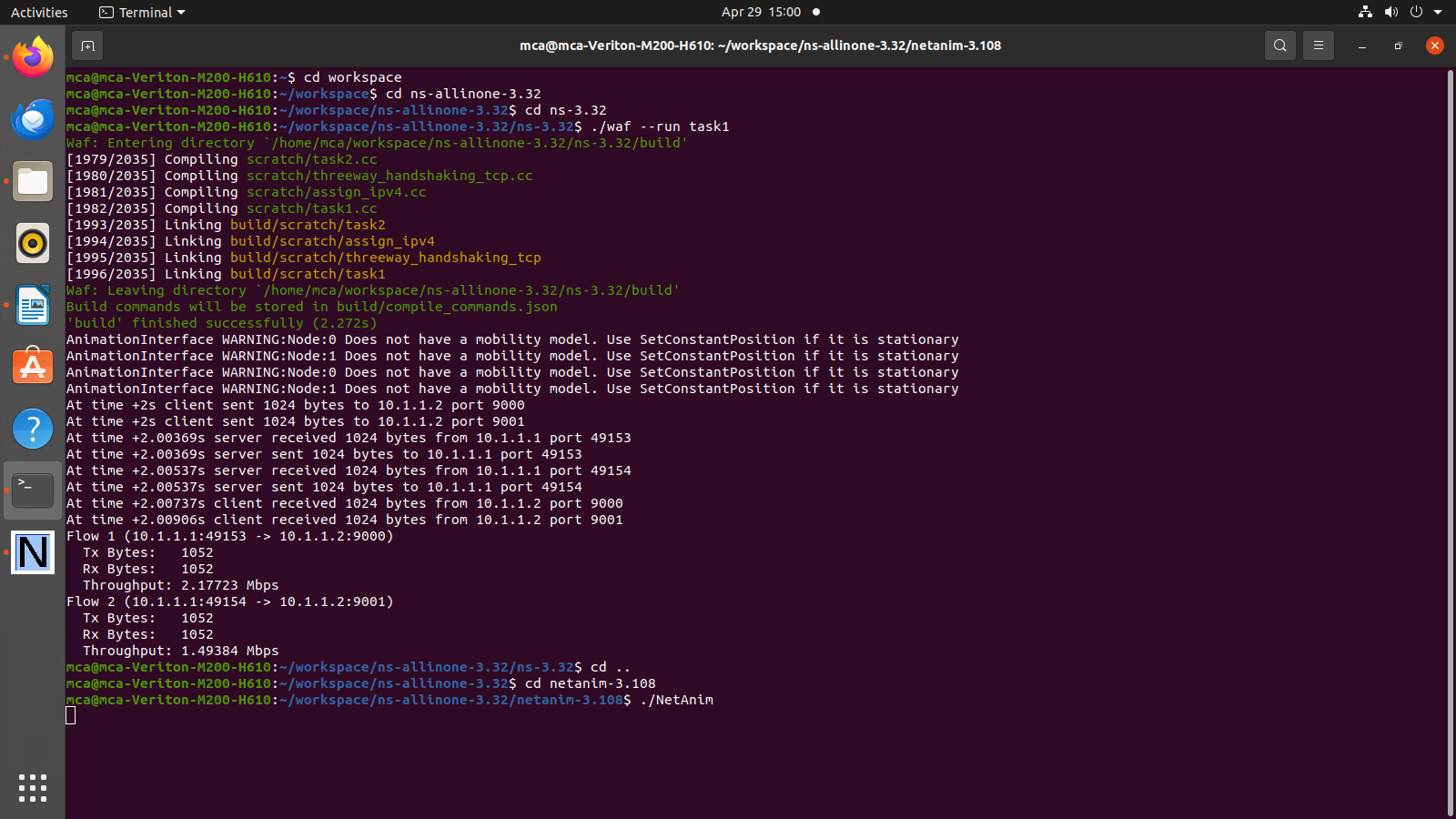
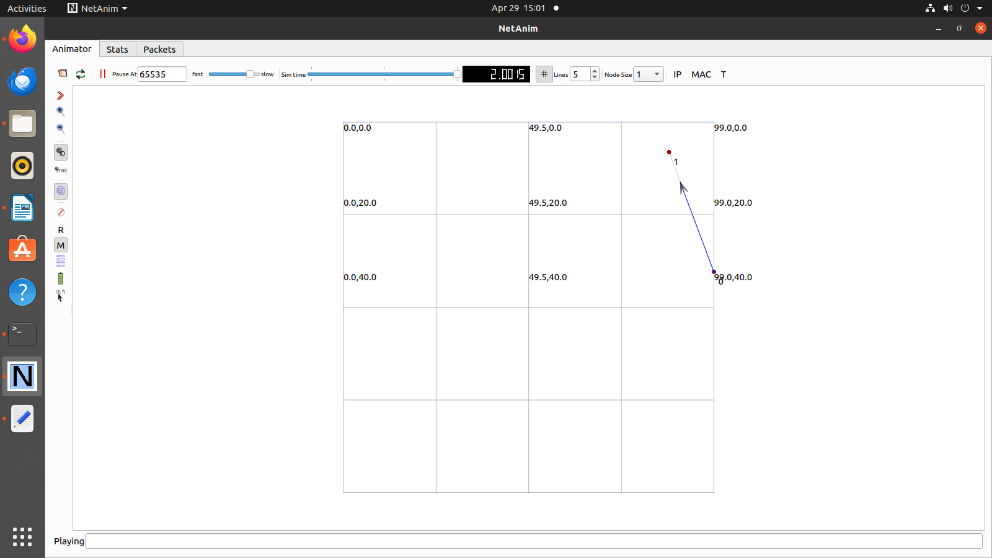
        }

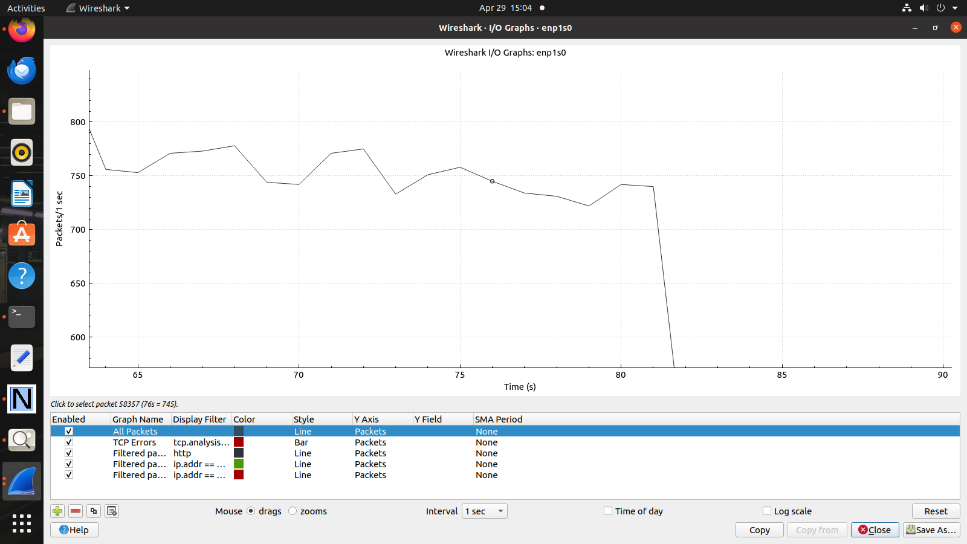
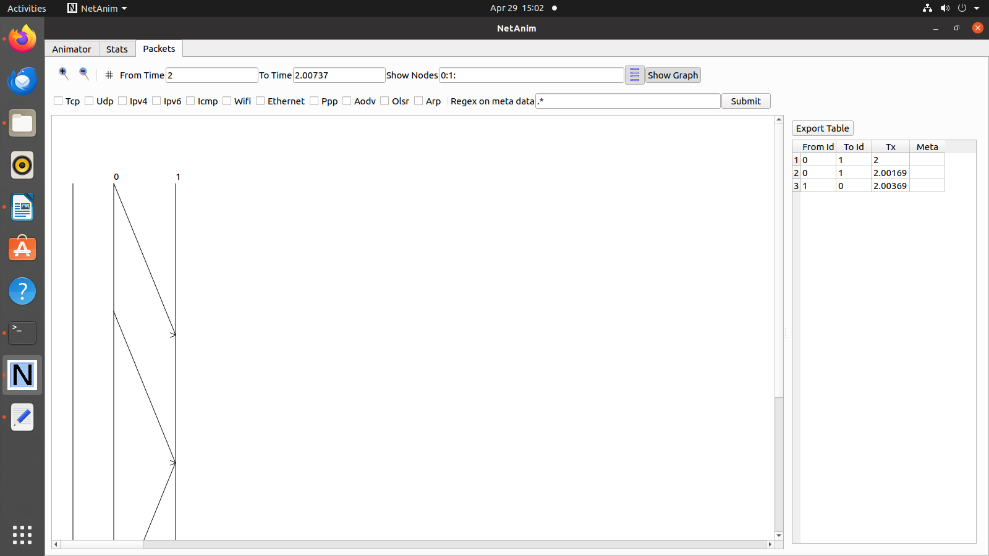
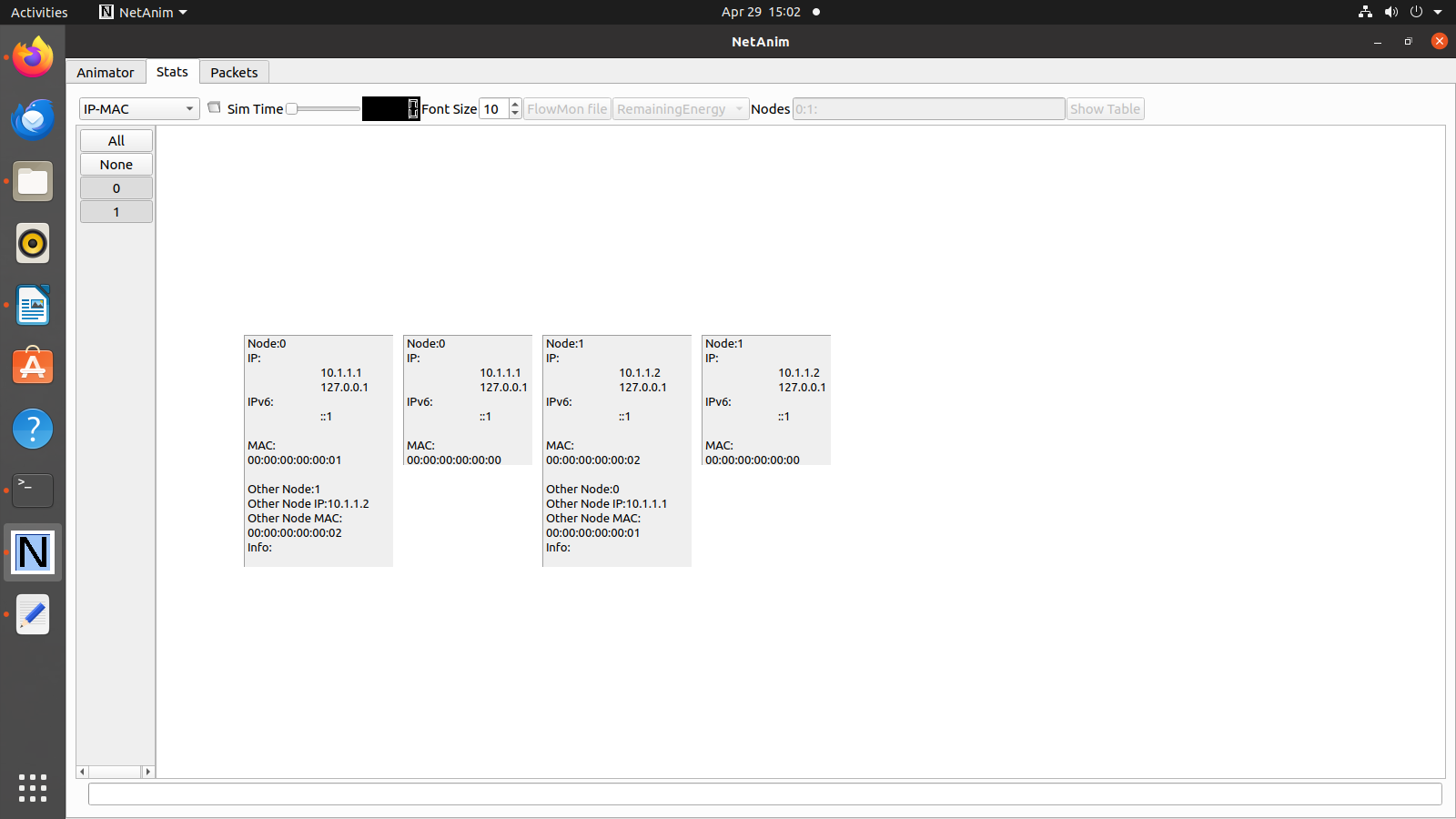
       }

  Simulator::Destroy ();

  return 0;

}

**Output: -**



**Conclusion: -**

Successful analysis of given network throughput.

**TASK-2: ANALYSE THE CONGESTION AND PLOT THE GRAPH OF CONGESTION WINDOW USING NS3.**

**Theory: -**

Network congestion refers to a reduction in quality of service (QOS) that causes packet loss, queueing delay, or the blocking of new connections. Typically, network congestion occurs in cases of traffic overloading when a link or network node is handling data in excess of its capacity.

**Program: -**

#include <fstream>

#include "ns3/core-module.h"

#include "ns3/network-module.h"

#include "ns3/internet-module.h"

#include "ns3/point-to-point-module.h"

#include "ns3/applications-module.h"

using namespace ns3;

NS\_LOG\_COMPONENT\_DEFINE ("FifthScriptExample");

// ===========================================================================

//

//         node 0                 node 1

//   +----------------+    +----------------+

//   |    ns-3 TCP    |    |    ns-3 TCP    |

//   +----------------+    +----------------+

//   |    10.1.1.1    |    |    10.1.1.2    |

//   +----------------+    +----------------+

//   | point-to-point |    | point-to-point |

//   +----------------+    +----------------+

//           |                     |

//           +---------------------+

//                5 Mbps, 2 ms

//

//

// We want to look at changes in the ns-3 TCP congestion window.  We need

// to crank up a flow and hook the CongestionWindow attribute on the socket

// of the sender.  Normally one would use an on-off application to generate a

// flow, but this has a couple of problems.  First, the socket of the on-off

// application is not created until Application Start time, so we wouldn't be

// able to hook the socket (now) at configuration time.  Second, even if we

// could arrange a call after start time, the socket is not public so we

// couldn't get at it.

//

// So, we can cook up a simple version of the on-off application that does what

// we want.  On the plus side we don't need all of the complexity of the on-off

// application.  On the minus side, we don't have a helper, so we have to get

// a little more involved in the details, but this is trivial.

//

// So first, we create a socket and do the trace connect on it; then we pass

// this socket into the constructor of our simple application which we then

// install in the source node.

// ===========================================================================

//

class MyApp : public Application

{

public:

  MyApp ();

  virtual ~MyApp();

  void Setup (Ptr<Socket> socket, Address address, uint32\_t packetSize, uint32\_t nPackets, DataRate dataRate);

private:

  virtual void StartApplication (void);

  virtual void StopApplication (void);

  void ScheduleTx (void);

  void SendPacket (void);

  Ptr<Socket>     m\_socket;

  Address         m\_peer;

  uint32\_t        m\_packetSize;

  uint32\_t        m\_nPackets;

  DataRate        m\_dataRate;

  EventId         m\_sendEvent;

  bool            m\_running;

  uint32\_t        m\_packetsSent;

};

MyApp::MyApp ()

  : m\_socket (0),

    m\_peer (),

    m\_packetSize (0),

    m\_nPackets (0),

    m\_dataRate (0),

    m\_sendEvent (),

    m\_running (false),

    m\_packetsSent (0)

{

}

MyApp::~MyApp()

{

  m\_socket = 0;

}

void

MyApp::Setup (Ptr<Socket> socket, Address address, uint32\_t packetSize, uint32\_t nPackets, DataRate dataRate)

{

  m\_socket = socket;

  m\_peer = address;

  m\_packetSize = packetSize;

  m\_nPackets = nPackets;

  m\_dataRate = dataRate;

}

void

MyApp::StartApplication (void)

{

  m\_running = true;

  m\_packetsSent = 0;

  m\_socket->Bind ();

  m\_socket->Connect (m\_peer);

  SendPacket ();

}

void

MyApp::StopApplication (void)

{

  m\_running = false;

  if (m\_sendEvent.IsRunning ())

    {

      Simulator::Cancel (m\_sendEvent);

    }

  if (m\_socket)

    {

      m\_socket->Close ();

    }

}

void

MyApp::SendPacket (void)

{

  Ptr<Packet> packet = Create<Packet> (m\_packetSize);

  m\_socket->Send (packet);

  if (++m\_packetsSent < m\_nPackets)

    {

      ScheduleTx ();

    }

}

void

MyApp::ScheduleTx (void)

{

  if (m\_running)

    {

      Time tNext (Seconds (m\_packetSize \* 8 / static\_cast<double> (m\_dataRate.GetBitRate ())));

      m\_sendEvent = Simulator::Schedule (tNext, &MyApp::SendPacket, this);

    }

}

static void

CwndChange (uint32\_t oldCwnd, uint32\_t newCwnd)

{

  NS\_LOG\_UNCOND (Simulator::Now ().GetSeconds () << "\t" << newCwnd);

}

static void

RxDrop (Ptr<const Packet> p)

{

  NS\_LOG\_UNCOND ("RxDrop at " << Simulator::Now ().GetSeconds ());

}

int

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

{

  CommandLine cmd (\_\_FILE\_\_);

  cmd.Parse (argc, argv);

  NodeContainer nodes;

  nodes.Create (2);

  PointToPointHelper pointToPoint;

  pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));

  pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

  NetDeviceContainer devices;

  devices = pointToPoint.Install (nodes);

  Ptr<RateErrorModel> em = CreateObject<RateErrorModel> ();

  em->SetAttribute ("ErrorRate", DoubleValue (0.00001));

  devices.Get (1)->SetAttribute ("ReceiveErrorModel", PointerValue (em));

  InternetStackHelper stack;

  stack.Install (nodes);

Ipv4AddressHelper address;

  address.SetBase ("10.1.1.0", "255.255.255.252");

  Ipv4InterfaceContainer interfaces = address.Assign (devices);

  uint16\_t sinkPort = 8080;

  Address sinkAddress (InetSocketAddress (interfaces.GetAddress (1), sinkPort));

  PacketSinkHelper packetSinkHelper ("ns3::TcpSocketFactory", InetSocketAddress (Ipv4Address::GetAny (), sinkPort));

  ApplicationContainer sinkApps = packetSinkHelper.Install (nodes.Get (1));

  sinkApps.Start (Seconds (0.));

  sinkApps.Stop (Seconds (20.));

  Ptr<Socket> ns3TcpSocket = Socket::CreateSocket (nodes.Get (0), TcpSocketFactory::GetTypeId ());

  ns3TcpSocket->TraceConnectWithoutContext ("CongestionWindow", MakeCallback (&CwndChange));

  Ptr<MyApp> app = CreateObject<MyApp> ();

  app->Setup (ns3TcpSocket, sinkAddress, 1040, 1000, DataRate ("1Mbps"));

  nodes.Get (0)->AddApplication (app);

  app->SetStartTime (Seconds (1.));

  app->SetStopTime (Seconds (20.));

  devices.Get (1)->TraceConnectWithoutContext ("PhyRxDrop", MakeCallback (&RxDrop));

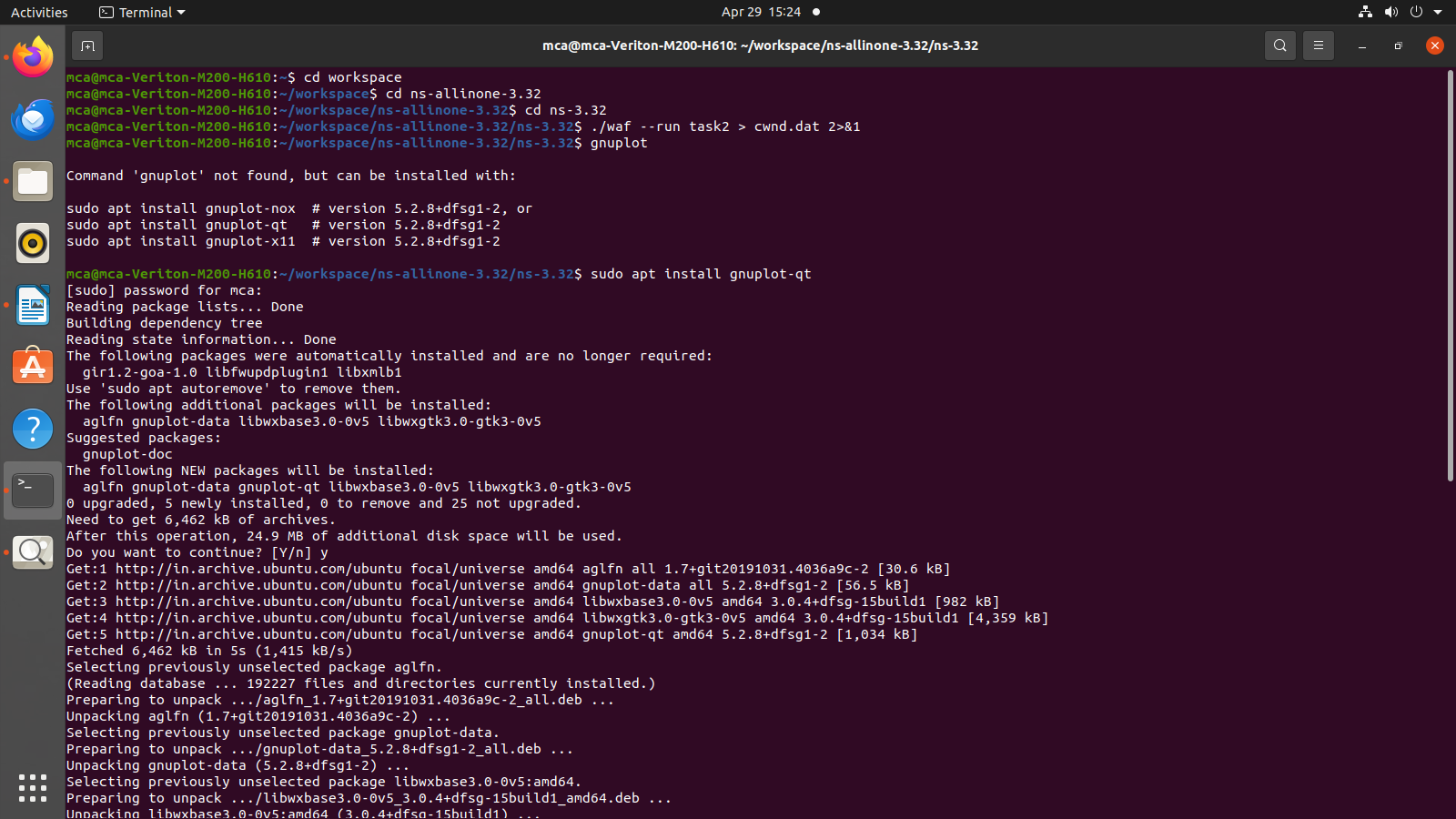
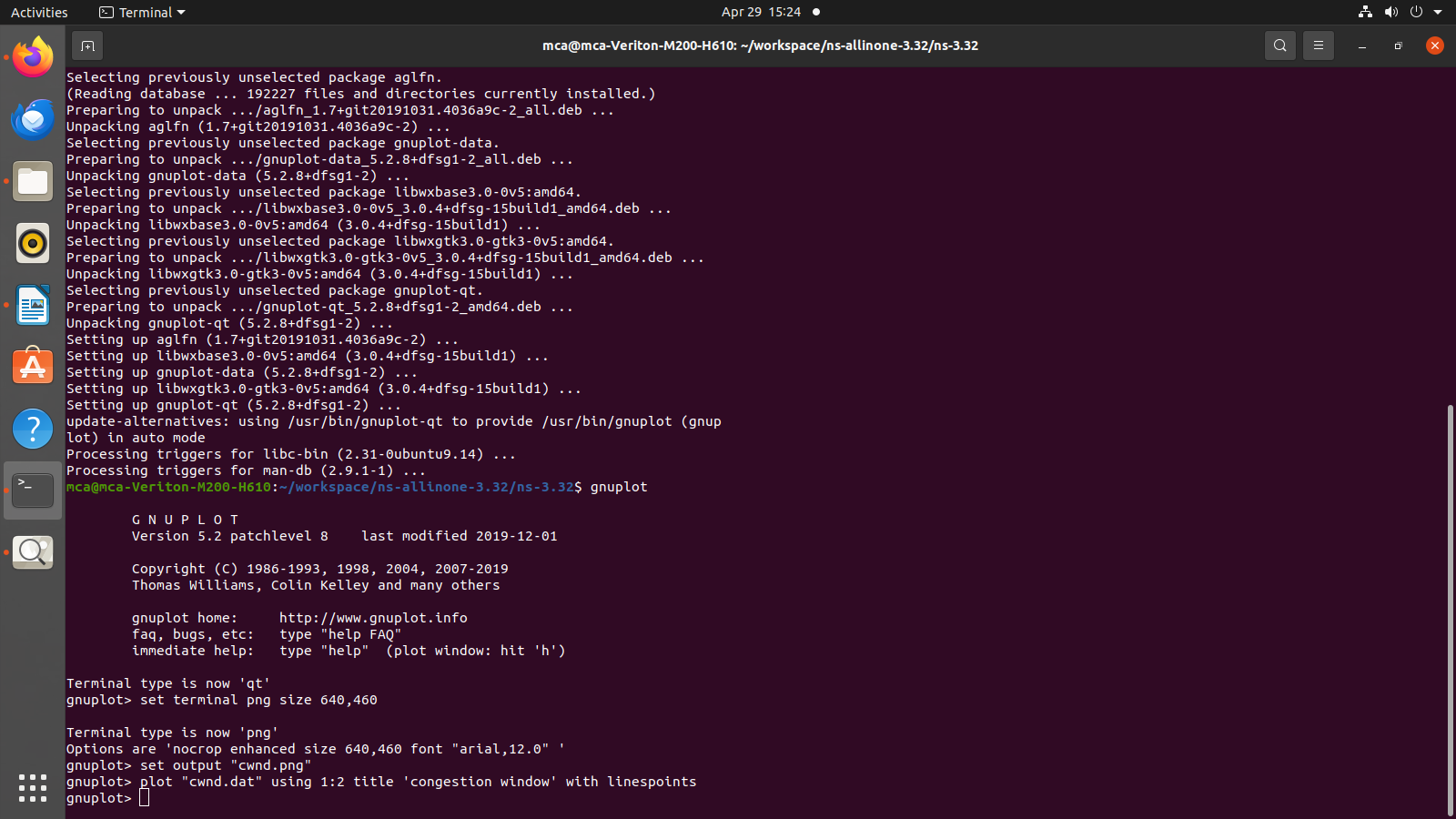
  Simulator::Stop (Seconds (20));

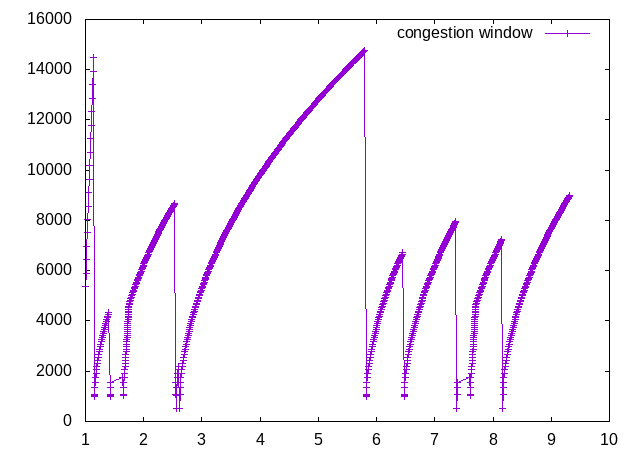
  Simulator::Run ();

  Simulator::Destroy ();

  return 0;

}

**Output: -**

****

**Conclusion: -**

Successful presentation of congestion window.