**NAME :- CHODVADIYA SUJAL HASMUKHBHAI**

**ROLL NO:- 2022BCS0187**

**SUB :- ICS224( LAB ASSIGNMENT 6 )**

**QUE 1 :-**

**CODE:-**

#include "ns3/applications-module.h"

#include "ns3/core-module.h"

#include "ns3/internet-module.h"

#include "ns3/network-module.h"

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

// Default Network Topology

//

//       10.1.1.0

// n0 -------------- n1

//    point-to-point

//

using namespace ns3;

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

int

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

{

    CommandLine cmd(\_\_FILE\_\_);

    cmd.Parse(argc, argv);

    Time::SetResolution(Time::NS);

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

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

    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);

    UdpEchoServerHelper echoServer(9);

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

    serverApps.Start(Seconds(1.0));

    serverApps.Stop(Seconds(10.0));

    UdpEchoClientHelper echoClient(interfaces.GetAddress(1), 9);

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

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

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

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

    clientApps.Start(Seconds(2.0));

    clientApps.Stop(Seconds(10.0));

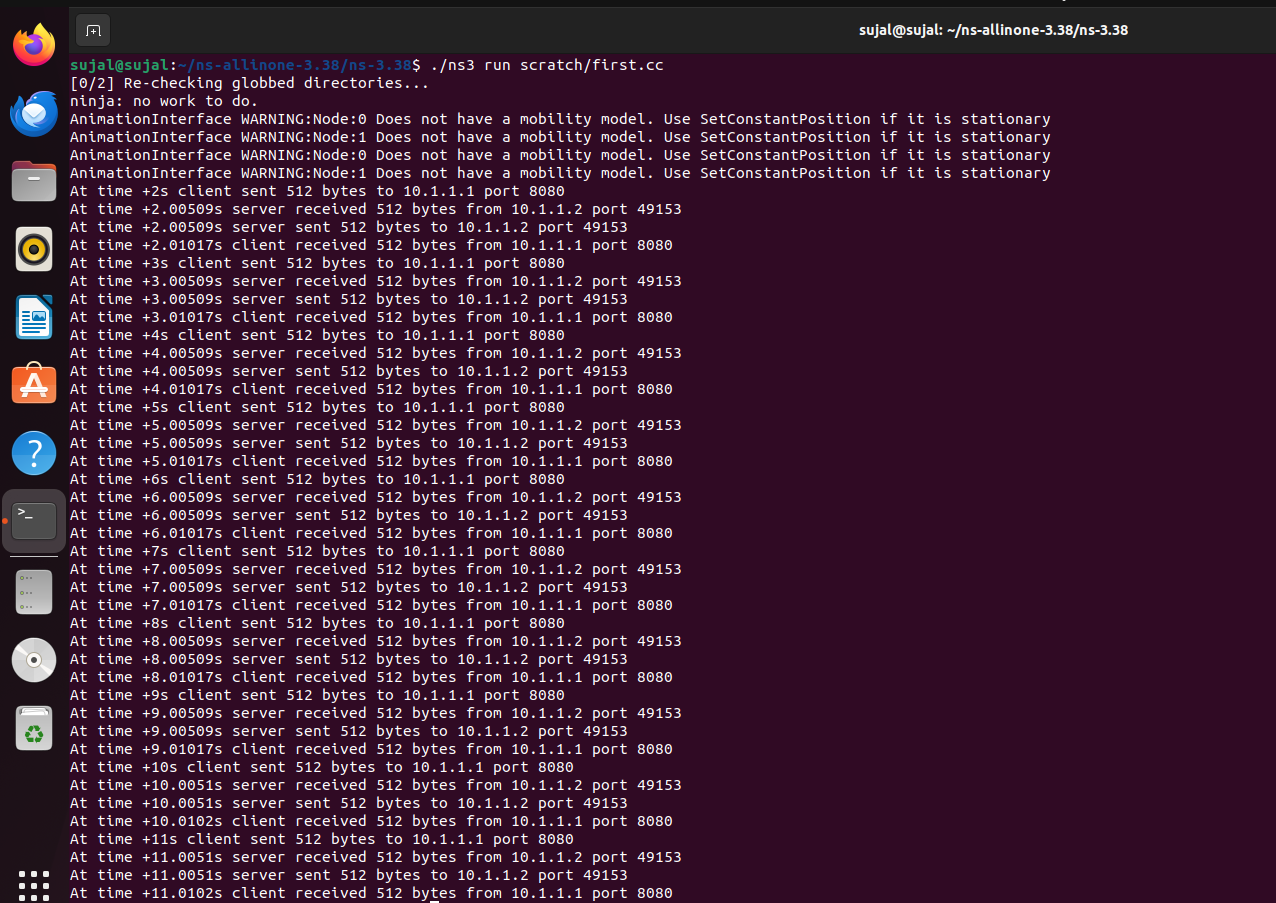
    Simulator::Run();

    Simulator::Destroy();

    return 0;

}

**OUTPUT:-**

****

**QUE 2 :-**

**CODE:-**

/\* -\*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -\*- \*/

#include <fstream>

#include <string>

#include "ns3/core-module.h"

#include "ns3/network-module.h"

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

#include "ns3/applications-module.h"

#include "ns3/internet-module.h"

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

#include "ns3/ipv4-global-routing-helper.h"

using namespace ns3;

NS\_LOG\_COMPONENT\_DEFINE ("Lab2");

class MyApp : public Application

{

public:

  MyApp ();

  virtual ~MyApp();

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

  void ChangeRate(DataRate newrate);

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);

    }

}

void

MyApp::ChangeRate(DataRate newrate)

{

   m\_dataRate = newrate;

   return;

}

static void

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

{

  std::cout << Simulator::Now ().GetSeconds () << "\t" << newCwnd <<"\n";

}

void

IncRate (Ptr<MyApp> app, DataRate rate)

{

  app->ChangeRate(rate);

    return;

}

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

{

  std::string lat = "2ms";

  std::string rate = "500kb/s"; // P2P link

  bool enableFlowMonitor = false;

  CommandLine cmd;

  cmd.AddValue ("latency", "P2P link Latency in miliseconds", lat);

  cmd.AddValue ("rate", "P2P data rate in bps", rate);

  cmd.AddValue ("EnableMonitor", "Enable Flow Monitor", enableFlowMonitor);

  cmd.Parse (argc, argv);

//

// Explicitly create the nodes required by the topology (shown above).

//

  NS\_LOG\_INFO ("Create nodes.");

  NodeContainer c; // ALL Nodes

  c.Create(6);

  NodeContainer n0n4 = NodeContainer (c.Get (0), c.Get (4));

  NodeContainer n1n4 = NodeContainer (c.Get (1), c.Get (4));

  NodeContainer n2n5 = NodeContainer (c.Get (2), c.Get (5));

  NodeContainer n3n5 = NodeContainer (c.Get (3), c.Get (5));

  NodeContainer n4n5 = NodeContainer (c.Get (4), c.Get (5));

//

// Install Internet Stack

//

  InternetStackHelper internet;

  internet.Install (c);

  // We create the channels first without any IP addressing information

  NS\_LOG\_INFO ("Create channels.");

  PointToPointHelper p2p;

  p2p.SetDeviceAttribute ("DataRate", StringValue (rate));

  p2p.SetChannelAttribute ("Delay", StringValue (lat));

  NetDeviceContainer d0d4 = p2p.Install (n0n4);

  NetDeviceContainer d1d4 = p2p.Install (n1n4);

  NetDeviceContainer d4d5 = p2p.Install (n4n5);

  NetDeviceContainer d2d5 = p2p.Install (n2n5);

  NetDeviceContainer d3d5 = p2p.Install (n3n5);

    // Later, we add IP addresses.

  NS\_LOG\_INFO ("Assign IP Addresses.");

  Ipv4AddressHelper ipv4;

  ipv4.SetBase ("10.1.1.0", "255.255.255.0");

  Ipv4InterfaceContainer i0i4 = ipv4.Assign (d0d4);

  ipv4.SetBase ("10.1.2.0", "255.255.255.0");

  Ipv4InterfaceContainer i1i4 = ipv4.Assign (d1d4);

  ipv4.SetBase ("10.1.3.0", "255.255.255.0");

  Ipv4InterfaceContainer i4i5 = ipv4.Assign (d4d5);

  ipv4.SetBase ("10.1.4.0", "255.255.255.0");

  Ipv4InterfaceContainer i2i5 = ipv4.Assign (d2d5);

  ipv4.SetBase ("10.1.5.0", "255.255.255.0");

  Ipv4InterfaceContainer i3i5 = ipv4.Assign (d3d5);

  NS\_LOG\_INFO ("Enable static global routing.");

  //

  // Turn on global static routing so we can actually be routed across the network.

  //

  Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

  NS\_LOG\_INFO ("Create Applications.");

  // TCP connfection from N0 to N2

  uint16\_t sinkPort = 8080;

  Address sinkAddress (InetSocketAddress (i2i5.GetAddress (0), sinkPort)); // interface of n2

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

  ApplicationContainer sinkApps = packetSinkHelper.Install (c.Get (2)); //n2 as sink

  sinkApps.Start (Seconds (0.));

  sinkApps.Stop (Seconds (100.));

  Ptr<Socket> ns3TcpSocket = Socket::CreateSocket (c.Get (0), TcpSocketFactory::GetTypeId ()); //source at n0

  // Trace Congestion window

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

  // Create TCP application at n0

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

  app->Setup (ns3TcpSocket, sinkAddress, 1040, 100000, DataRate ("250Kbps"));

  c.Get (0)->AddApplication (app);

  app->SetStartTime (Seconds (1.));

  app->SetStopTime (Seconds (100.));

  // UDP connfection from N1 to N3

  uint16\_t sinkPort2 = 6;

  Address sinkAddress2 (InetSocketAddress (i3i5.GetAddress (0), sinkPort2)); // interface of n3

  PacketSinkHelper packetSinkHelper2 ("ns3::UdpSocketFactory", InetSocketAddress (Ipv4Address::GetAny (), sinkPort2));

  ApplicationContainer sinkApps2 = packetSinkHelper2.Install (c.Get (3)); //n3 as sink

  sinkApps2.Start (Seconds (0.));

  sinkApps2.Stop (Seconds (100.));

  Ptr<Socket> ns3UdpSocket = Socket::CreateSocket (c.Get (1), UdpSocketFactory::GetTypeId ()); //source at n1

  // Create UDP application at n1

  Ptr<MyApp> app2 = CreateObject<MyApp> ();

  app2->Setup (ns3UdpSocket, sinkAddress2, 1040, 100000, DataRate ("250Kbps"));

  c.Get (1)->AddApplication (app2);

  app2->SetStartTime (Seconds (20.));

  app2->SetStopTime (Seconds (100.));

// Increase UDP Rate

  Simulator::Schedule (Seconds(30.0), &IncRate, app2, DataRate("500kbps"));

  // Flow Monitor

  Ptr<FlowMonitor> flowmon;

  if (enableFlowMonitor)

    {

      FlowMonitorHelper flowmonHelper;

      flowmon = flowmonHelper.InstallAll ();

    }

//

// Now, do the actual simulation.

//

  NS\_LOG\_INFO ("Run Simulation.");

  Simulator::Stop (Seconds(100.0));

  Simulator::Run ();

  if (enableFlowMonitor)

    {

    flowmon->CheckForLostPackets ();

    flowmon->SerializeToXmlFile("lab-2.flowmon", true, true);

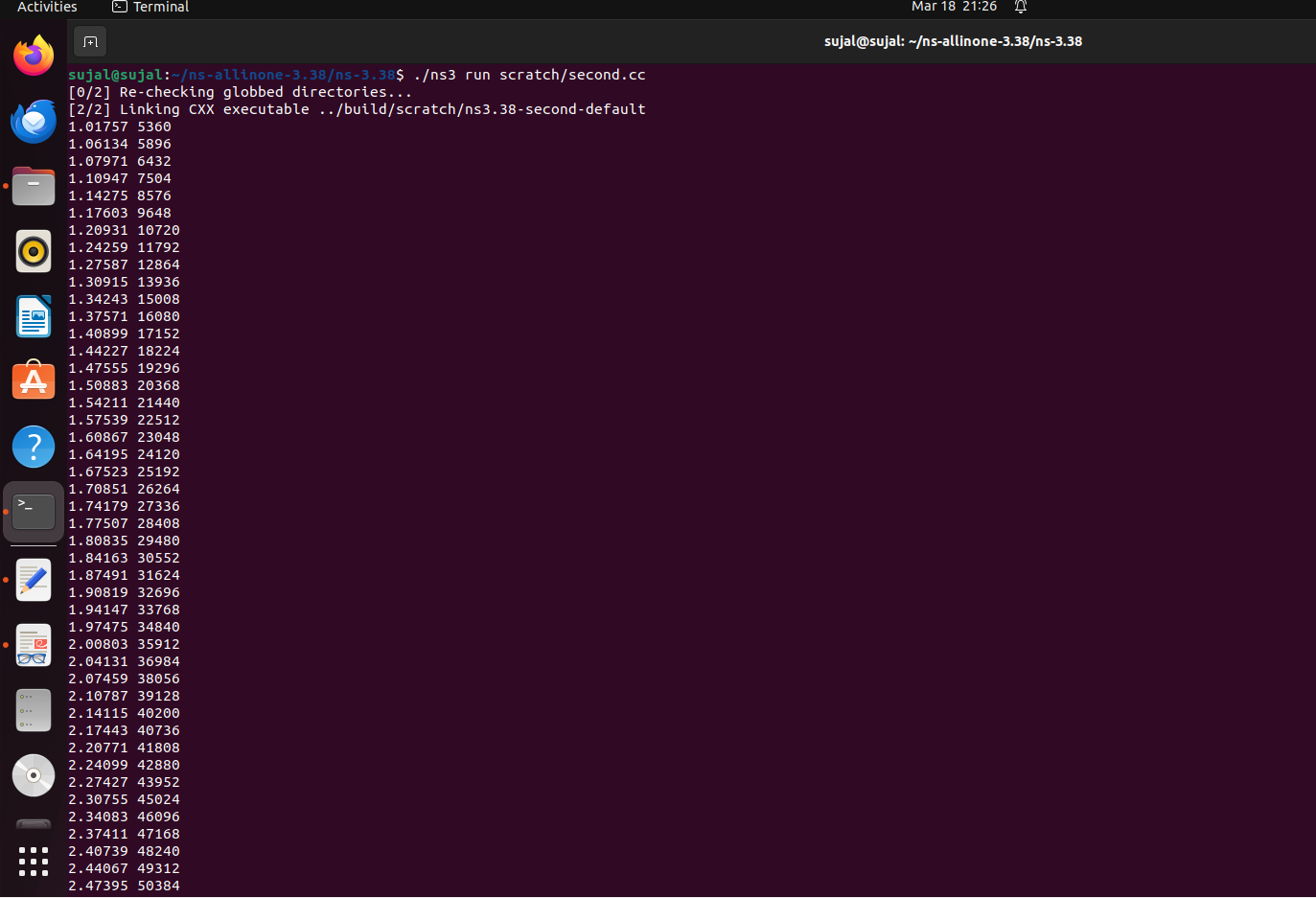
    }

  Simulator::Destroy ();

  NS\_LOG\_INFO ("Done.");

}

**OUTPUT:-**

****