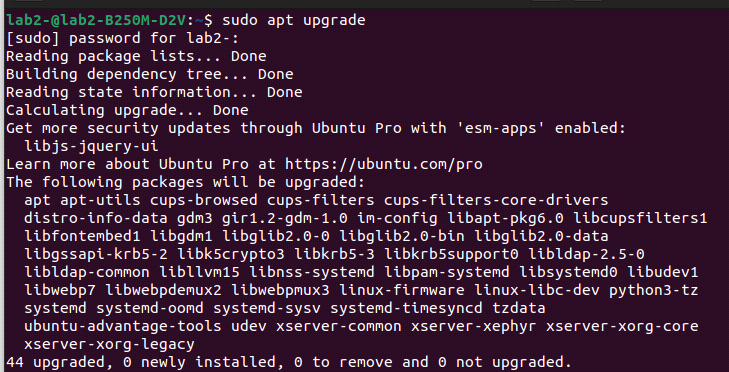
# Practical No: 1

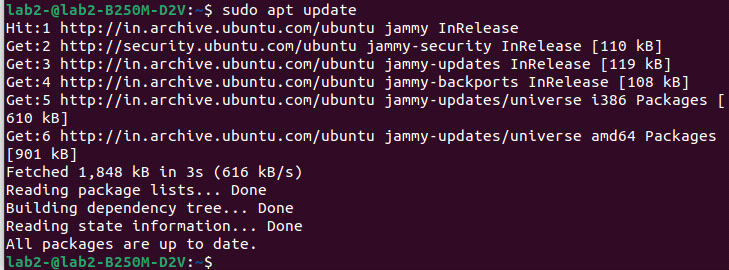
## Aim: Installation of NS-3 in Linux

## Steps:

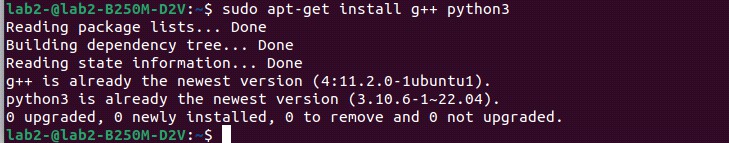
1. sudo apt upgrade



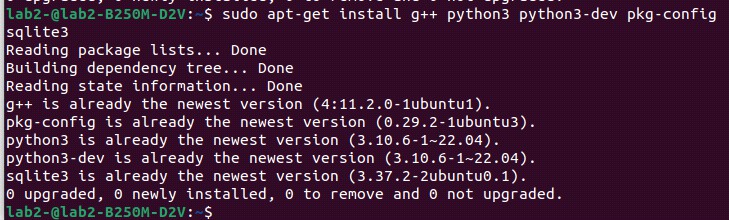
1. sudo apt update



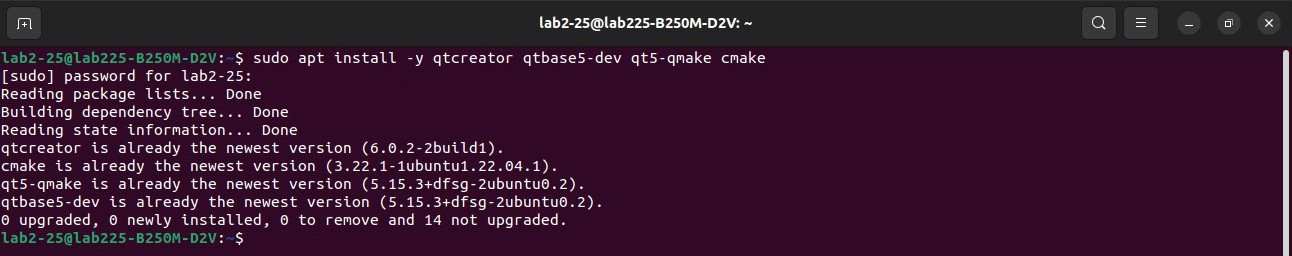
1. sudo apt-get install g++ python3

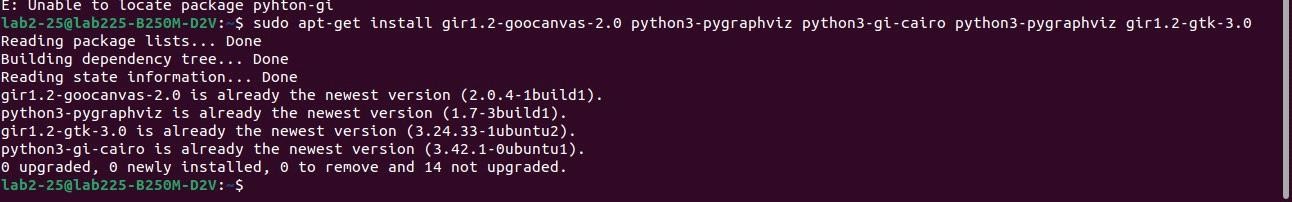


1. sudo apt-get install g++ python3python3-dev pkg-config sqlite3



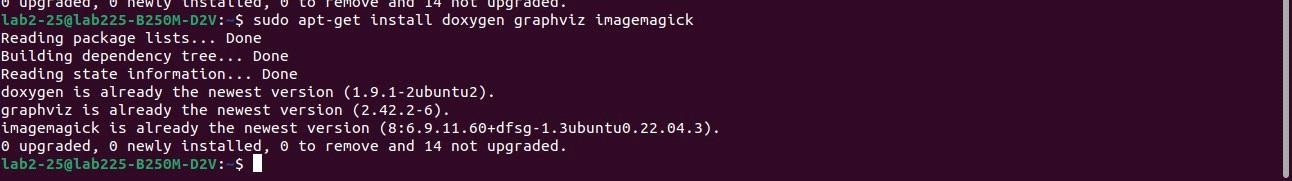
1. sudo apt install -y qtcreator qtbase5-dev qt5-qmake cmake



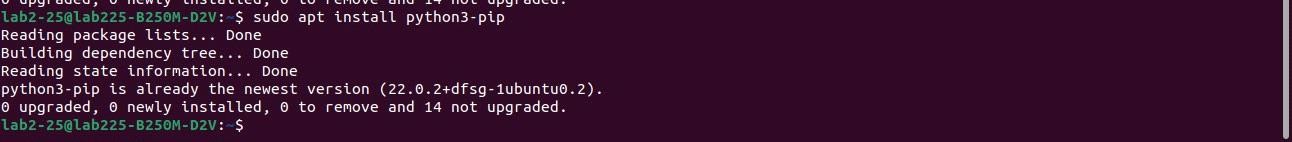
1. sudo apt-get install gir1.2-goocanvas-2.0 python3-pygraphviz python3-gi- cairo python3-pygraphviz gir1.2-gtk-3.0
2. sudo apt-get install gdb valgrind



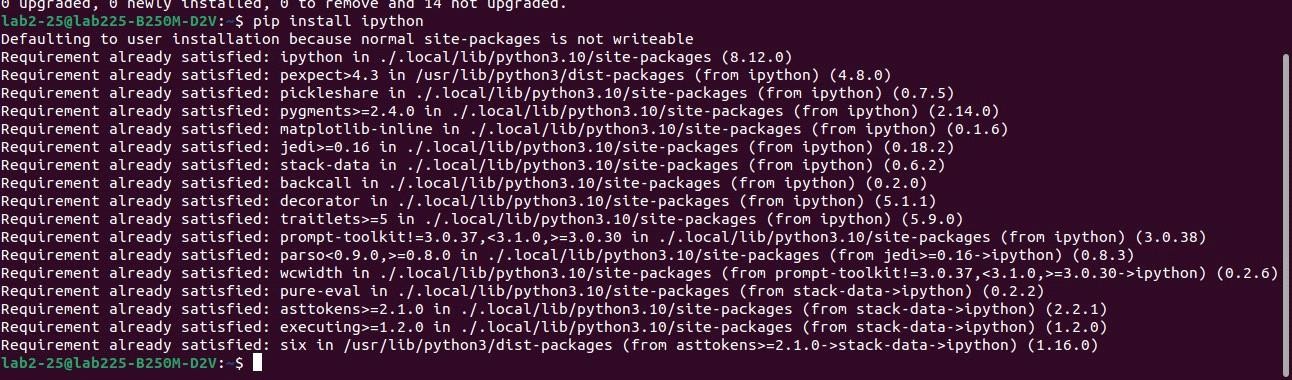
1. sudo apt-get install doxygen graphviz imagemagick



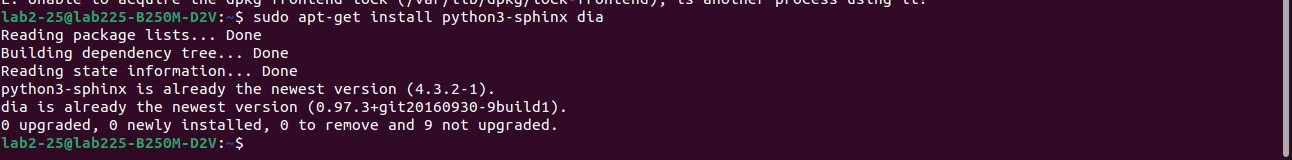
1. sudo apt install python3-pip



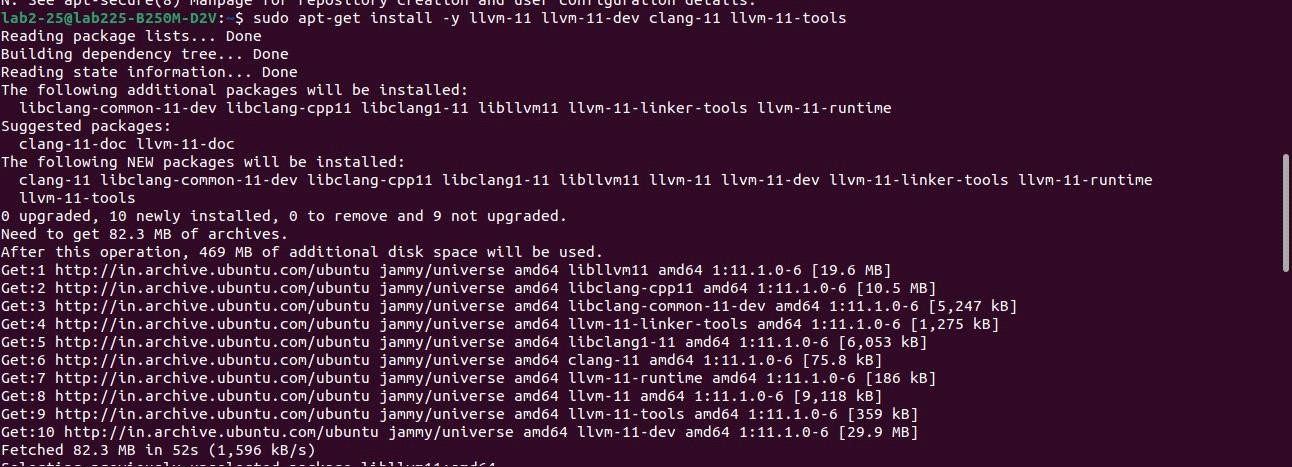
1. pip install ipython



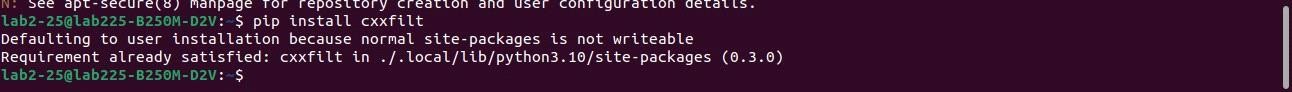
1. sudo apt-get install python3-sphinx dia



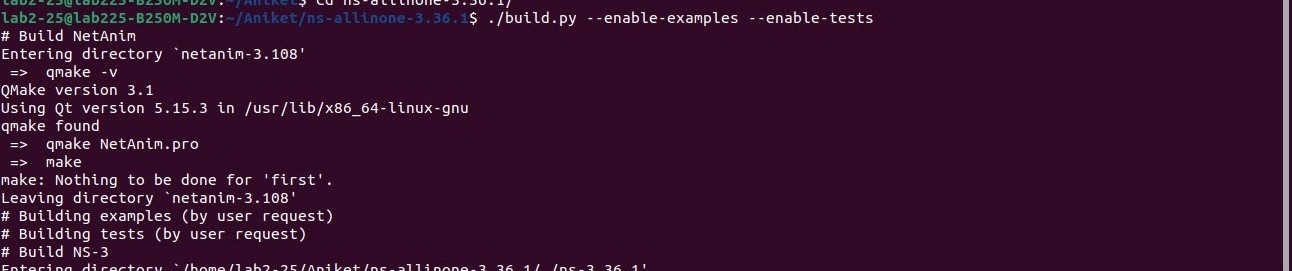
1. sudo apt-get install tcpdump
2. sudo apt-get install -y llvm-11 llvm-11-dev clang-11 llvm-11-tools

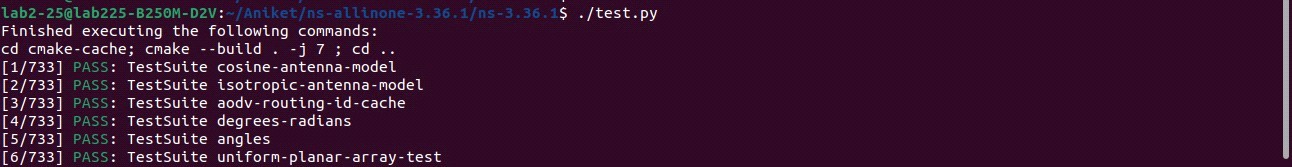


1. pip install cxxfilt



1. ./build.py--enable-examples–enable-tests



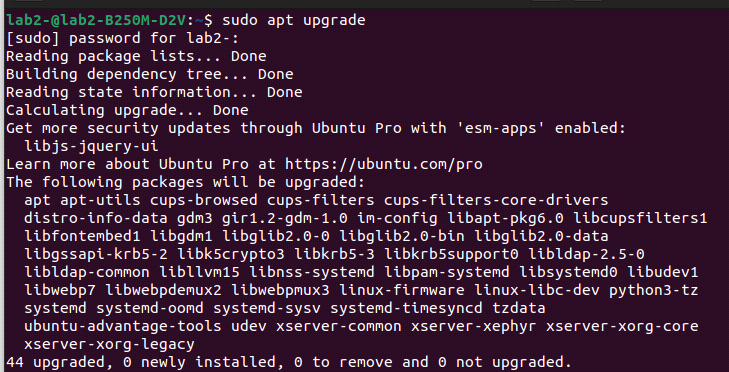
1. cdns-3.36.1
2. ./test.py

**Practical No: 2**

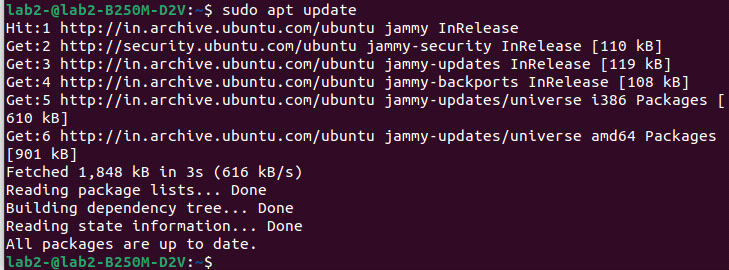
## Aim: Installation of NetAnim

**Steps:**

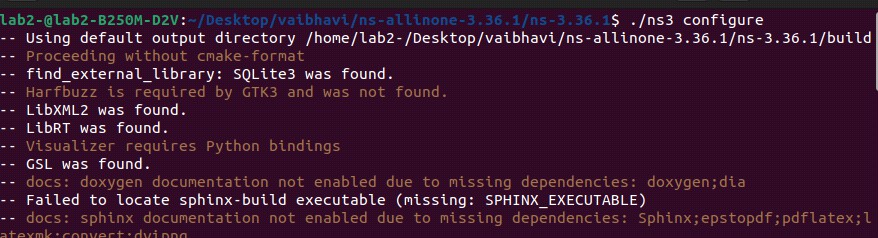
1. Sudo apt upgrade



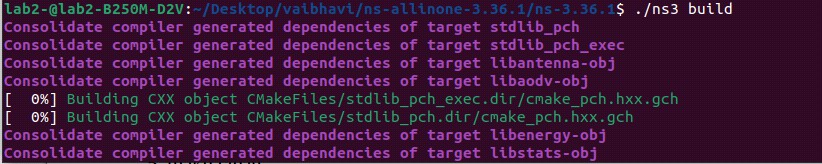
1. sudo apt update

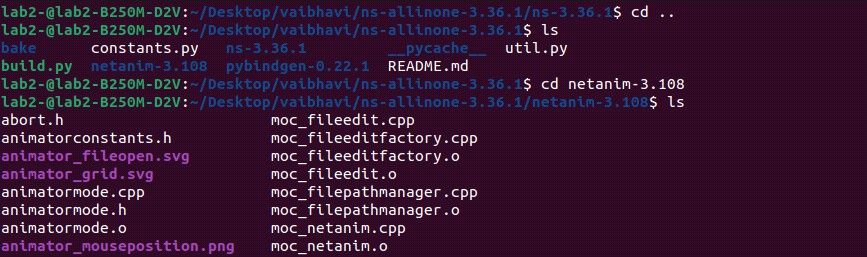


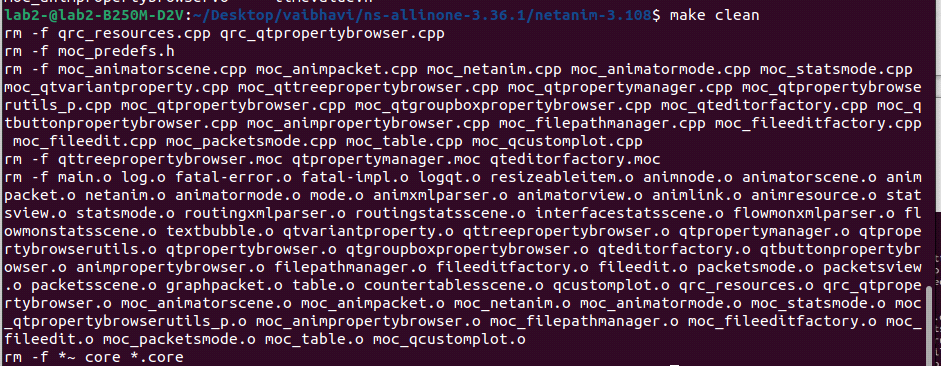
1. ./ns3 configure



1. ./ns3 build



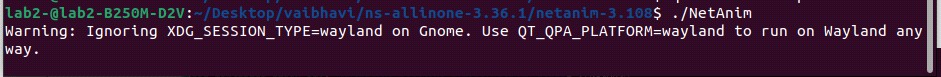
1. cd and ls
2. makeclean

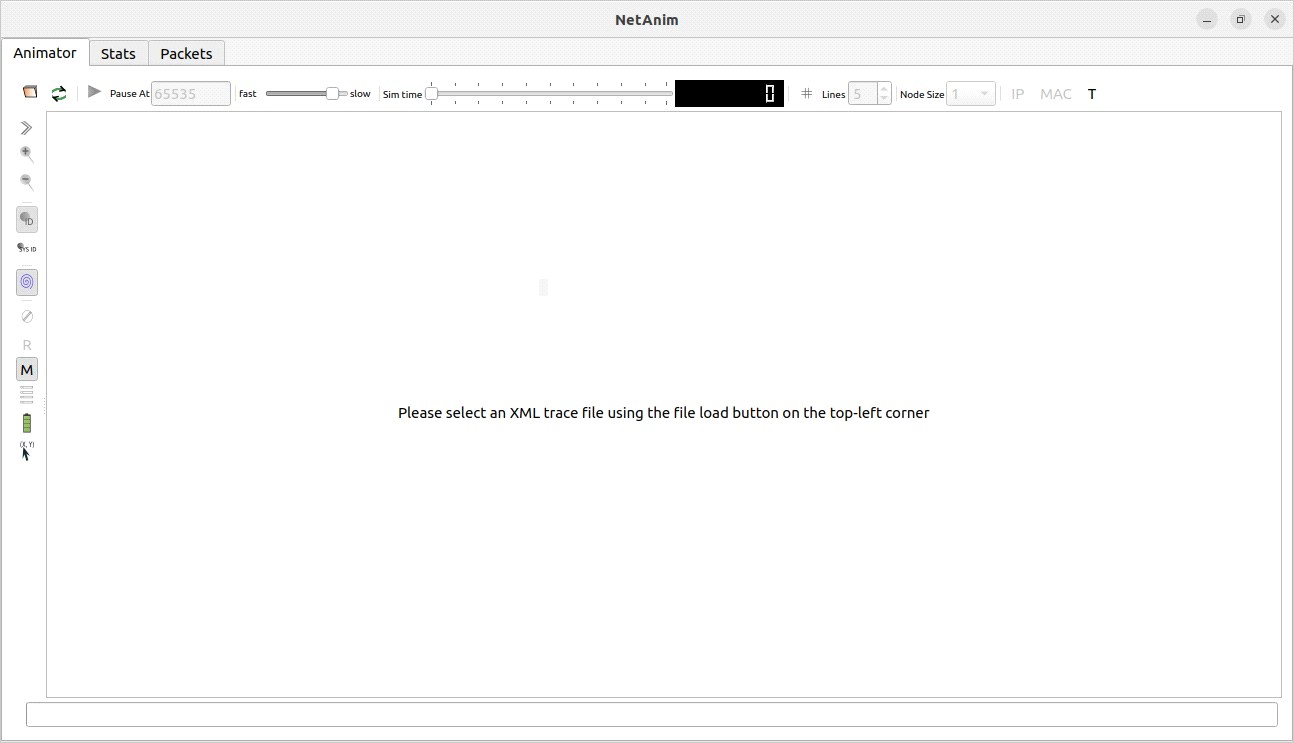


1. qmakeNetAnim.pro



1. ./NetAnim



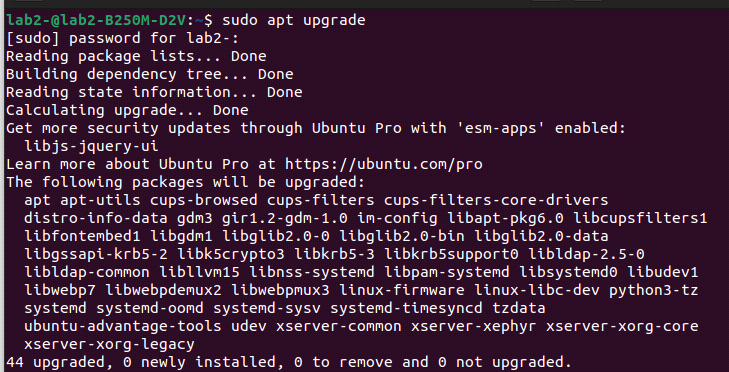


# Practical No: 3

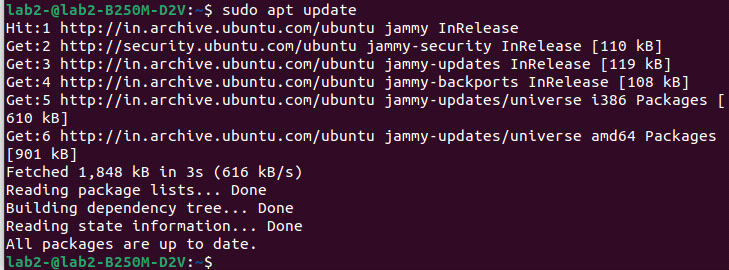
## Aim: Installation of Wireshark

**Steps:**

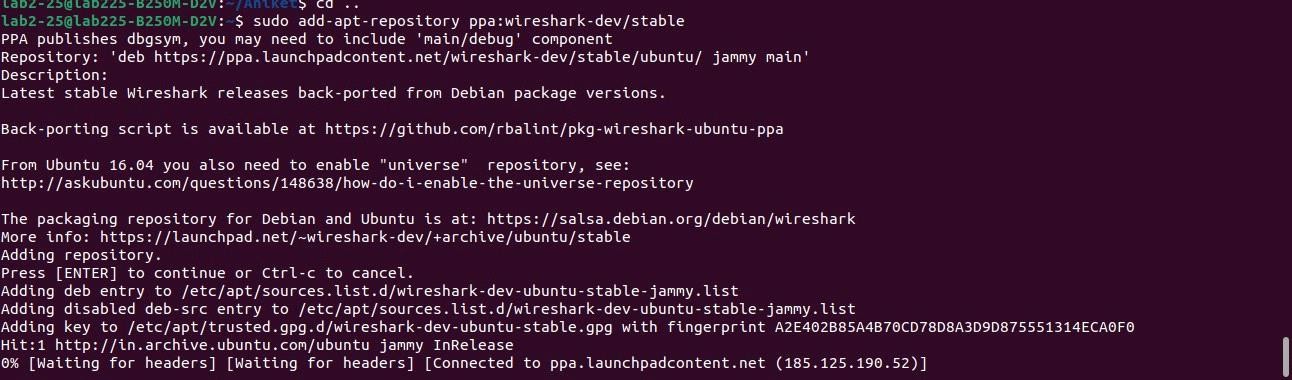
* 1. sudo apt upgrade



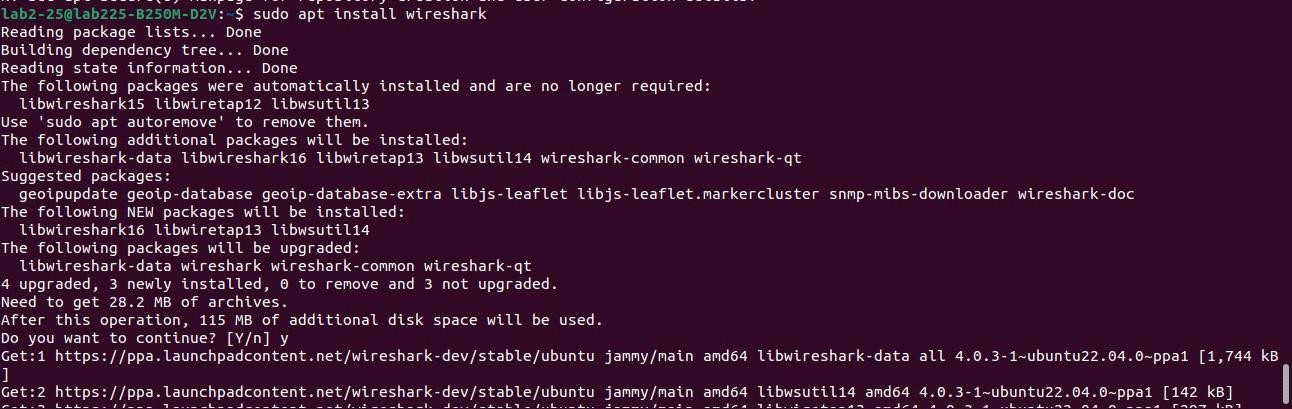
* 1. sudo apt update



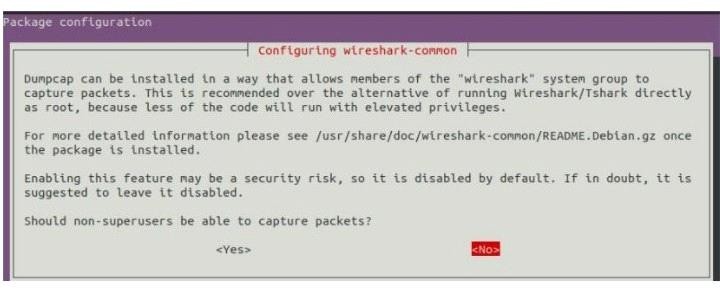
* 1. sudo add-apt-repository ppa:wireshark-dev/stable



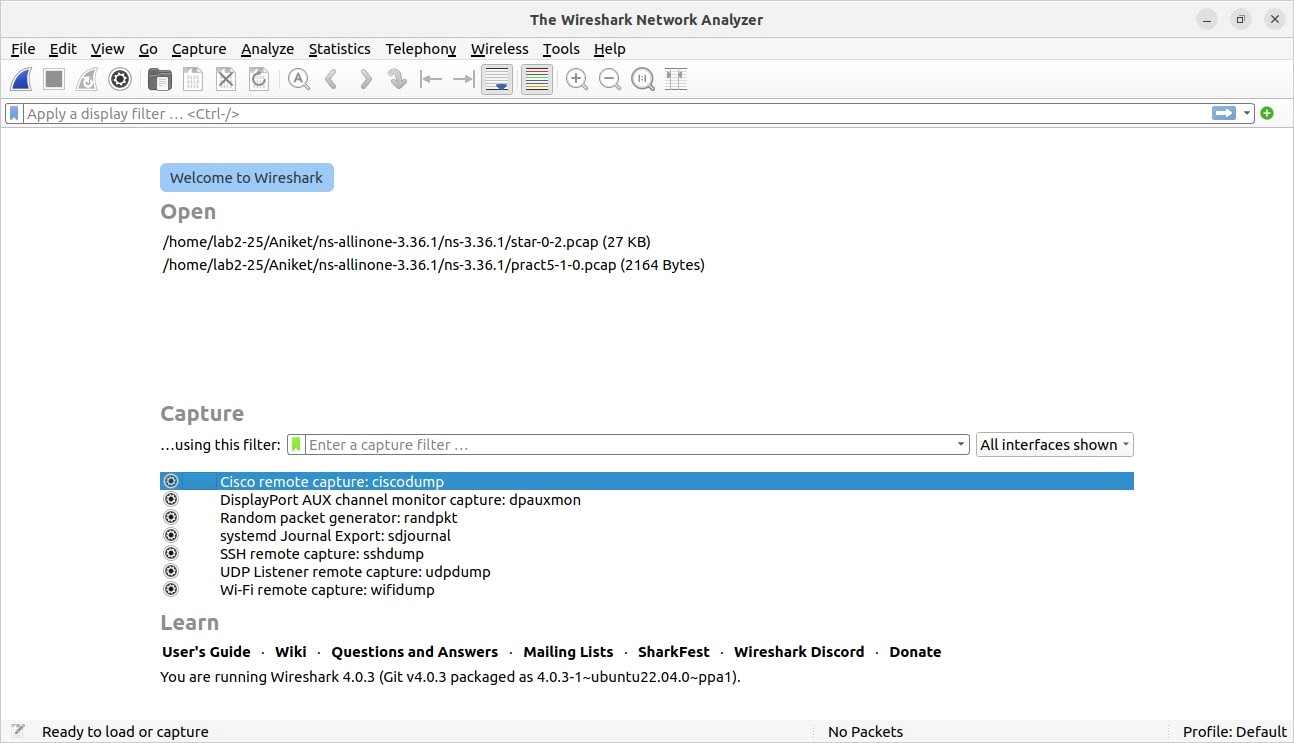
* 1. sudo apt install wireshark



* 1. configuring wireshark click yes







# Practical No: 4

## Aim: Program to simulate traffic between two nodes Code:

**Point.cc**

#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 "ns3/netanim-module.h" #include "ns3/mobility-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));

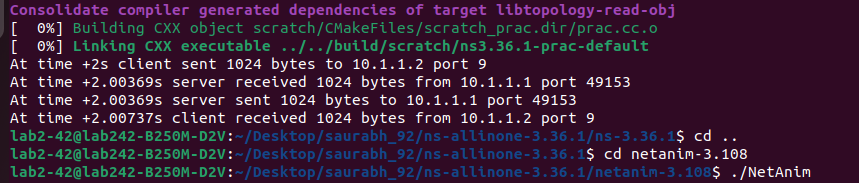
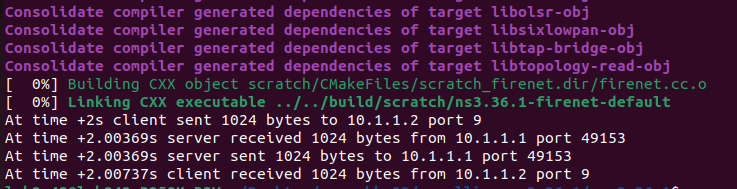
clientApps.Stop (Seconds (10.0)); MobilityHelper mobility;

mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel"); mobility.Install(nodes);

AnimationInterface anim("first.xml"); AnimationInterface::SetConstantPosition(nodes.Get(0),10,25); AnimationInterface::SetConstantPosition(nodes.Get(1),40,25); anim.EnablePacketMetadata(true); pointToPoint.EnablePcapAll("point");

Simulator::Run (); Simulator::Destroy (); return 0; }

## Output:



**Practical No: 4.1 Aim: Point to point topology with ns3 & Net-Anim. Code:**

#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 "ns3/netanim-module.h" #include "ns3/mobility-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 (

F ILE );

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)); MobilityHelper mobility;

mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel"); mobility.Install(nodes);

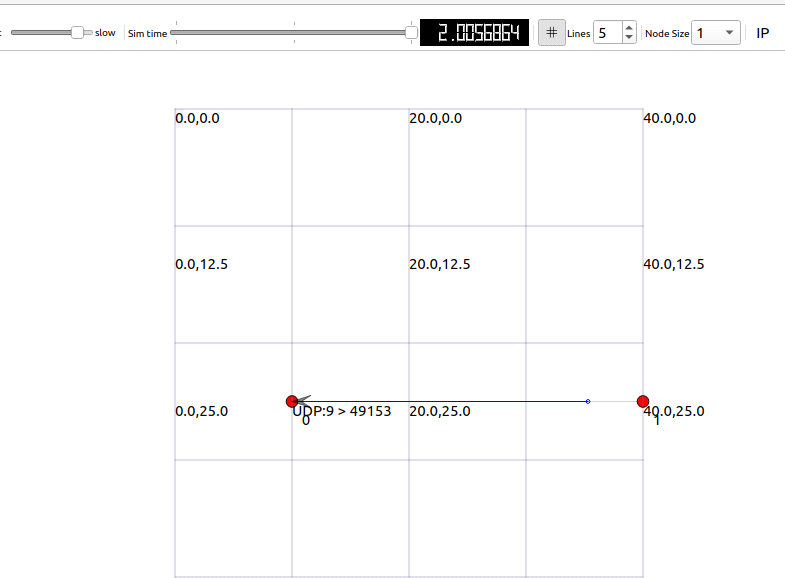
AnimationInterface anim("first.xml"); AnimationInterface::SetConstantPosition(nodes.Get(0),10,25); AnimationInterface::SetConstantPosition(nodes.Get(1),40,25); anim.EnablePacketMetadata(true); pointToPoint.EnablePcapAll("pract2Mana");

Simulator::Run ();

Simulator::Destro y (); return 0; }

**Output**:





# Practical No: 5

## Aim: Program to simulate star topology

**Code:**

#include "ns3/network-module.h" #include "ns3/internet-module.h" #include "ns3/point-to-point-star.h" #include "ns3/applications-module.h" #include "ns3/netanim-module.h" #include "ns3/mobility-module.h" #include "ns3/onoff-application.h" using namespace ns3;

//NS\_LOG\_COMPONENT\_DEFINE ("Star");

NS\_LOG\_COMPONENT\_DEFINE ("StarExample"); int

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

{

// Set up some default values for the simulation. Config::SetDefault ("ns3::OnOffApplication::PacketSize", UintegerValue (137));

// ??? try and stick 15kb/s into the data rate

Config::SetDefault ("ns3::OnOffApplication::DataRate", StringValue ("14kb/s"));

// Default number of nodes in the star. Overridable by command line argument. uint32\_t nSpokes = 8;

CommandLine cmd ( FILE );

cmd.AddValue ("nSpokes", "Number of nodes to place in the star",nSpokes); cmd.Parse (argc, argv);

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

NS\_LOG\_INFO ("Build star topology."); PointToPointHelper pointToPoint;

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

NS\_LOG\_INFO ("Install internet stack on all nodes."); InternetStackHelper internet;

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

star.AssignIpv4Addresses (Ipv4AddressHelper ("10.1.1.0",

"255.255.255.0"));

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

// Create a packet sink on the star "hub" to receive packets. uint16\_t port = 50000;

Address hubLocalAddress (InetSocketAddress (Ipv4Address::GetAny (), port));

PacketSinkHelper packetSinkHelper ("ns3::TcpSocketFactory", hubLocalAddress);

ApplicationContainer hubApp = packetSinkHelper.Install (star.GetHub ());

hubApp.Start (Seconds (1.0));

hubApp.Stop (Seconds (10.0));

// Create OnOff applications to send TCP to the hub, one on each spokenode. OnOffHelper onOffHelper ("ns3::TcpSocketFactory", Address ()); onOffHelper.SetAttribute ("OnTime", StringValue("ns3::ConstantRandomVariable[Constant=1]")); onOffHelper.SetAttribute ("OffTime", StringValue("ns3::ConstantRandomVariable[Constant=0]")); ApplicationContainer spokeApps;

for (uint32\_t i = 0; i < star.SpokeCount (); ++i)

{

AddressValue remoteAddress (InetSocketAddress (star.GetHubIpv4Address (i), port)); onOffHelper.SetAttribute ("Remote", remoteAddress);

spokeApps.Add (onOffHelper.Install (star.GetSpokeNode (i)));

}

spokeApps.Start (Seconds (1.0));

spokeApps.Stop (Seconds (10.0)); NS\_LOG\_INFO ("Enable static global routing.");

// Turn on global static routing so we can actually be routed across thestar. Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

NS\_LOG\_INFO ("Enable pcap tracing.");

//

// Do pcap tracing on all point-to-point devices on all nodes.

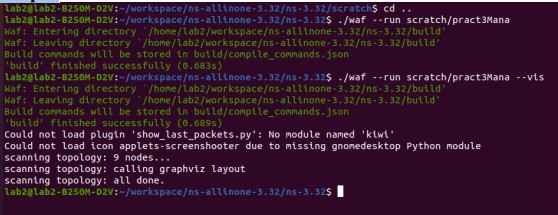
//

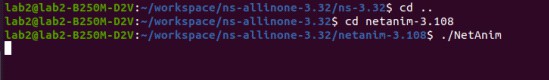
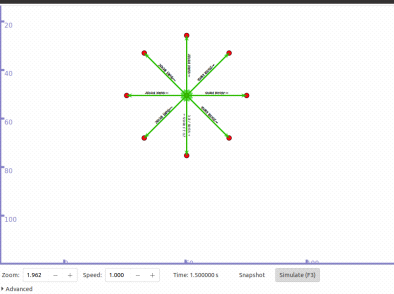
pointToPoint.EnablePcapAll ("star"); star.BoundingBox(1,1,100,100); AnimationInterface anim("star.xml"); NS\_LOG\_INFO ("Run Simulation."); Simulator::Run (); Simulator::Destroy (); NS\_LOG\_INFO ("Done.");

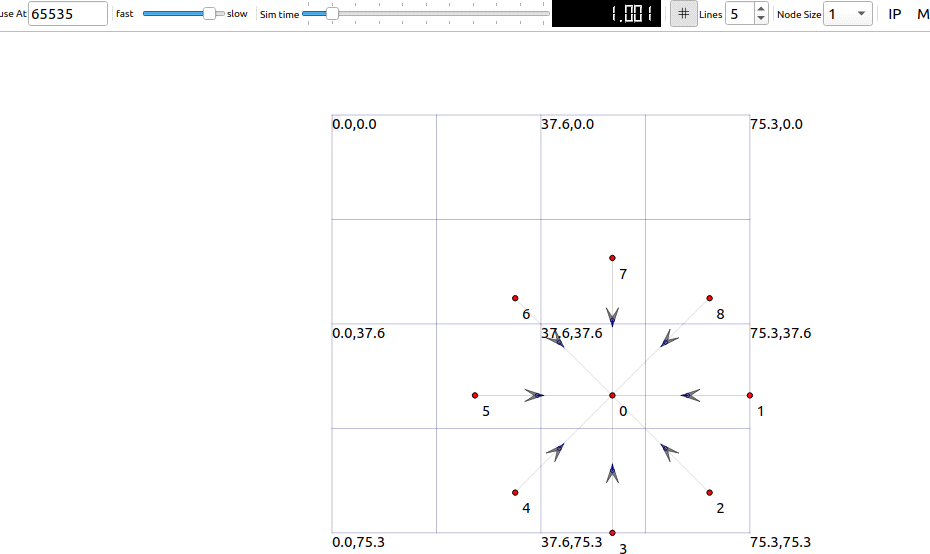
return 0;

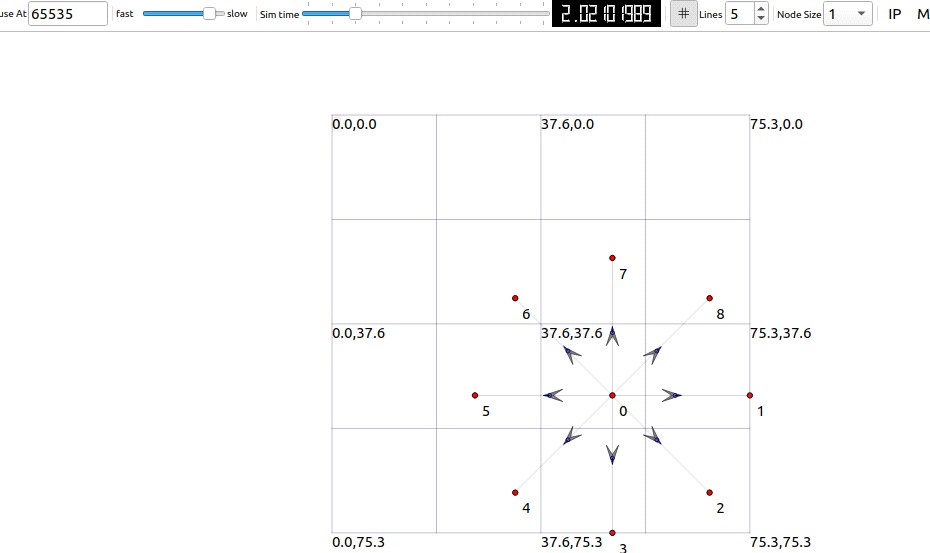
}

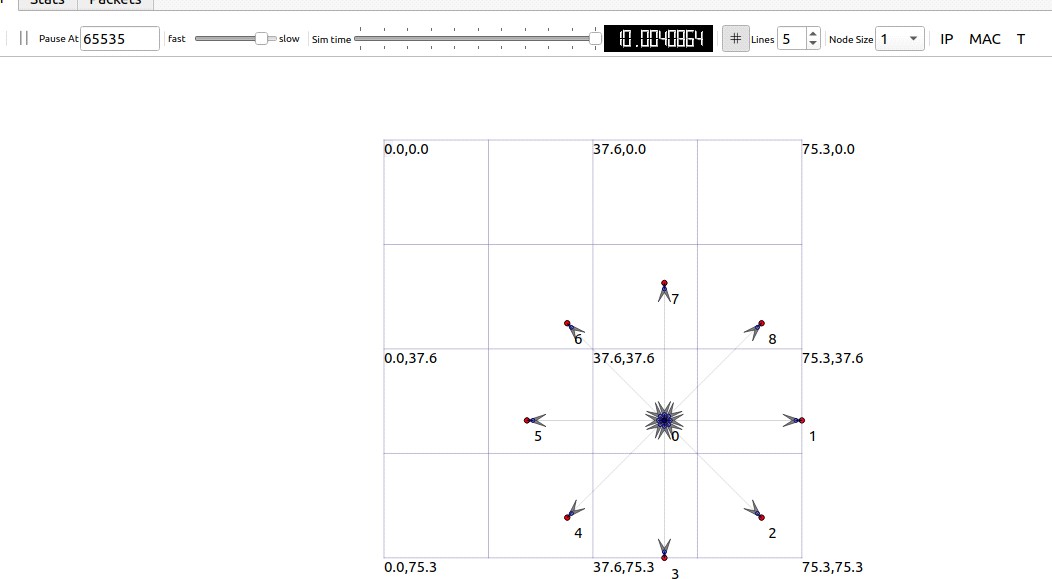
**Output:**











# Practical No: 6

## Aim: Program to simulate bus topology Code:

**pract7.cc**

#include "ns3/core-module.h" #include "ns3/network-module.h" #include "ns3/csma-module.h" #include "ns3/internet-module.h"

#include "ns3/point-to-point-module.h" #include "ns3/applications-module.h" #include "ns3/ipv4-global-routing-helper.h"

// Default Network Topology

//

//

//10.1.1.0

// n0 n1 n2 n3 n4

//

//

//

//point-to-point | | |

//|

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

//LAN 10.1.2.0

using namespace ns3;

NS\_LOG\_COMPONENT\_DEFINE ("SecondScriptExample"); int

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

{

bool verbose = true; uint32\_t nCsma = 5;

CommandLine cmd ( FILE );

cmd.AddValue ("nCsma", "Number of \"extra\" CSMA nodes/devices", nCsma); cmd.AddValue ("verbose", "Tell echo applications to log if true", verbose); cmd.Parse (argc,argv);

if (verbose)

{

LogComponentEnable ("UdpEchoClientApplication", LOG\_LEVEL\_INFO); LogComponentEnable ("UdpEchoServerApplication", LOG\_LEVEL\_INFO); nCsma = nCsma == 0 ? 1 : nCsma;

NodeContainer p2pNodes; p2pNodes.Create (2); NodeContainer csmaNodes; csmaNodes.Add (p2pNodes.Get (1)); csmaNodes.Create (nCsma); PointToPointHelper pointToPoint;

pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("10Mbps")); pointToPoint.SetChannelAttribute ("Delay", StringValue ("5ms"));

NetDeviceContainer p2pDevices;

p2pDevices = pointToPoint.Install (p2pNodes); CsmaHelper csma;

csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps")); csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560))); NetDeviceContainer csmaDevices;

csmaDevices = csma.Install (csmaNodes); InternetStackHelper stack;

stack.Install (p2pNodes.Get (0)); stack.Install (csmaNodes); Ipv4AddressHelper address;

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

Ipv4InterfaceContainer p2pInterfaces; p2pInterfaces = address.Assign (p2pDevices); address.SetBase ("10.1.2.0", "255.255.255.0");

Ipv4InterfaceContainer csmaInterfaces; csmaInterfaces = address.Assign (csmaDevices); UdpEchoServerHelper echoServer (9);

ApplicationContainer serverApps = echoServer.Install (csmaNodes.Get (nCsma)); serverApps.Start (Seconds (1.0));

serverApps.Stop (Seconds (10.0));

UdpEchoClientHelper echoClient (csmaInterfaces.GetAddress (nCsma), 9); echoClient.SetAttribute ("MaxPackets", UintegerValue (1)); echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0))); echoClient.SetAttribute ("PacketSize", UintegerValue (1024)); ApplicationContainer clientApps = echoClient.Install (p2pNodes.Get (0)); clientApps.Start (Seconds (2.0));

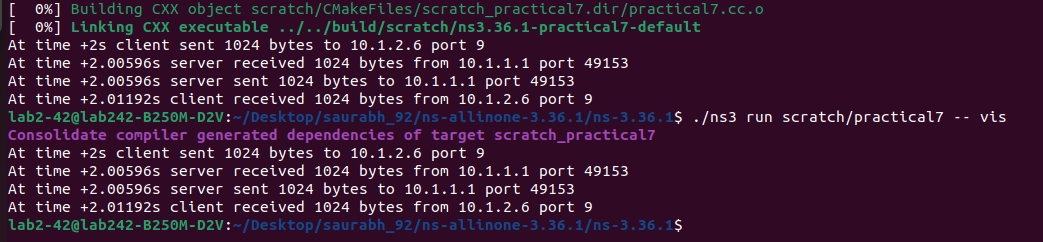
clientApps.Stop (Seconds (10.0)); Ipv4GlobalRoutingHelper::PopulateRoutingTables (); pointToPoint.EnablePcapAll ("pract2Mana"); csma.EnablePcap ("pract2Mana", csmaDevices.Get (1), true); Simulator::Run ();

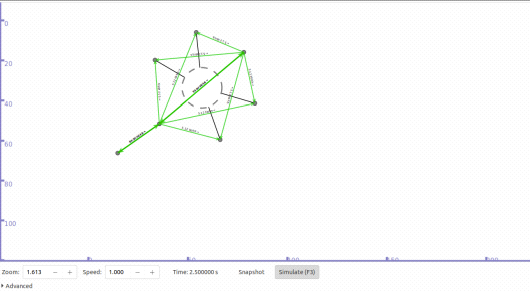
Simulator::Destroy (); return 0;

}

}

**Output:**





# Practical No: 7

## Aim: Program to simulate mesh topology Code:

**practmesh.cc**

#include "ns3/applications-module.h" #include "ns3/core-module.h" #include "ns3/internet-module.h" #include "ns3/mesh-helper.h" #include "ns3/mesh-module.h" #include "ns3/mobility-module.h" #include "ns3/network-module.h" #include "ns3/yans-wifi-helper.h" #include <fstream>

#include <iostream> #include <sstream> using namespace ns3;

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

// Declaring these variables outside of main() for use in trace sinks uint32\_t g\_udpTxCount = 0; //!< Rx packet counter.

uint32\_t g\_udpRxCount = 0; //!< Tx packet counter.

/\*\*

* Transmission trace sink.

\*

* \param p The sent packet.

\*/ void

TxTrace(Ptr<const Packet> p)

{

NS\_LOG\_DEBUG("Sent " << p->GetSize() << " bytes"); g\_udpTxCount++;

}

/\*\*

* Reception trace sink,
* ​
* \param p The received packet.

\*/ void

RxTrace(Ptr<const Packet> p)

{

NS\_LOG\_DEBUG("Received " << p->GetSize() << " bytes"); g\_udpRxCount++;

}

/\*\*

* \ingroup mesh
* \brief MeshTest class

\*/

class MeshTest

{

public:

/// Init test

MeshTest();

/\*\*

* Configure test from command line arguments
* ​
* \param argc command line argument count
* \param argv command line arguments

\*/

void Configure(int argc, char\*\* argv);

/\*\*

* Run test
* \returns the test status

\*/

int Run(); private:

int m\_xSize; ///< X size int m\_ySize; ///< Y size double m\_step; ///< step

double m\_randomStart; ///< random start double m\_totalTime; ///< total time

double m\_packetInterval; ///< packet interval uint16\_t m\_packetSize; ///< packet size uint32\_t m\_nIfaces; ///< number interfaces bool m\_chan; ///< channel

bool m\_pcap; ///< PCAP bool m\_ascii; ///< ASCII std::string m\_stack; ///< stack std::string m\_root; ///< root

/// List of network nodes NodeContainer nodes;

/// List of all mesh point devices NetDeviceContainer meshDevices;

/// Addresses of interfaces: Ipv4InterfaceContainer interfaces;

/// MeshHelper. Report is not static methods MeshHelper mesh;

private:

/// Create nodes and setup their mobility void CreateNodes();

/// Install internet m\_stack on nodes void InstallInternetStack();

/// Install applications void InstallApplication();

/// Print mesh devices diagnostics void Report();

};

MeshTest::MeshTest()

: m\_xSize(3), m\_ySize(3), m\_step(50.0), m\_randomStart(0.1), m\_totalTime(100.0), m\_packetInterval(1), m\_packetSize(1024), m\_nIfaces(1),

m\_chan(true), m\_pcap(false), m\_ascii(false),

m\_stack("ns3::Dot11sStack"), m\_root("ff:ff:ff:ff:ff:ff")

{

}

void

MeshTest::Configure(int argc, char\* argv[])

{

CommandLine cmd( FILE );

cmd.AddValue("x-size", "Number of nodes in a row grid", m\_xSize); cmd.AddValue("y-size", "Number of rows in a grid", m\_ySize); cmd.AddValue("step", "Size of edge in our grid (meters)", m\_step);

// Avoid starting all mesh nodes at the same time (beacons may collide) cmd.AddValue("start", "Maximum random start delay for beacon jitter (sec)", m\_randomStart);

cmd.AddValue("time", "Simulation time (sec)", m\_totalTime); cmd.AddValue("packet-interval", "Interval between packets in UDP ping (sec)", m\_packetInterval);

cmd.AddValue("packet-size", "Size of packets in UDP ping (bytes)", m\_packetSize); cmd.AddValue("interfaces", "Number of radio interfaces used by each mesh point", m\_nIfaces);

cmd.AddValue("channels", "Use different frequency channels for different interfaces", m\_chan);

cmd.AddValue("pcap", "Enable PCAP traces on interfaces", m\_pcap); cmd.AddValue("ascii", "Enable Ascii traces on interfaces", m\_ascii); cmd.AddValue("stack", "Type of protocol stack. ns3::Dot11sStack by default", m\_stack); cmd.AddValue("root", "Mac address of root mesh point in HWMP", m\_root); cmd.Parse(argc, argv);

NS\_LOG\_DEBUG("Grid:" << m\_xSize << "\*" << m\_ySize); NS\_LOG\_DEBUG("Simulation time: " << m\_totalTime << " s"); if (m\_ascii)

{

PacketMetadata::Enable();

}

}

void MeshTest::CreateNodes()

{

/\*

* Create m\_ySize\*m\_xSize stations to form a grid topology

\*/

nodes.Create(m\_ySize \* m\_xSize);

// Configure YansWifiChannel YansWifiPhyHelper wifiPhy;

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

/\*

* Create mesh helper and set stack installer to it
* Stack installer creates all needed protocols and install them to
* mesh point device

\*/

mesh = MeshHelper::Default();

if (!Mac48Address(m\_root.c\_str()).IsBroadcast())

{

mesh.SetStackInstaller(m\_stack, "Root", Mac48AddressValue(Mac48Address(m\_root.c\_str())));

}

else

{

// If root is not set, we do not use "Root" attribute, because it

// is specified only for 11s mesh.SetStackInstaller(m\_stack);

}

if (m\_chan)

{

mesh.SetSpreadInterfaceChannels(MeshHelper::SPREAD\_CHANNELS);

}

else

{

mesh.SetSpreadInterfaceChannels(MeshHelper::ZERO\_CHANNEL);

}

mesh.SetMacType("RandomStart", TimeValue(Seconds(m\_randomStart)));

// Set number of interfaces - default is single-interface mesh point mesh.SetNumberOfInterfaces(m\_nIfaces);

// Install protocols and return container if MeshPointDevices meshDevices = mesh.Install(wifiPhy, nodes);

// AssignStreams can optionally be used to control random variable streams mesh.AssignStreams(meshDevices, 0);

// Setup mobility - static grid topology MobilityHelper mobility;

mobility.SetPositionAllocator("ns3::GridPositionAllocator",

"MinX",

DoubleValue(0.0), "MinY",

DoubleValue(0.0), "DeltaX",

DoubleValue(m\_step), "DeltaY", DoubleValue(m\_step), "GridWidth", UintegerValue(m\_xSize), "LayoutType", StringValue("RowFirst"));

mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel"); mobility.Install(nodes);

if (m\_pcap)

{

wifiPhy.EnablePcapAll(std::string("mp"));

}

if (m\_ascii)

{

AsciiTraceHelper ascii; wifiPhy.EnableAsciiAll(ascii.CreateFileStream("mesh.tr"));

}

}

void MeshTest::InstallInternetStack()

{

InternetStackHelper internetStack; internetStack.Install(nodes); Ipv4AddressHelper address; address.SetBase("10.1.1.0", "255.255.255.0");

interfaces = address.Assign(meshDevices);

}

void MeshTest::InstallApplication()

{

uint16\_t portNumber = 9;

UdpEchoServerHelper echoServer(portNumber); uint16\_t sinkNodeId = m\_xSize \* m\_ySize - 1;

ApplicationContainer serverApps = echoServer.Install(nodes.Get(sinkNodeId)); serverApps.Start(Seconds(1.0));

serverApps.Stop(Seconds(m\_totalTime + 1));

UdpEchoClientHelper echoClient(interfaces.GetAddress(sinkNodeId), portNumber); echoClient.SetAttribute("MaxPackets",

UintegerValue((uint32\_t)(m\_totalTime \* (1 / m\_packetInterval)))); echoClient.SetAttribute("Interval", TimeValue(Seconds(m\_packetInterval))); echoClient.SetAttribute("PacketSize", UintegerValue(m\_packetSize)); ApplicationContainer clientApps = echoClient.Install(nodes.Get(0)); Ptr<UdpEchoClient> app = clientApps.Get(0)->GetObject<UdpEchoClient>(); app->TraceConnectWithoutContext("Tx", MakeCallback(&TxTrace));

app->TraceConnectWithoutContext("Rx", MakeCallback(&RxTrace)); clientApps.Start(Seconds(1.0));

clientApps.Stop(Seconds(m\_totalTime + 1.5));

}

int MeshTest::Run()

{

CreateNodes(); InstallInternetStack(); InstallApplication();

Simulator::Schedule(Seconds(m\_totalTime), &MeshTest::Report, this); Simulator::Stop(Seconds(m\_totalTime + 2));

Simulator::Run(); Simulator::Destroy();

std::cout << "UDP echo packets sent: " << g\_udpTxCount << " received: " << g\_udpRxCount

<< std::endl; return 0;

}

void MeshTest::Report()

{

unsigned n(0);

for (NetDeviceContainer::Iterator i = meshDevices.Begin(); i != meshDevices.End(); ++i,

++n)

{

std::ostringstream os;

os<< "mp-report-" << n << ".xml";

std::cerr << "Printing mesh point device #" << n <<" diagnostics to " << os.str() << "\n"; std::ofstream of;

of.open(os.str().c\_str()); if (!of.is\_open())

{

std::cerr << "Error: Can't open file " <<os.str() << "\n"; return;

}

mesh.Report(\*i, of); of.close();

}

}

int

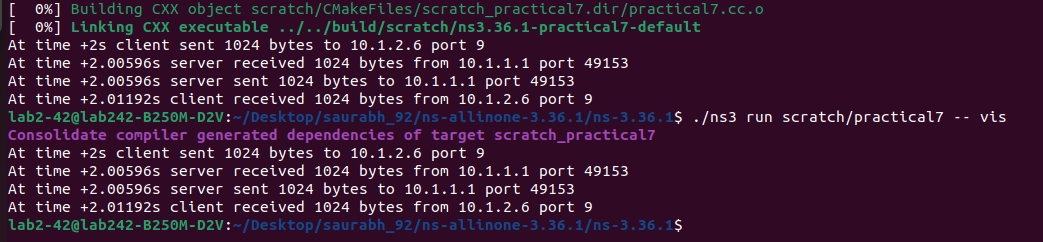
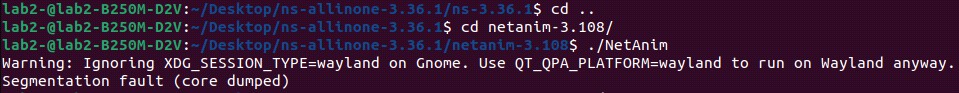
main(int argc, char\* argv[])

{

MeshTest t; t.Configure(argc, argv); return t.Run();

}

**Output:**



# Practical No: 8

## Aim: Program to simulate hybrid topology Code:

**hybrid.cc**

#include "ns3/core-module.h"

#include "ns3/point-to-point-module.h" #include "ns3/network-module.h" #include "ns3/applications-module.h" #include "ns3/mobility-module.h" #include "ns3/csma-module.h" #include "ns3/internet-module.h" #include "ns3/yans-wifi-helper.h" #include "ns3/ssid.h"

// Default Network Topology

//

// Wifi 10.1.3.0

// AP

// \* \* \* \*

// | | | | 10.1.1.0

// n5 n6 n7 n0 n1 n2 n3 n4

// point-to-point | | | |

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

// LAN 10.1.2.0

using namespace ns3;

NS\_LOG\_COMPONENT\_DEFINE ("ThirdScriptExample"); int

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

{

bool verbose = true; uint32\_t nCsma = 3; uint32\_t nWifi = 3; bool tracing = false;

CommandLine cmd ( FILE );

cmd.AddValue ("nCsma", "Number of \"extra\" CSMA nodes/devices", nCsma); cmd.AddValue ("nWifi", "Number of wifi STA devices", nWifi); cmd.AddValue ("verbose", "Tell echo applications to log if true", verbose); cmd.AddValue ("tracing", "Enable pcap tracing", tracing);

cmd.Parse (argc,argv);

// The underlying restriction of 18 is due to the grid position

// allocator's configuration; the grid layout will exceed the

// bounding box if more than 18 nodes are provided. if (nWifi > 18)

{

std::cout << "nWifi should be 18 or less; otherwise grid layout exceeds the bounding box" << std::endl;

return 1;

}

if (verbose)

{

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

}

NodeContainer p2pNodes; p2pNodes.Create (2); PointToPointHelper pointToPoint;

pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps")); pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms")); NetDeviceContainer p2pDevices;

p2pDevices = pointToPoint.Install (p2pNodes); NodeContainer csmaNodes;

csmaNodes.Add (p2pNodes.Get (1)); csmaNodes.Create (nCsma); CsmaHelper csma;

csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps")); csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560))); NetDeviceContainer csmaDevices;

csmaDevices = csma.Install (csmaNodes); NodeContainer wifiStaNodes; wifiStaNodes.Create (nWifi);

NodeContainer wifiApNode = p2pNodes.Get (0); YansWifiChannelHelper channel = YansWifiChannelHelper::Default (); YansWifiPhyHelper phy;

phy.SetChannel (channel.Create ()); WifiMacHelper mac;

Ssid ssid = Ssid ("ns-3-ssid"); WifiHelper wifi; NetDeviceContainer staDevices; mac.SetType ("ns3::StaWifiMac", "Ssid", SsidValue (ssid),

"ActiveProbing", BooleanValue (false)); staDevices = wifi.Install (phy, mac, wifiStaNodes);

NetDeviceContainer apDevices; mac.SetType ("ns3::ApWifiMac", "Ssid", SsidValue (ssid));

apDevices = wifi.Install (phy, mac, wifiApNode); MobilityHelper mobility;

mobility.SetPositionAllocator ("ns3::GridPositionAllocator", "MinX", DoubleValue (0.0),

"MinY", DoubleValue (0.0),

"DeltaX", DoubleValue (5.0),

"DeltaY", DoubleValue (10.0),

"GridWidth", UintegerValue (3), "LayoutType", StringValue ("RowFirst"));

mobility.SetMobilityModel ("ns3::RandomWalk2dMobilityModel", "Bounds", RectangleValue (Rectangle (-50, 50, -50, 50)));

mobility.Install (wifiStaNodes);

mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel"); mobility.Install (wifiApNode);

InternetStackHelper stack; stack.Install (csmaNodes); stack.Install (wifiApNode); stack.Install (wifiStaNodes); Ipv4AddressHelper address;

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

Ipv4InterfaceContainer p2pInterfaces; p2pInterfaces = address.Assign (p2pDevices); address.SetBase ("10.1.2.0", "255.255.255.0");

Ipv4InterfaceContainer csmaInterfaces; csmaInterfaces = address.Assign (csmaDevices); address.SetBase ("10.1.3.0", "255.255.255.0");

address.Assign (staDevices); address.Assign (apDevices); UdpEchoServerHelper echoServer (9);

ApplicationContainer serverApps = echoServer.Install (csmaNodes.Get (nCsma)); serverApps.Start (Seconds (1.0));

serverApps.Stop (Seconds (10.0));

UdpEchoClientHelper echoClient (csmaInterfaces.GetAddress (nCsma), 9); echoClient.SetAttribute ("MaxPackets", UintegerValue (1)); echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0))); echoClient.SetAttribute ("PacketSize", UintegerValue (1024)); ApplicationContainer clientApps =

echoClient.Install (wifiStaNodes.Get (nWifi - 1)); clientApps.Start (Seconds (2.0));

clientApps.Stop (Seconds (10.0)); Ipv4GlobalRoutingHelper::PopulateRoutingTables (); Simulator::Stop (Seconds (10.0));

if (tracing)

{

phy.SetPcapDataLinkType (WifiPhyHelper::DLT\_IEEE802\_11\_RADIO); pointToPoint.EnablePcapAll ("third");

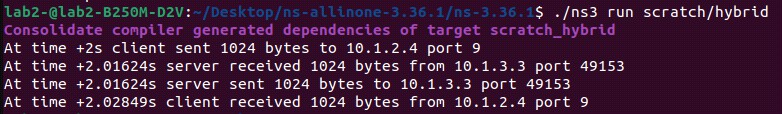
phy.EnablePcap ("third", apDevices.Get (0)); csma.EnablePcap ("third", csmaDevices.Get (0), true);

}

Simulator::Run (); Simulator::Destroy (); return 0;

}

**Output:**



# Practical No : 9

## Aim: Program to simulate UDP Server client.

**Code:**

**udp.cc**

#include <fstream>

#include "ns3/core-module.h" #include "ns3/core-module.h" #include "ns3/csma-module.h" #include "ns3/applications-module.h" #include "ns3/internet-module.h"

using namespace ns3;

NS\_LOG\_COMPONENT\_DEFINE ("UdpClientServerExample"); int

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

{

//

// Enable logging for UdpClient and

//

LogComponentEnable ("UdpClient", LOG\_LEVEL\_INFO); LogComponentEnable ("UdpServer", LOG\_LEVEL\_INFO);

bool useV6 = false; Address serverAddress;

CommandLine cmd;

cmd.AddValue ("useIpv6", "Use Ipv6", useV6); cmd.Parse (argc, argv);

//

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

//

NS\_LOG\_INFO ("Create nodes."); NodeContainer n;

n.Create (2);

InternetStackHelper internet; internet.Install (n);

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

//

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

//

CsmaHelper csma;

csma.SetChannelAttribute ("DataRate", DataRateValue (DataRate (5000000)));

csma.SetChannelAttribute ("Delay", TimeValue (MilliSeconds (2))); csma.SetDeviceAttribute ("Mtu", UintegerValue (1400)); NetDeviceContainer d = csma.Install (n);

//

// We've got the "hardware" in place. Now we need to add IP addresses.

//

NS\_LOG\_INFO ("Assign IP Addresses."); if (useV6 == false)

{

Ipv4AddressHelper ipv4;

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

Ipv4InterfaceContainer i = ipv4.Assign (d); serverAddress = Address (i.GetAddress (1));

}

else

{

Ipv6AddressHelper ipv6;

ipv6.SetBase ("2001:0000:f00d:cafe::", Ipv6Prefix (64)); Ipv6InterfaceContainer i6 = ipv6.Assign (d);

serverAddress = Address(i6.GetAddress (1,1));

}

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

//

// Create one udpServer applications on node one.

//

uint16\_t port = 4000; UdpServerHelper server (port);

ApplicationContainer apps = server.Install (n.Get (1)); apps.Start (Seconds (1.0));

apps.Stop (Seconds (10.0));

//

// Create one UdpClient application to send UDP datagrams from node zero to

// node one.

//

uint32\_t MaxPacketSize = 1024;

Time interPacketInterval = Seconds (0.05); uint32\_t maxPacketCount = 320;

UdpClientHelper client (serverAddress, port);

client.SetAttribute ("MaxPackets", UintegerValue (maxPacketCount)); client.SetAttribute ("Interval", TimeValue (interPacketInterval)); client.SetAttribute ("PacketSize", UintegerValue (MaxPacketSize));

apps = client.Install (n.Get (0)); apps.Start (Seconds (2.0));

apps.Stop (Seconds (10.0));

//

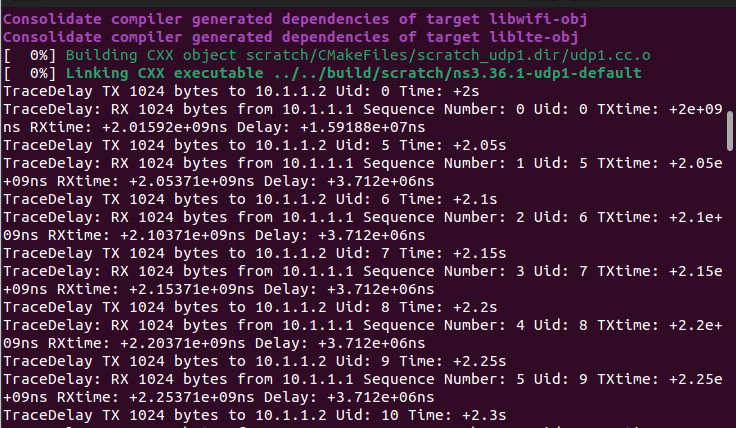
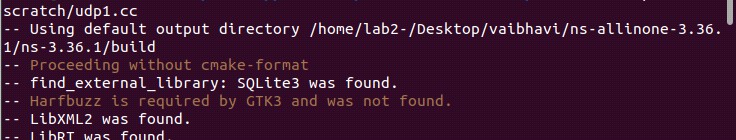
// Now, do the actual simulation.

//

NS\_LOG\_INFO ("Run Simulation."); Simulator::Run (); Simulator::Destroy (); NS\_LOG\_INFO ("Done.");

}

## Output:



**Practical No: 10**

**Aim: Program to simulate DHCP server and n clients.**

## Code:

#include "ns3/core-module.h" #include "ns3/internet-apps-module.h" #include "ns3/csma-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 ("DhcpExample"); int main (int argc, char \*argv[])

{

CommandLine cmd (\_FILE\_); bool verbose = true;

bool tracing = false;

cmd.AddValue ("verbose", "turn on the logs", verbose); cmd.AddValue ("tracing", "turn on the tracing", tracing); cmd.Parse (argc, argv);

// GlobalValue::Bind ("ChecksumEnabled", BooleanValue (true)); if (verbose)

{

LogComponentEnable ("DhcpServer", LOG\_LEVEL\_ALL); LogComponentEnable ("DhcpClient", LOG\_LEVEL\_ALL); LogComponentEnable ("UdpEchoServerApplication", LOG\_LEVEL\_INFO); LogComponentEnable ("UdpEchoClientApplication", LOG\_LEVEL\_INFO);

}

Time stopTime = Seconds (20); NS\_LOG\_INFO ("Create nodes.");

NodeContainer nodes; NodeContainer router; nodes.Create (3);

router.Create (2);

NodeContainer net (nodes, router); NS\_LOG\_INFO ("Create channels."); CsmaHelper csma;

csma.SetChannelAttribute ("DataRate", StringValue ("5Mbps")); csma.SetChannelAttribute ("Delay", StringValue ("2ms")); csma.SetDeviceAttribute ("Mtu", UintegerValue (1500)); NetDeviceContainer devNet = csma.Install (net);

NodeContainer p2pNodes; p2pNodes.Add (net.Get (4));

p2pNodes.Create (1); PointToPointHelper pointToPoint;

pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps")); pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms")); NetDeviceContainer p2pDevices;

p2pDevices = pointToPoint.Install (p2pNodes); InternetStackHelper tcpip;

tcpip.Install (nodes); tcpip.Install (router); tcpip.Install (p2pNodes.Get (1)); Ipv4AddressHelper address;

address.SetBase ("172.30.1.0", "255.255.255.0");

Ipv4InterfaceContainer p2pInterfaces; p2pInterfaces = address.Assign (p2pDevices);

// manually add a routing entry because we don't want to add a dynamic routing Ipv4StaticRoutingHelper ipv4RoutingHelper;

Ptr<Ipv4> ipv4Ptr = p2pNodes.Get (1)->GetObject<Ipv4> ();

Ptr<Ipv4StaticRouting> staticRoutingA = ipv4RoutingHelper.GetStaticRouting (ipv4Ptr); staticRoutingA->AddNetworkRouteTo (Ipv4Address ("172.30.0.0"), Ipv4Mask ("/24"), Ipv4Address ("172.30.1.1"), 1);

NS\_LOG\_INFO ("Setup the IP addresses and create DHCP applications."); DhcpHelper dhcpHelper;

// The router must have a fixed IP.

Ipv4InterfaceContainer fixedNodes = dhcpHelper.InstallFixedAddress (devNet.Get (4), Ipv4Address ("172.30.0.17"), Ipv4Mask ("/24"));

// Not really necessary, IP forwarding is enabled by default in IPv4. fixedNodes.Get (0).first->SetAttribute ("IpForward", BooleanValue (true));

// DHCP server

ApplicationContainer dhcpServerApp = dhcpHelper.InstallDhcpServer (devNet.Get (3), Ipv4Address ("172.30.0.12"),

Ipv4Address ("172.30.0.0"), Ipv4Mask ("/24"), Ipv4Address ("172.30.0.10"), Ipv4Address ("172.30.0.15"),

Ipv4Address ("172.30.0.17"));

// This is just to show how it can be done.

DynamicCast<DhcpServer> (dhcpServerApp.Get (0))->AddStaticDhcpEntry (devNet.Get (2)->GetAddress (), Ipv4Address ("172.30.0.14"));

dhcpServerApp.Start (Seconds (0.0)); dhcpServerApp.Stop (stopTime);

// DHCP clients

NetDeviceContainer dhcpClientNetDevs; dhcpClientNetDevs.Add (devNet.Get (0));

dhcpClientNetDevs.Add (devNet.Get (1));

dhcpClientNetDevs.Add (devNet.Get (2));

ApplicationContainer dhcpClients = dhcpHelper.InstallDhcpClient (dhcpClientNetDevs); dhcpClients.Start (Seconds (1.0));

dhcpClients.Stop (stopTime); UdpEchoServerHelper echoServer (9);

ApplicationContainer serverApps = echoServer.Install (p2pNodes.Get (1)); serverApps.Start (Seconds (0.0));

serverApps.Stop (stopTime);

UdpEchoClientHelper echoClient (p2pInterfaces.GetAddress (1), 9); echoClient.SetAttribute ("MaxPackets", UintegerValue (100)); echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0))); echoClient.SetAttribute ("PacketSize", UintegerValue (1024)); ApplicationContainer clientApps = echoClient.Install (nodes.Get (1)); clientApps.Start (Seconds (10.0));

clientApps.Stop (stopTime);

Simulator::Stop (stopTime + Seconds (10.0)); if (tracing)

{

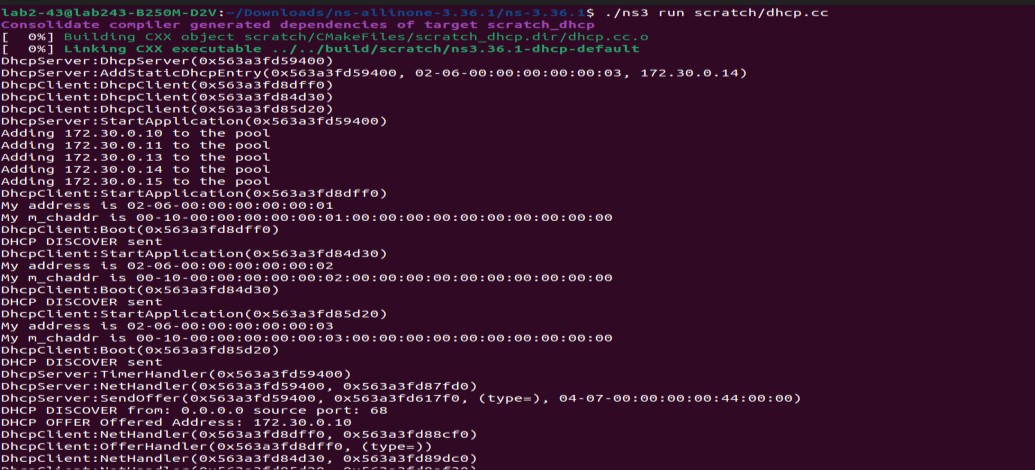
csma.EnablePcapAll ("dhcp-csma"); pointToPoint.EnablePcapAll ("dhcp-p2p");

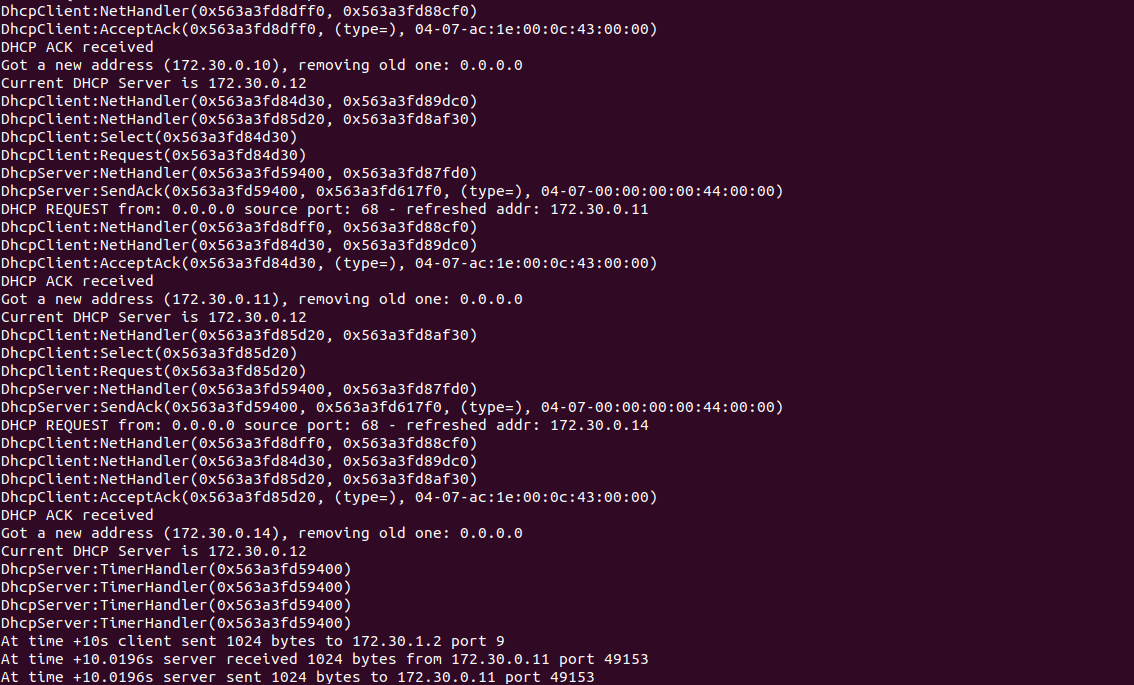
}

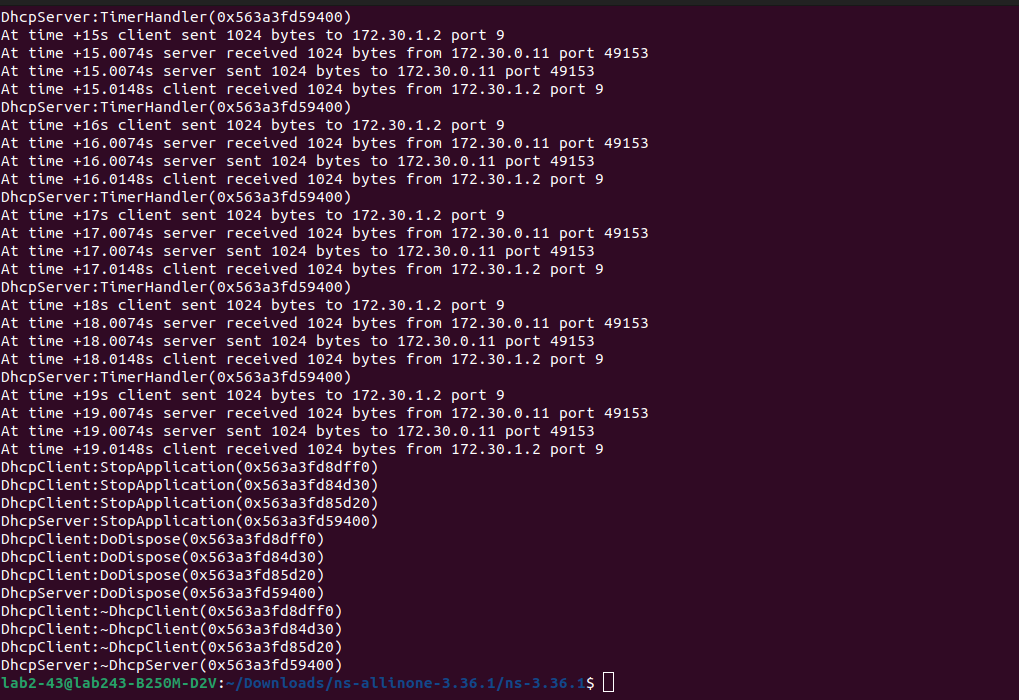
NS\_LOG\_INFO ("Run Simulation."); Simulator::Run (); Simulator::Destroy (); NS\_LOG\_INFO ("Done.");

}

**Output:**







# Practical No : 11

## Aim: Program to simulate FTP using TCP protocol Code:

[**ftp.cc**](file://localhost/C:/Users/ADMIN/Downloads/ftp.cc)

#include <string> #include <fstream>

#include "ns3/core-module.h"

#include "ns3/point-to-point-module.h" #include "ns3/internet-module.h" #include "ns3/applications-module.h" #include "ns3/network-module.h" #include "ns3/packet-sink.h"

using namespace ns3;

NS\_LOG\_COMPONENT\_DEFINE ("TcpBulkSendExample"); int

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

{

bool tracing = false; uint32\_t maxBytes = 0;

//

// Allow the user to override any of the defaults at

// run-time, via command-line arguments

//

CommandLine cmd;

cmd.AddValue ("tracing", "Flag to enable/disable tracing", tracing); cmd.AddValue ("maxBytes",

"Total number of bytes for application to send", maxBytes); cmd.Parse (argc, argv);

//

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

//

NS\_LOG\_INFO ("Create nodes."); NodeContainer nodes; nodes.Create (2);

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

//

// Explicitly create the point-to-point link required by the topology (shown above).

//

PointToPointHelper pointToPoint;

pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("500Kbps")); pointToPoint.SetChannelAttribute ("Delay", StringValue ("5ms"));

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

//

// Install the internet stack on the nodes

//

InternetStackHelper internet; internet.Install (nodes);

//

// We've got the "hardware" in place. Now we need to add IP addresses.

//

NS\_LOG\_INFO ("Assign IP Addresses."); Ipv4AddressHelper ipv4;

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

Ipv4InterfaceContainer i = ipv4.Assign (devices); NS\_LOG\_INFO ("Create Applications.");

//

// Create a BulkSendApplication and install it on node 0

//

uint16\_t port = 9; // well-known echo port number

BulkSendHelper source ("ns3::TcpSocketFactory", InetSocketAddress (i.GetAddress (1), port));

// Set the amount of data to send in bytes. Zero is unlimited. source.SetAttribute ("MaxBytes", UintegerValue (maxBytes)); ApplicationContainer sourceApps = source.Install (nodes.Get (0)); sourceApps.Start (Seconds (0.0));

sourceApps.Stop (Seconds (10.0));

//

// Create a PacketSinkApplication and install it on node 1

//

PacketSinkHelper sink ("ns3::TcpSocketFactory", InetSocketAddress (Ipv4Address::GetAny (), port));

ApplicationContainer sinkApps = sink.Install (nodes.Get (1)); sinkApps.Start (Seconds (0.0));

sinkApps.Stop (Seconds (10.0));

//

// Set up tracing if enabled

//

if (tracing)

{

AsciiTraceHelper ascii;

pointToPoint.EnableAsciiAll (ascii.CreateFileStream ("tcp-bulk-send.tr")); pointToPoint.EnablePcapAll ("tcp-bulk-send", false);

}

//

// Now, do the actual simulation.

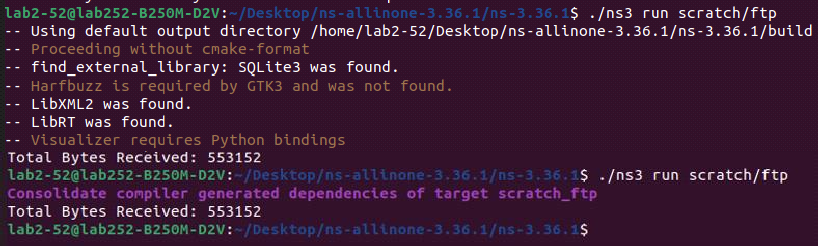
//

NS\_LOG\_INFO ("Run Simulation."); Simulator::Stop (Seconds (10.0)); Simulator::Run (); Simulator::Destroy (); NS\_LOG\_INFO ("Done.");

Ptr<PacketSink> sink1 = DynamicCast<PacketSink> (sinkApps.Get (0)); std::cout << "Total Bytes Received: " << sink1->GetTotalRx () << std::endl;

}

**Output:**



## Practical No: 12

**Aim: Animate a simple network using Net Anim in Network Simulator.**

## Code:

#include "ns3/coremodule.h" #include "ns3/network-module.h" #include "ns3/csmamodule.h" #include "ns3/internet-module.h"

#include "ns3/point-to-pointmodule.h" #include "ns3/applications-module.h" #include "ns3/ipv4-globalrouting-helper.h"

//Inculding Header File

#include "ns3/netanim-module.h"

// Default Network Topology

//

// 10.1.1.0 // n0 n1 n2 n3 n4

// point-to-point | | | |

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

// LAN 10.1.2.0

using namespace ns3;

NS\_LOG\_COMPONENT\_DEFINE ("SecondScriptExample"); int main (int argc, char \*argv[])

{ bool verbose = true; uint32\_t nCsma = 3; CommandLine cmd ( FILE );

cmd.AddValue ("nCsma", "Number of \"extra\" CSMA nodes/devices", nCsma);cmd.AddValue ("verbose", "Tell echo applications to log if true", verbose);

cmd.Parse (argc,argv); if (verbose)

{

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

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

}

nCsma = nCsma == 0 ? 1 : nCsma; NodeContainer p2pNodes; p2pNodes.Create (2); NodeContainer csmaNodes; csmaNodes.Add (p2pNodes.Get (1)); csmaNodes.Create (nCsma); PointToPointHelper pointToPoint;

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

NetDeviceContainer p2pDevices;

p2pDevices = pointToPoint.Install (p2pNodes); CsmaHelper csma;

csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps")); csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));

NetDeviceContainer csmaDevices; csmaDevices = csma.Install (csmaNodes); InternetStackHelper stack;

stack.Install (p2pNodes.Get (0));

stack.Install (csmaNodes); Ipv4AddressHelper address;

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

Ipv4InterfaceContainer p2pInterfaces; p2pInterfaces = address.Assign (p2pDevices);

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

Ipv4InterfaceContainer csmaInterfaces; csmaInterfaces = address.Assign (csmaDevices); UdpEchoServerHelper echoServer (9);

ApplicationContainer serverApps = echoServer.Install (csmaNodes.Get (nCsma));serverApps.Start (Seconds (1.0));

serverApps.Stop (Seconds (10.0));

UdpEchoClientHelper echoClient (csmaInterfaces.GetAddress (nCsma), 9); echoClient.SetAttribute ("MaxPackets", UintegerValue (1)); echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0))); echoClient.SetAttribute ("PacketSize", UintegerValue (1024)); ApplicationContainer clientApps = echoClient.Install (p2pNodes.Get (0)); clientApps.Start (Seconds (2.0));

clientApps.Stop (Seconds (10.0));

Ipv4GlobalRouting Helper::PopulateRo utingTables (); pointToPoint.Enabl ePcapAll ("pract9"); csma.EnablePcap ("pact9", csmaDevices.Get (1), true);

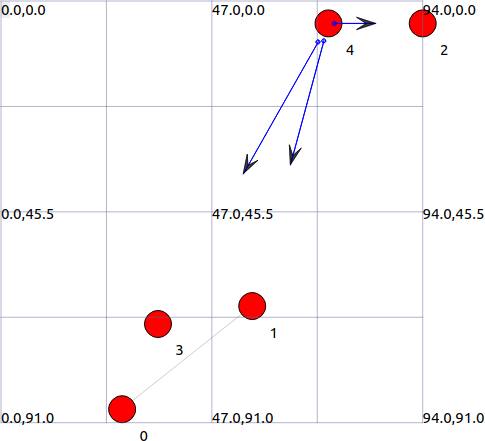
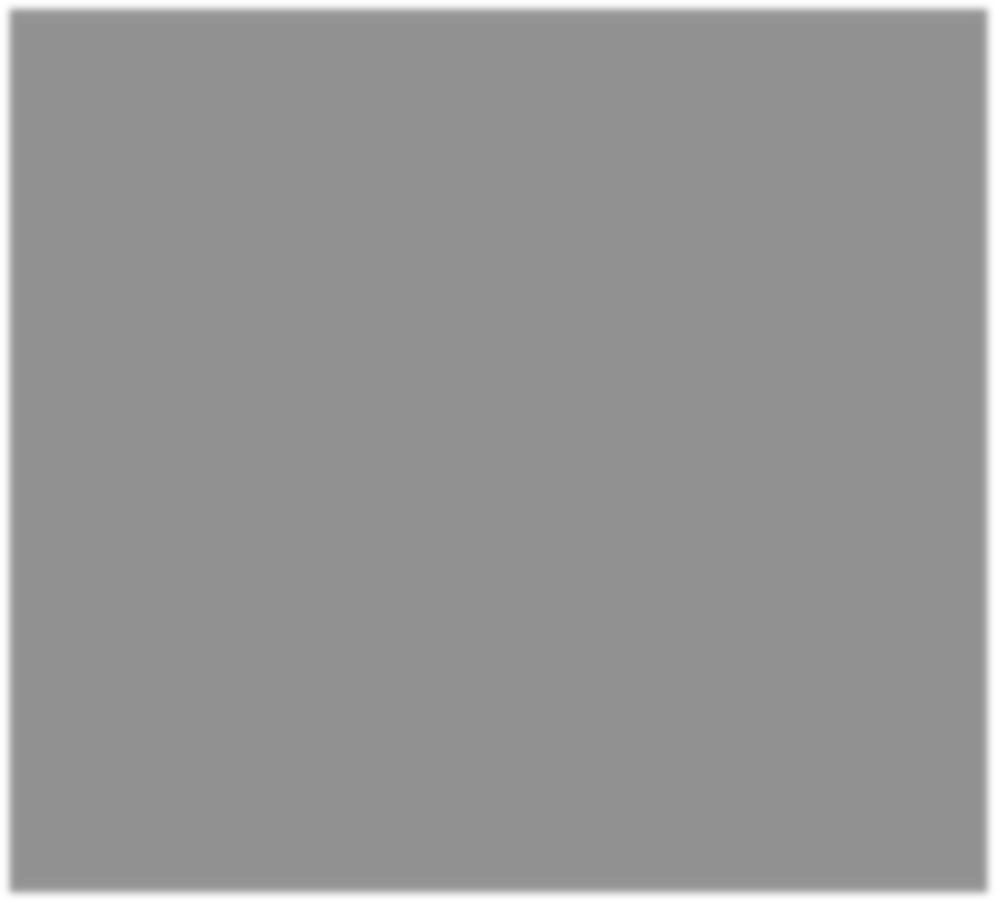
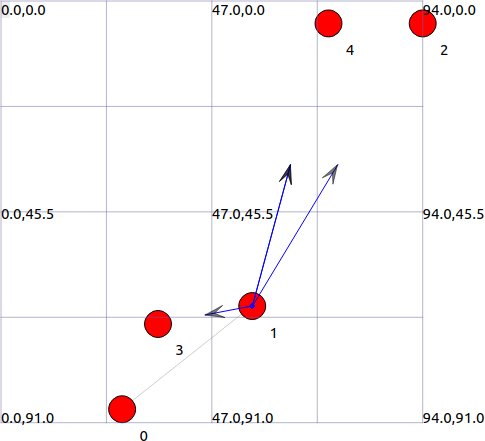
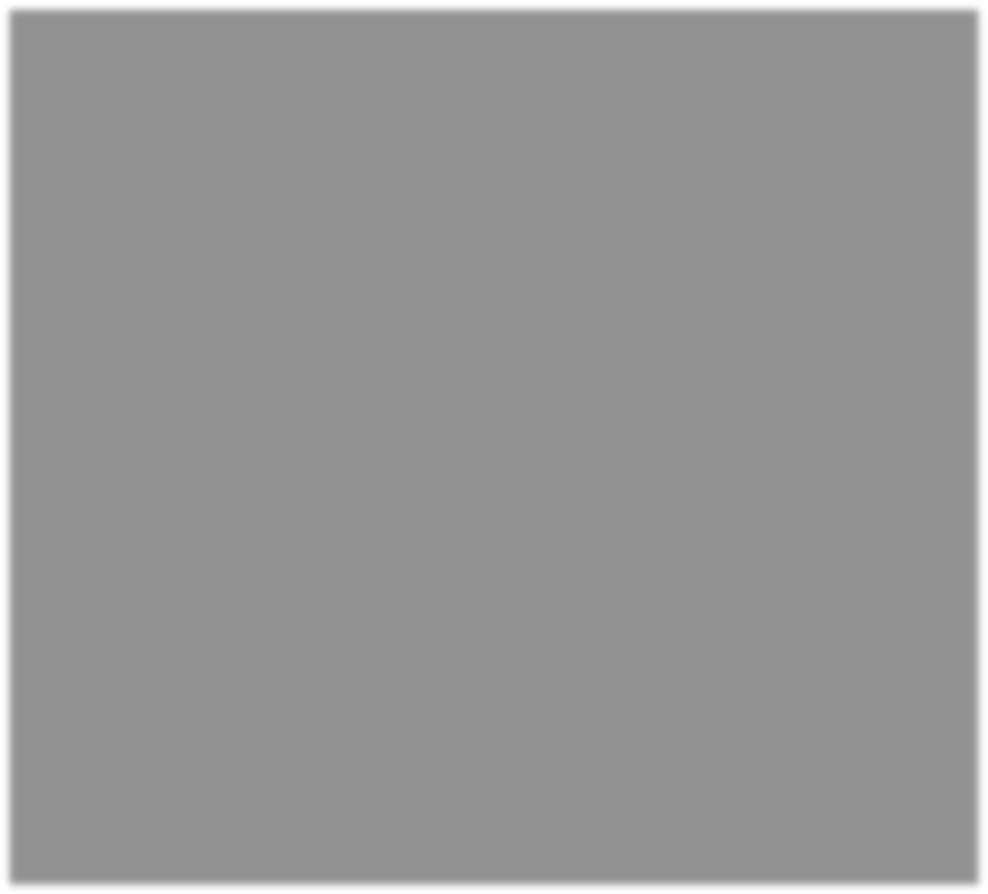
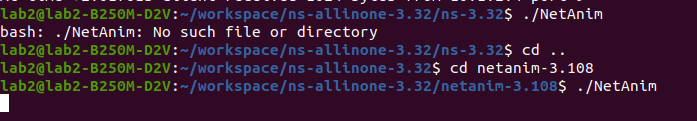
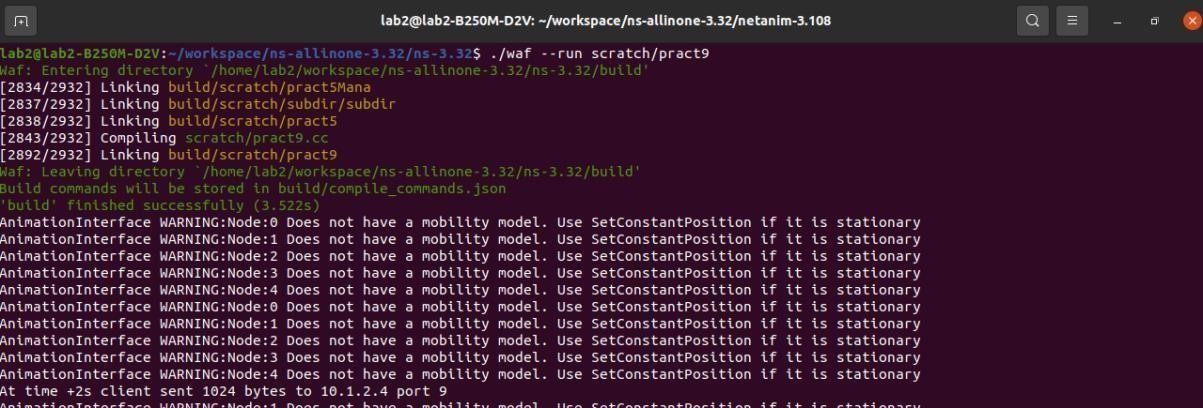
//Including Animation AnimationInterface anim("pract9.xml");

Simulator::Run (); Simulator::Destroy (); return 0;

}

**Output:**

Running the file and visualizing



**Practical No: 13**

## Aim: Animate three-way handshake for TCP Connection using NetAnim.

**Code:**

**pract9.cc**

#include "ns3/core-module.h" #include "ns3/network-module.h" #include "ns3/csma-module.h" #include "ns3/internet-module.h"

#include "ns3/point-to-point-module.h" #include "ns3/applications-module.h" #include "ns3/ipv4-global-routing-helper.h"

//Inculding Header File

#include "ns3/netanim-module.h"

// Default Network Topology

//

// 10.1.1.0

// n0 n1 n2 n3 n4

NETWORKING WITH LINUX

// point-to-point | | | |

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

// LAN 10.1.2.0

using namespace ns3;

NS\_LOG\_COMPONENT\_DEFINE ("SecondScriptExample"); int

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

{

bool verbose = true; uint32\_t nCsma = 3;

CommandLine cmd ( FILE );

cmd.AddValue ("nCsma", "Number of \"extra\" CSMA nodes/devices", nCsma); cmd.AddValue ("verbose", "Tell echo applications to log if true", verbose); cmd.Parse (argc,argv);

if (verbose)

{

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

}

nCsma = nCsma == 0 ? 1 : nCsma; NodeContainer p2pNodes; p2pNodes.Create (2); NodeContainer csmaNodes; csmaNodes.Add (p2pNodes.Get (1)); csmaNodes.Create (nCsma);

PointToPointHelper pointToPoint;

pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps")); pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms")); NetDeviceContainer p2pDevices;

p2pDevices = pointToPoint.Install (p2pNodes); CsmaHelper csma;

csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps")); csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560))); NetDeviceContainer csmaDevices;

csmaDevices = csma.Install (csmaNodes); InternetStackHelper stack;

stack.Install (p2pNodes.Get (0)); NETWORKING WITH LINUX

stack.Install (csmaNodes); Ipv4AddressHelper address;

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

Ipv4InterfaceContainer p2pInterfaces; p2pInterfaces = address.Assign (p2pDevices); address.SetBase ("10.1.2.0", "255.255.255.0");

Ipv4InterfaceContainer csmaInterfaces; csmaInterfaces = address.Assign (csmaDevices); UdpEchoServerHelper echoServer (9);

ApplicationContainer serverApps = echoServer.Install (csmaNodes.Get (nCsma)); serverApps.Start (Seconds (1.0));

serverApps.Stop (Seconds (10.0));

UdpEchoClientHelper echoClient (csmaInterfaces.GetAddress (nCsma), 9); echoClient.SetAttribute ("MaxPackets", UintegerValue (1)); echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0))); echoClient.SetAttribute ("PacketSize", UintegerValue (1024)); ApplicationContainer clientApps = echoClient.Install (p2pNodes.Get (0)); clientApps.Start (Seconds (2.0));

clientApps.Stop (Seconds (10.0)); Ipv4GlobalRoutingHelper::PopulateRoutingTables (); pointToPoint.EnablePcapAll ("pract9"); csma.EnablePcap ("pact9", csmaDevices.Get (1), true);

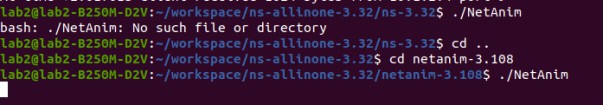
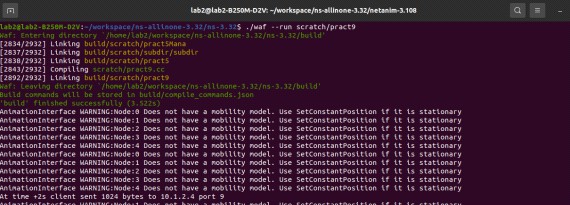
//Including Animation AnimationInterface anim("pract9.xml"); Simulator::Run ();

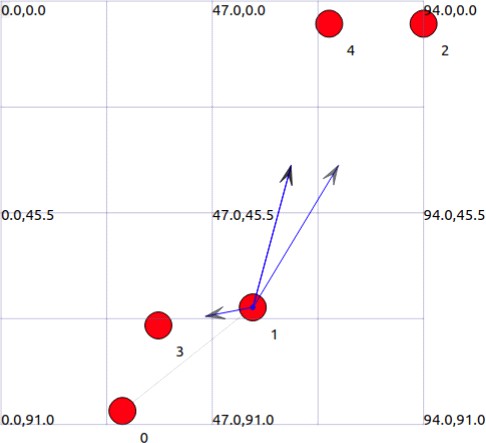
Simulator::Destroy (); return 0;

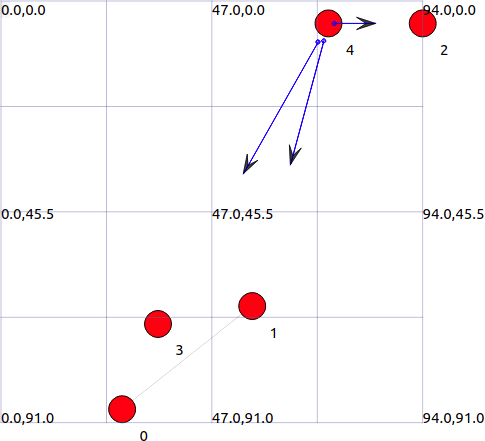
}

**Output**:

Running the file and visualizing it.







# Practical No: 14

## Aim: Program to assign IPv4 Addresses in NS3.

**Code:**

#include "ns3/core-module.h" #include "ns3/network-module.h" #include "ns3/internet-module.h"

#include "ns3/point-to-point-module.h" using namespace ns3;

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

// Create a node container NodeContainer nodes; nodes.Create(3);

// Create two point-to-point helpers PointToPointHelper pointToPoint1, pointToPoint2;

pointToPoint1.SetDeviceAttribute("DataRate", StringValue("5Mbps")); pointToPoint1.SetChannelAttribute("Delay", StringValue("2ms")); pointToPoint2.SetDeviceAttribute("DataRate", StringValue("5Mbps")); pointToPoint2.SetChannelAttribute("Delay", StringValue("2ms"));

// Create two net device containers and install devices NetDeviceContainer devices1, devices2;

devices1 = pointToPoint1.Install(nodes.Get(0), nodes.Get(1)); devices2 = pointToPoint2.Install(nodes.Get(1), nodes.Get(2));

// Install Internet stack on nodes InternetStackHelper internet; internet.Install(nodes);

// Assign IP addresses Ipv4AddressHelper ipv4; ipv4.SetBase("10.1.1.0", "255.255.255.0");

Ipv4InterfaceContainer interfaces1 = ipv4.Assign(devices1); Ipv4InterfaceContainer interfaces2 = ipv4.Assign(devices2);

// Print assigned IP addresses

for (uint32\_t i = 0; i < interfaces1.GetN(); ++i) {

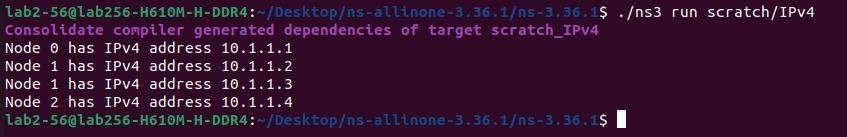
std::cout << "Node "<< i << " has IPv4 address " << interfaces1.GetAddress(i) << std::endl;

}

for (uint32\_t i = 0; i < interfaces2.GetN(); ++i) {

std::cout << "Node "<< i+1 << " has IPv4 address " << interfaces2.GetAddress(i) <<

std::endl;



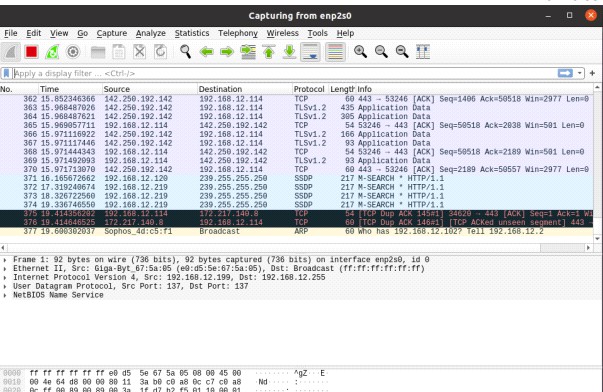
}

Simulator::Run(); Simulator::Destroy();

return 0;

}

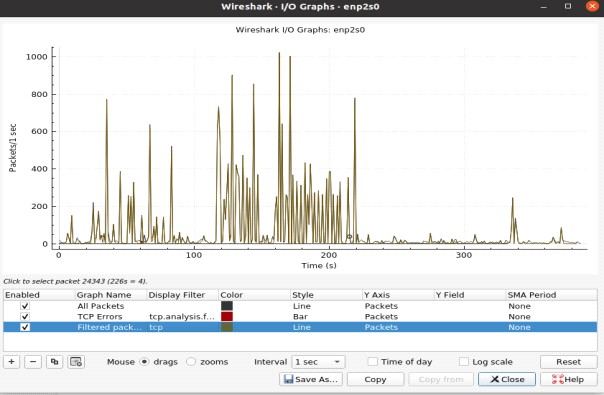
## Output:

**Practical No :15 Aim: Analyze the Network traffic using Wireshark Steps: Considering en2s0 network**

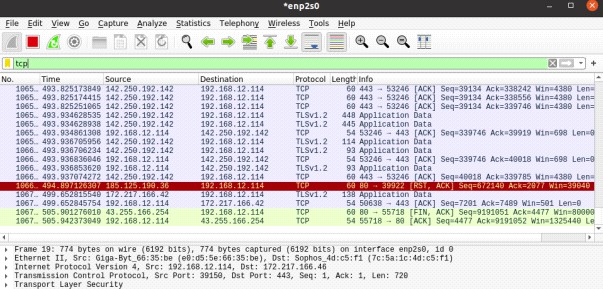
1. ARP



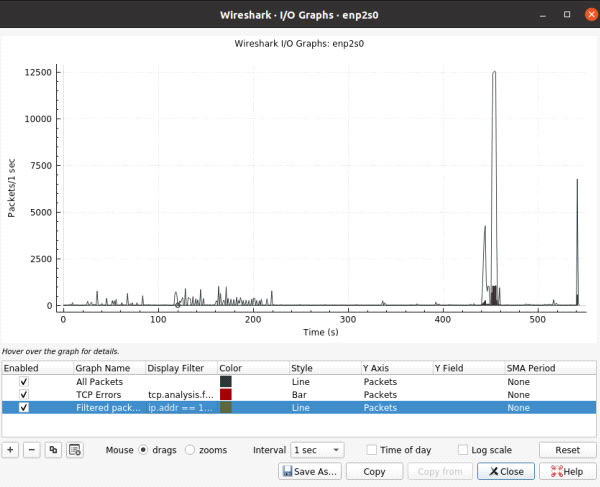
1. Graph



1. TCP



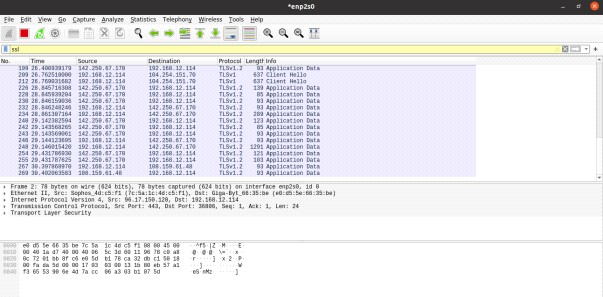


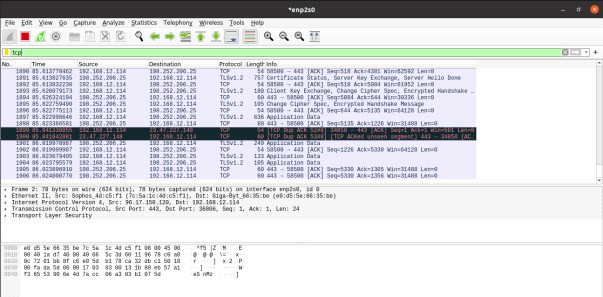


# Practical No: 16

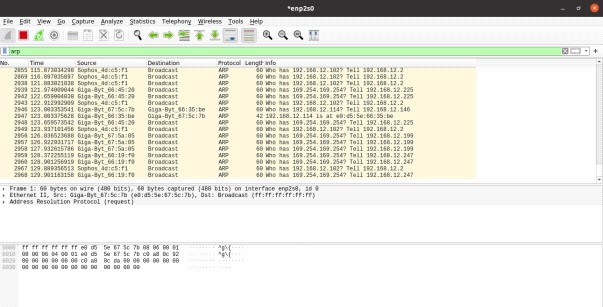
## Aim: Analyze the performance parameter of the network using Wireshark.

**Steps:**

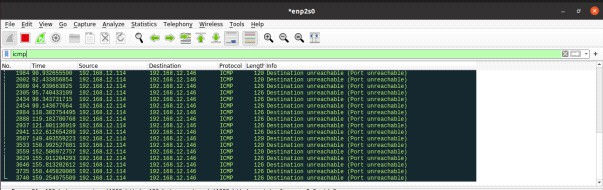
1. SSL
2. TCP

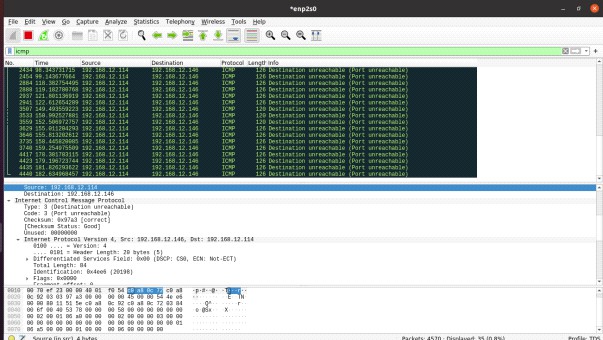


1. ARP



1. ICMP





1. Tcp.port == 80

