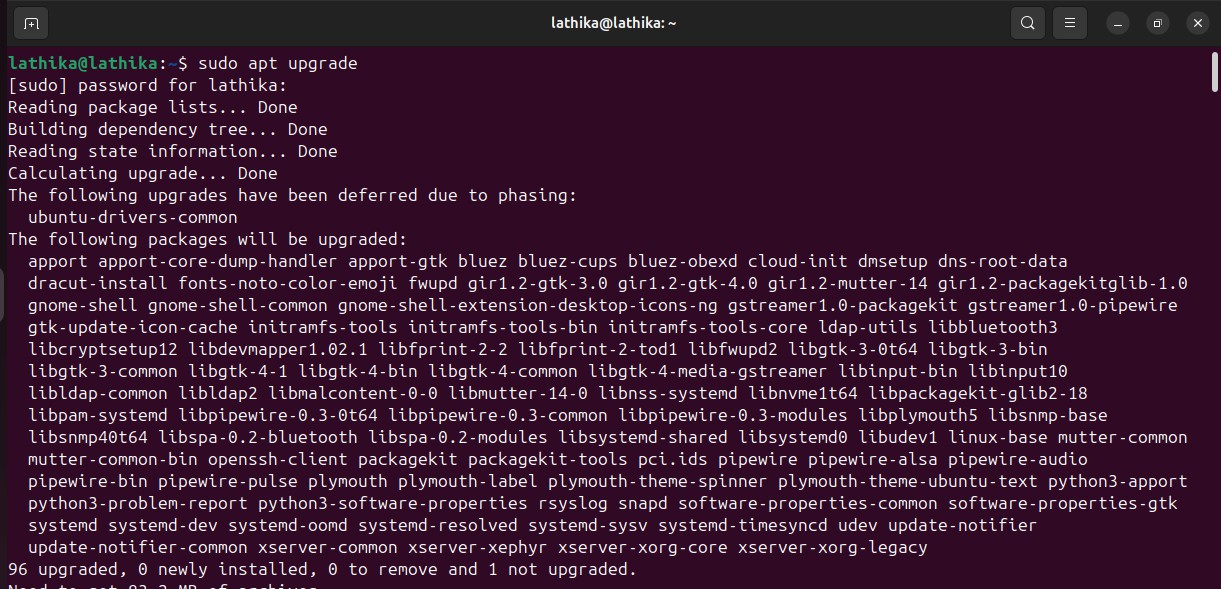
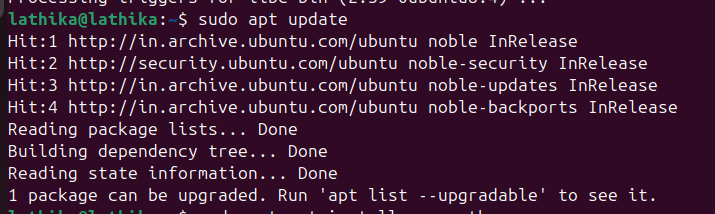
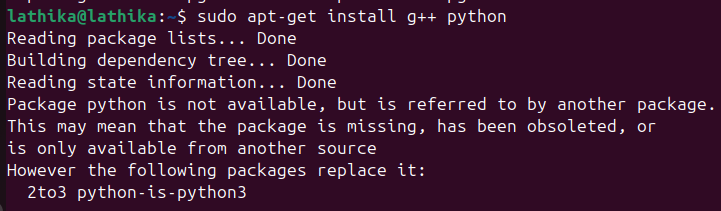
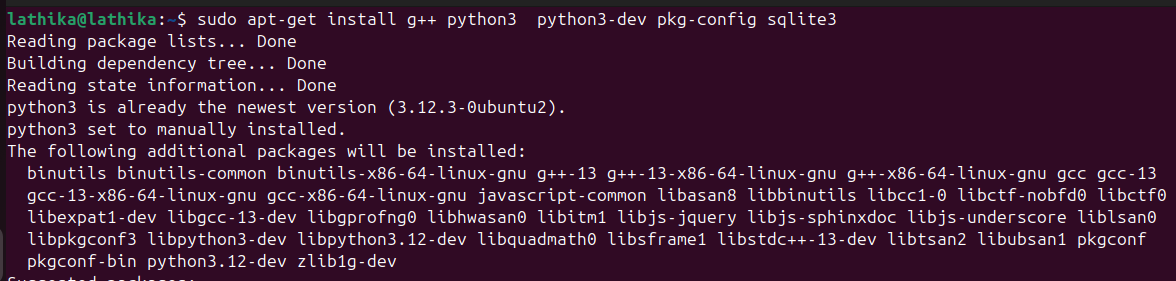
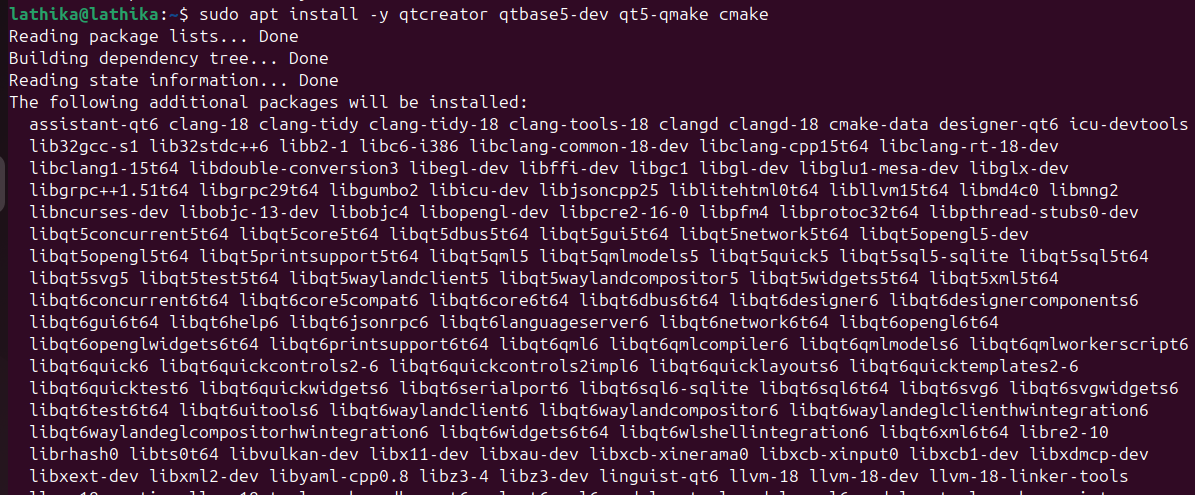
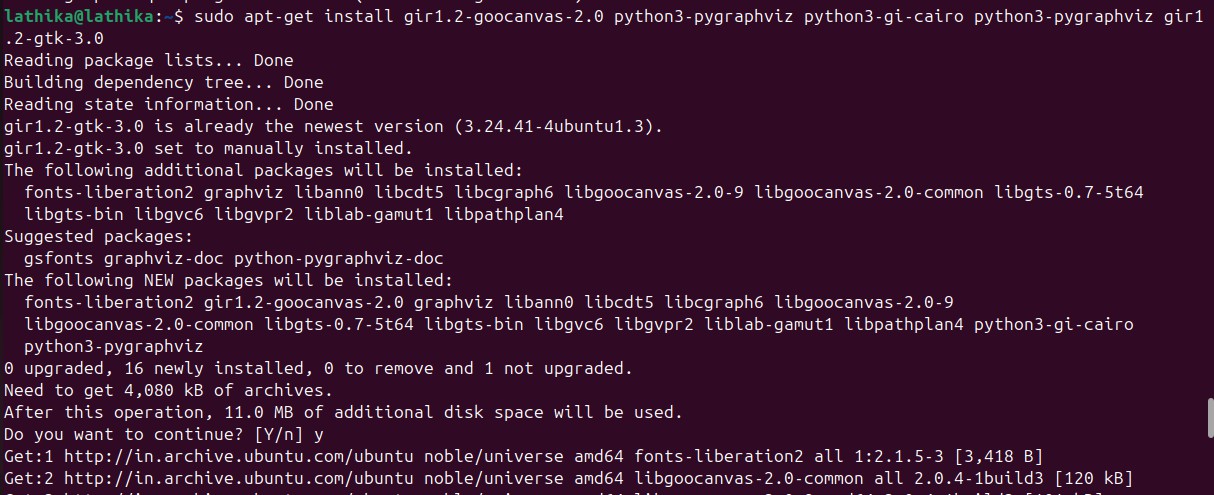
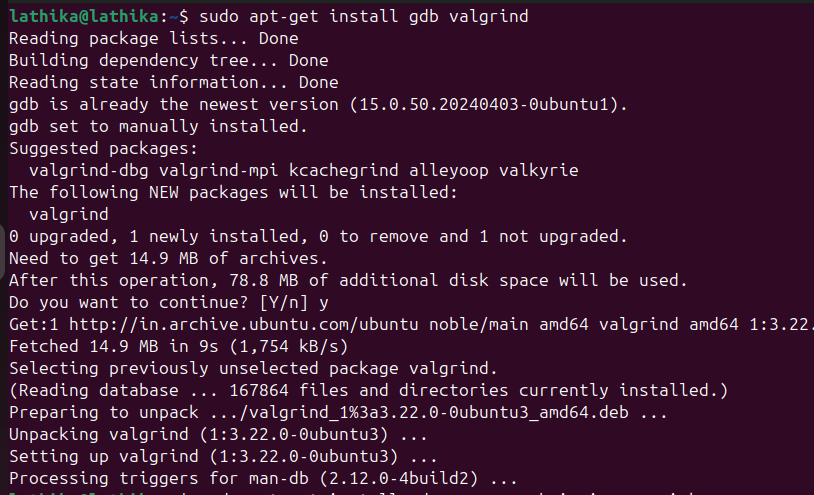
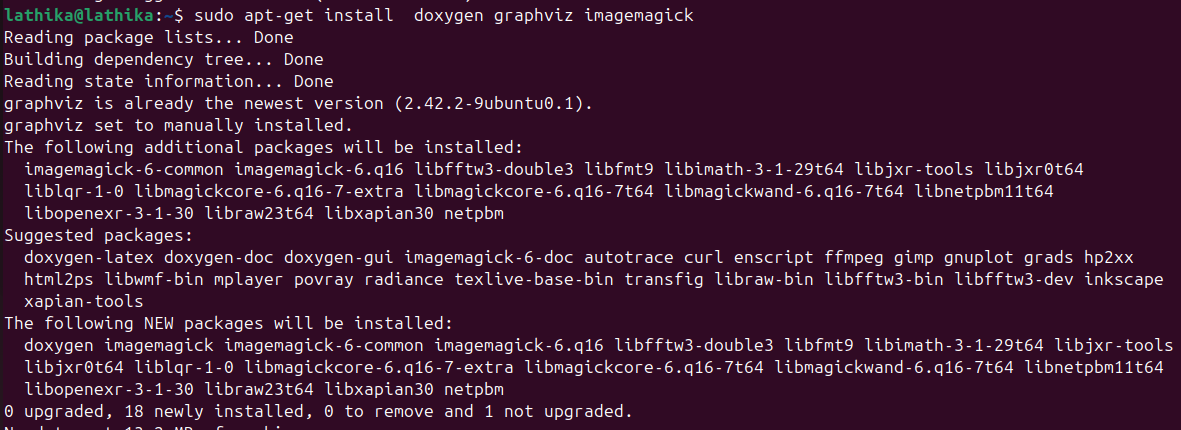
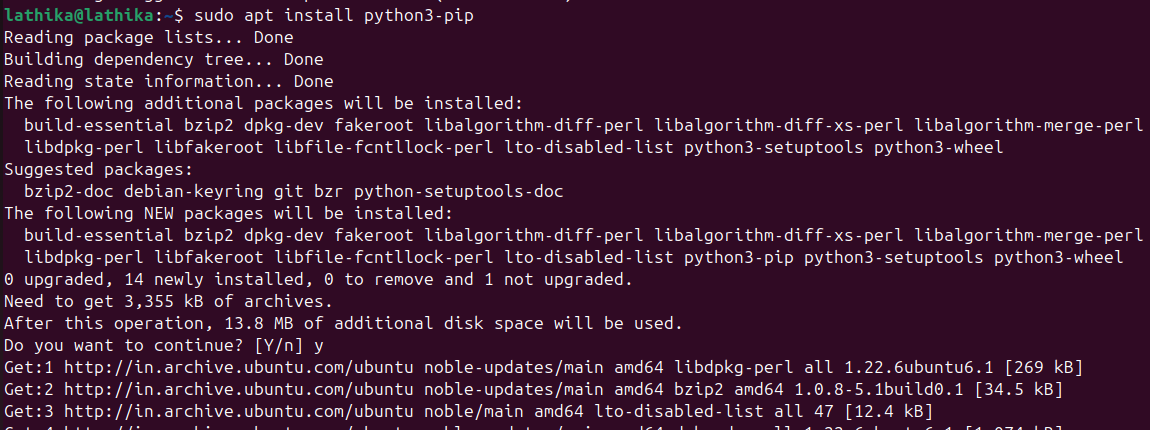
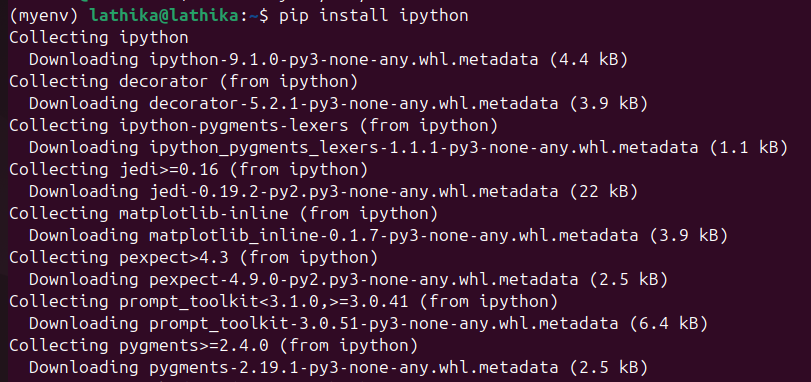
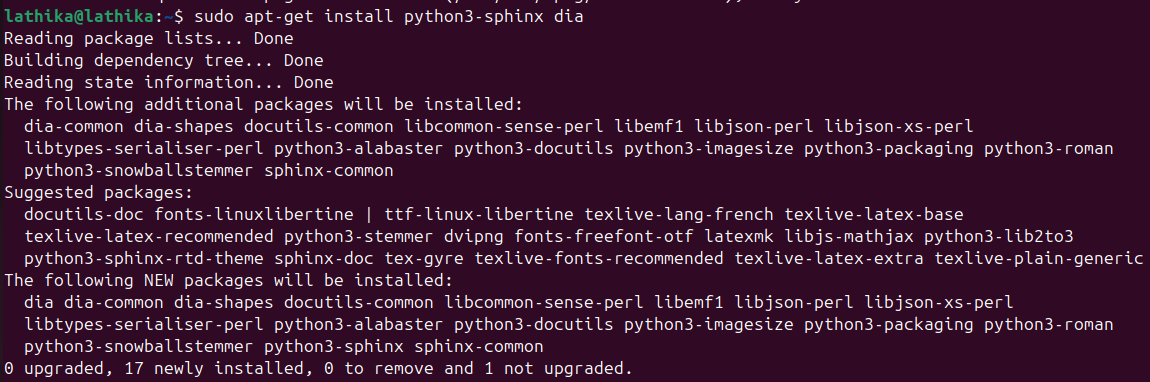
# Practical No - 1

## Aim: Installation: ns3 in Linux, NetAnim, Wireshark, PyViz, tcpdump

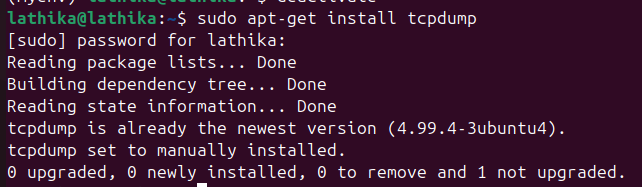
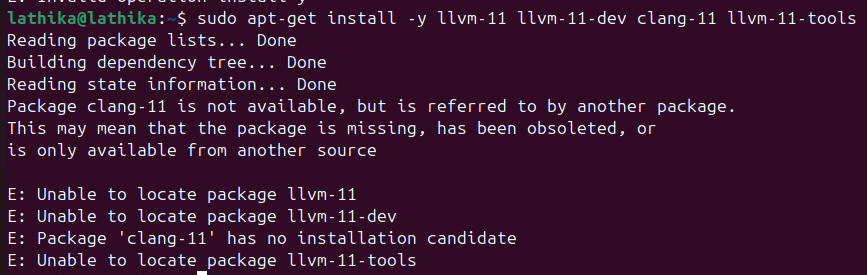
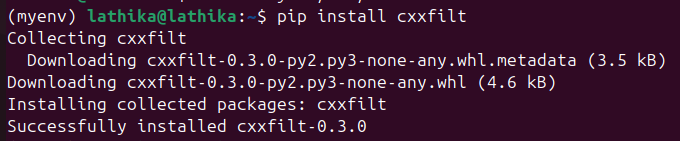
**Steps:**

* **sudo apt upgrade**
* **sudo apt update**

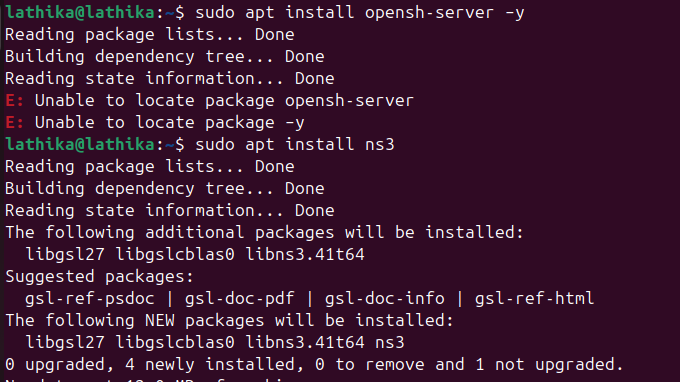
**PyViz**

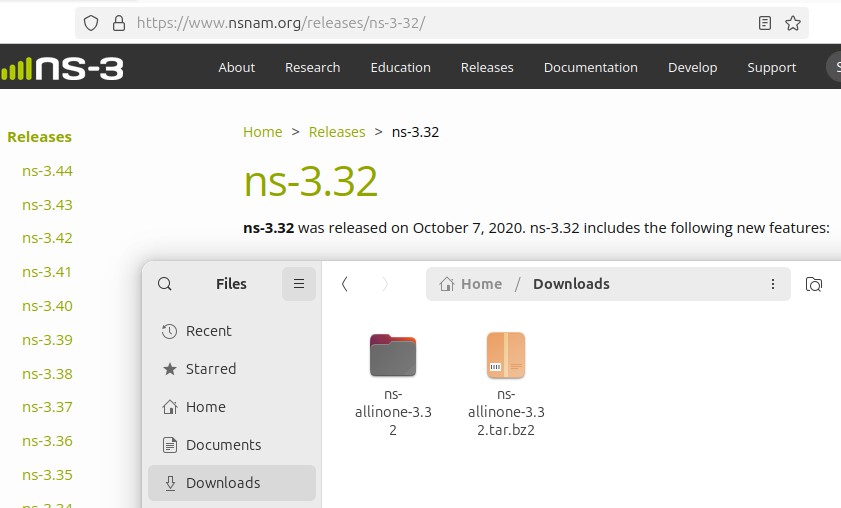
* **sudo apt-get install g++ python**
* **sudo apt-get install g++ python3 python3-dev pkg-config sqlite3**
* **sudo apt install -y qtcreator qtbase5-dev qt5-qmake cmake**
* **sudo apt-get install gir1.2-goocanvas-2.0 python3-pygraphviz python3-gi-cairo python3-pygraphviz gir1.2-gtk-3.0**
* **sudo apt-get install gdbvalgrind**
* **sudo apt-get install doxygengraphvizimagemagick**
* **sudo apt install python3-pip**
* **pip install ipython**
* **sudo apt-get install python3-sphinx dia**

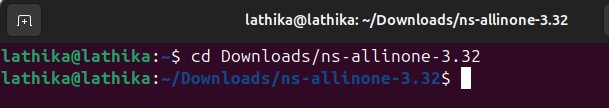
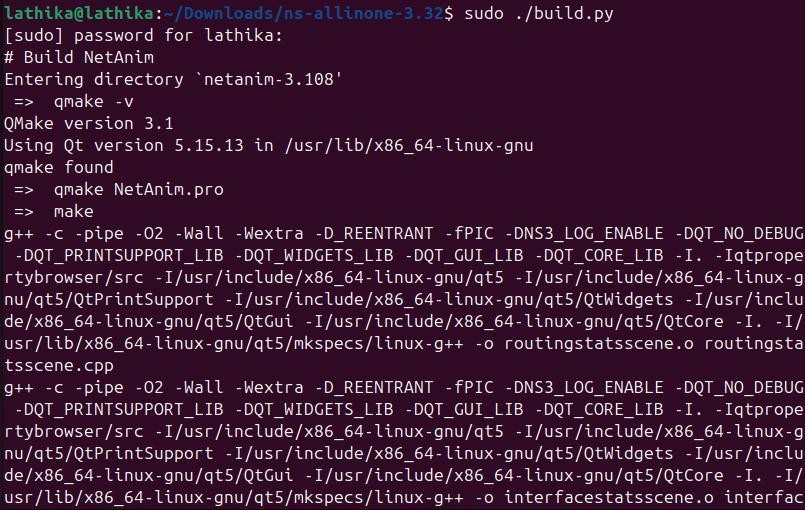
**tcpdump**

* **sudo apt-get install tcpdump**
* **sudo apt-get install-y llvm-11 llvm-11-dev clang-11 llvm-11-tools**
* **pip install cxxfilt**

**ns3 in Linux**

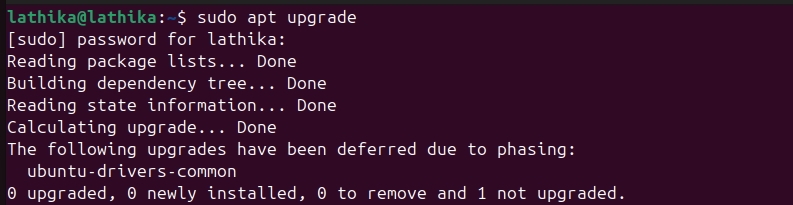
* **sudo apt install ns3**
* **Download NS3 from http**[**s://www.nsnam.org**](http://www.nsnam.org/) **and extract the folder.**

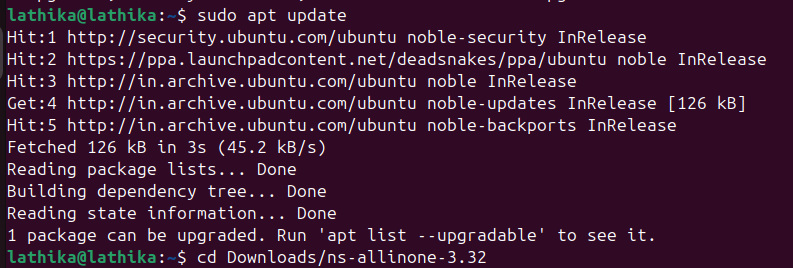


* **cd Downloads/ ns-allinone-3.32**
* **sudo ./build.py**
* **cd ns-3.36.1**
* **./text.py**

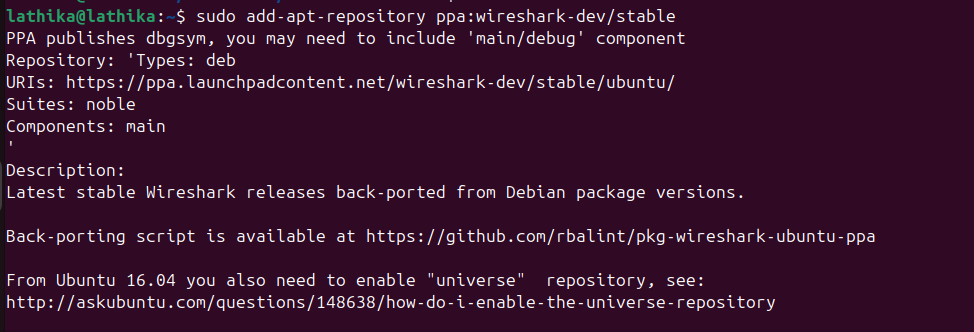


**Installation of Wireshark**

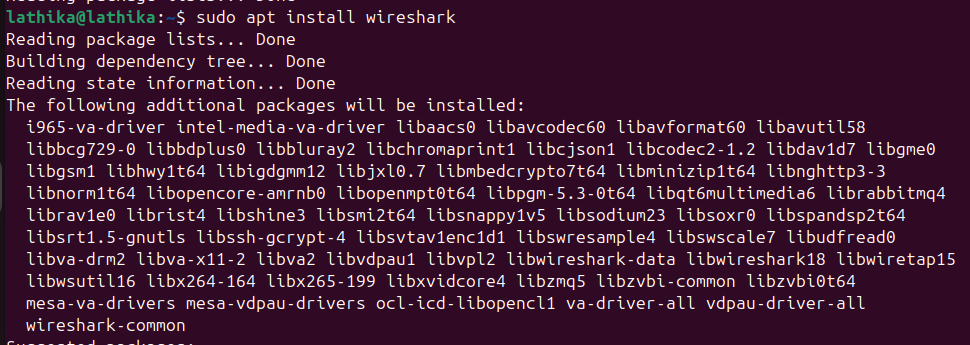
1. **sudo apt upgrade**
2. **Sudo apt update**

****

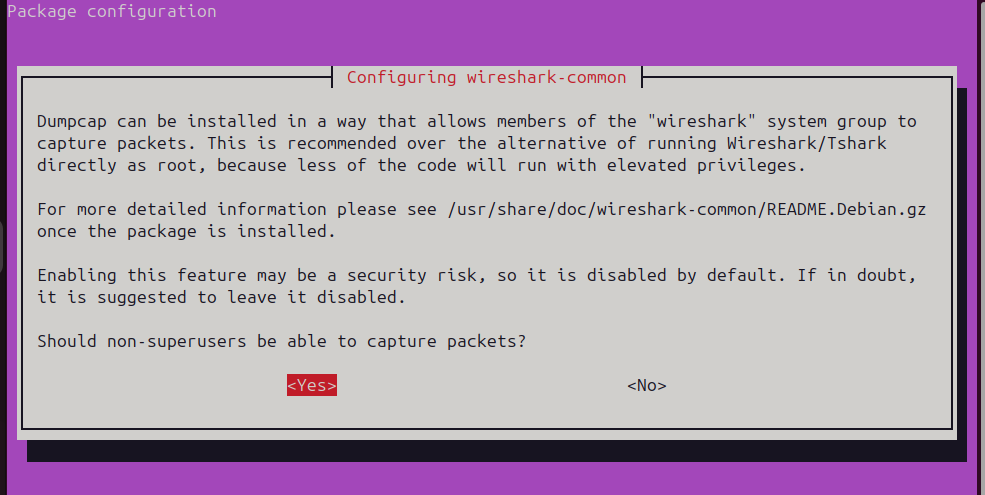
1. **Sudo add-apt-repository ppa:wireshark-dev/stable**

****

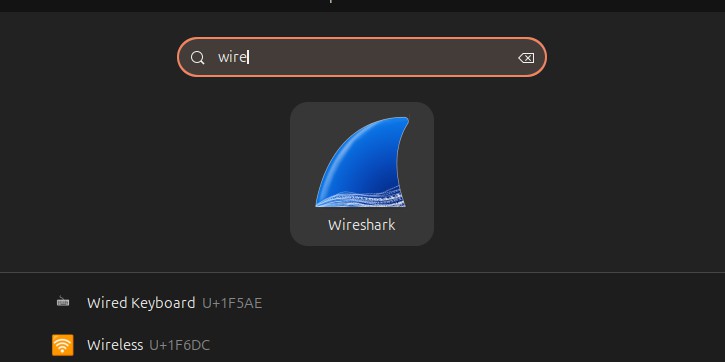
1. **sudoaptinstallwireshark**

****

1. **ConfiguringWireshark>Clickyes**

****

**Installs Wireshark**

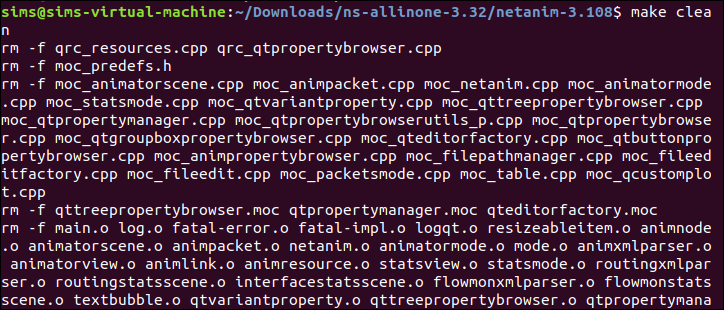
****

**NetAnim Installation**

1. **cd**

****

1. **make clean**

****

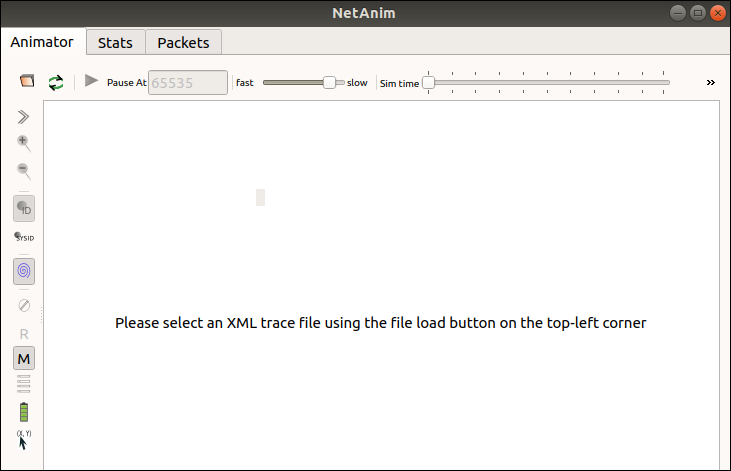
1. **qmake NetAnim.pro**

****

1. **qmake NetAnim.pro**

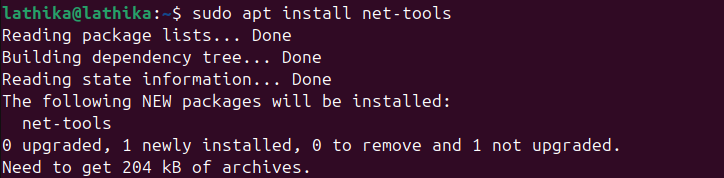
**./NetAnim**

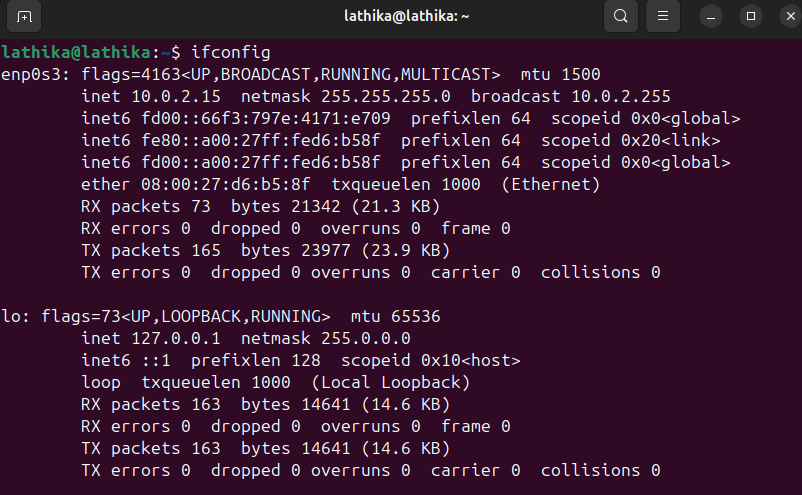
**OUTPUT :**

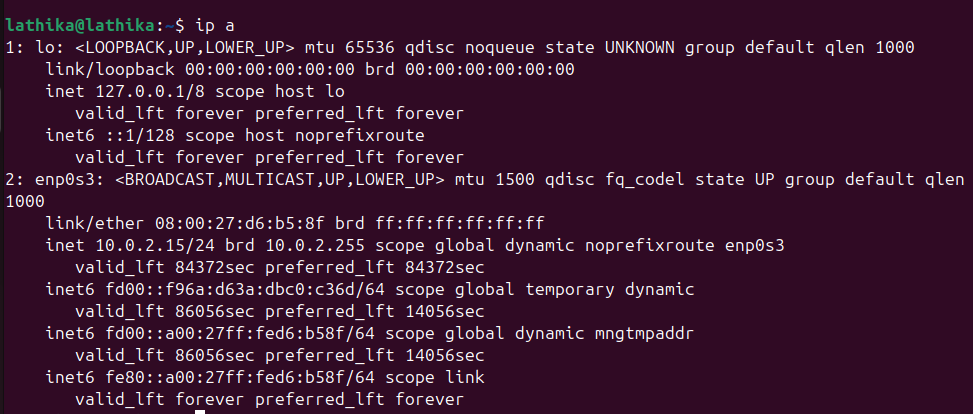


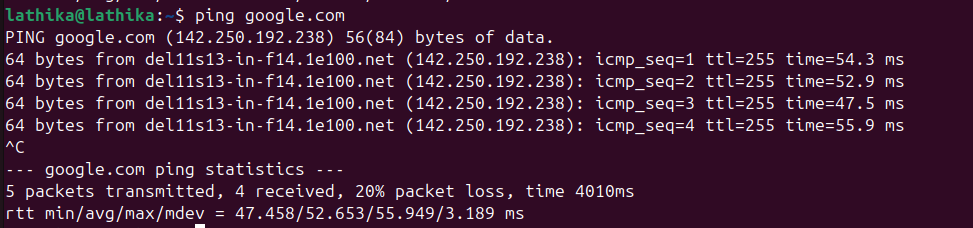
# Practical No -2

## Aim: Linux Network Commands – ifconfig, ip, ping, netstat, traceroute, nslookup, route, hostname.

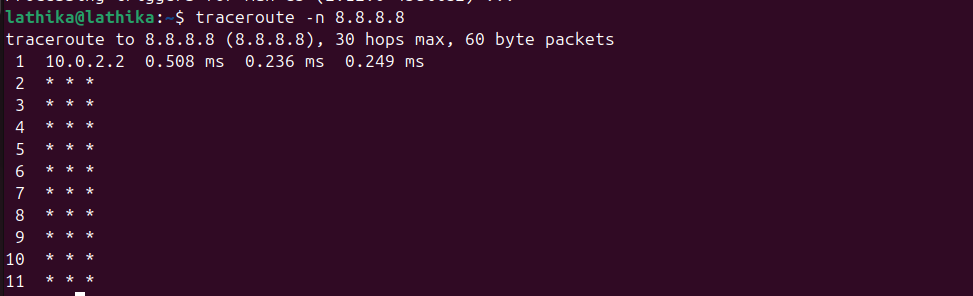
**Code:**

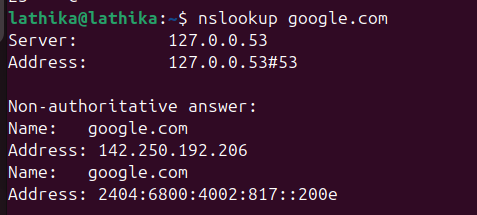
**Command: ifconfig**

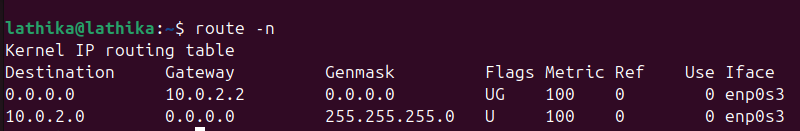
**Command: ip**

**Command: ping**

**Command: netstart**

**Command: traceroute**

**Command: nslookup**

**Command: route**

**Command: hostname**

# Practical No - 3

## Aim: Program to simulate Point to Point topology

**Code:**

**first.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"

// 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[])

{

CommandLinecmd ( FILE ); cmd.Parse (argc, argv);

// Take logs Time::SetResolution (Time::NS);

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

// Take n number of computer NodeContainer nodes; nodes.Create (2);

// Choose your technology to communicate PointToPointHelperpointToPoint;

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

// Install technology on computers NetDeviceContainer devices;

devices = pointToPoint.Install (nodes);

// Asking to follow rules InternetStackHelper stack; stack.Install (nodes);

// Assign Ip address to communicate Ipv4AddressHelper address;

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

Ipv4InterfaceContainer interfaces = address.Assign (devices);

// Create a x type of server on port x UdpEchoServerHelperechoServer (9);

// Install server on a node then Start and Stop the server ApplicationContainerserverApps = echoServer.Install (nodes.Get (1)); serverApps.Start (Seconds (1.0));

serverApps.Stop (Seconds (10.0));

// Create x type of client and set its attributes UdpEchoClientHelperechoClient (interfaces.GetAddress (1), 9); echoClient.SetAttribute ("MaxPackets", UintegerValue (1)); echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0))); echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

// Install the server then Start and Stop it. ApplicationContainerclientApps = echoClient.Install (nodes.Get (0)); clientApps.Start (Seconds (2.0));

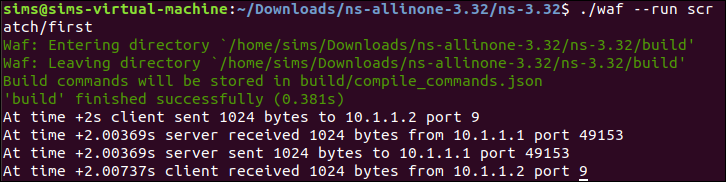
clientApps.Stop (Seconds (10.0));

// NetAnim AnimationInterfaceanim("NetAnimFolder/first.xml"); anim.SetConstantPosition(nodes.Get(0), 10.0, 10.0);

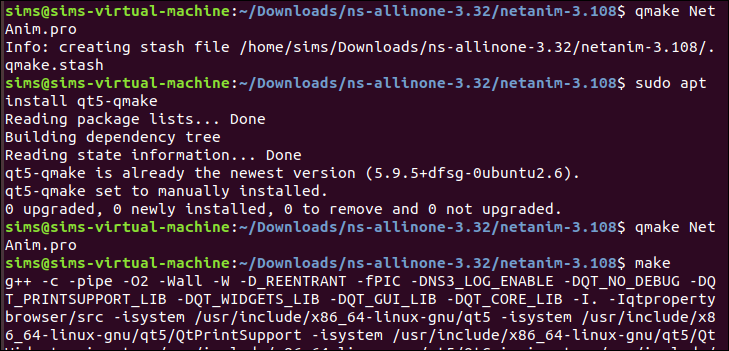
anim.SetConstantPosition(nodes.Get(1), 20.0, 20.0);

// Run the Simulation Simulator::Run (); Simulator::Destroy (); return 0;

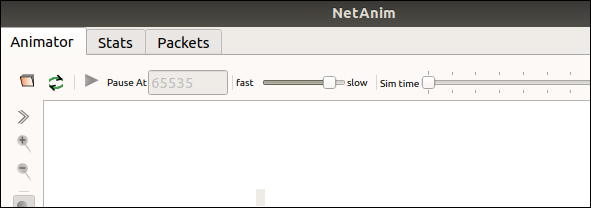
}

**./waf –run scratch/first**

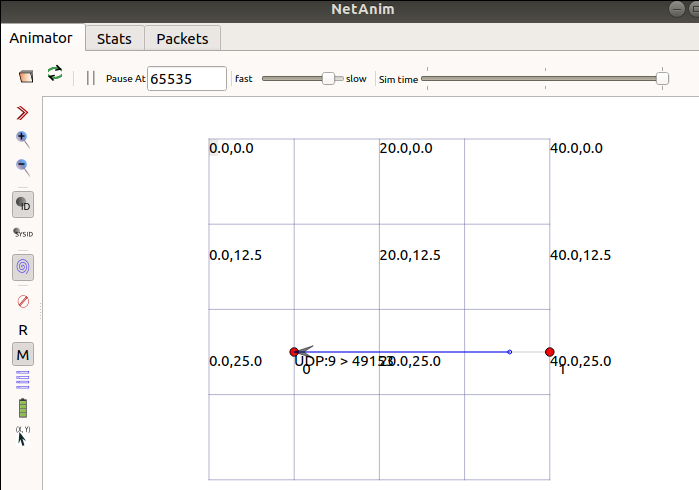
**In new terminal :**

* **qmake NetAnim.pro**
* **make**

**./NetAnim**

**Go to file and open first.xml**

**Output:**



# Practical No - 4

## Aim: Program to simulate Bus topology

**Code:**

**second.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"

#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;

CommandLinecmd ( 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);

NodeContainercsmaNodes; csmaNodes.Add (p2pNodes.Get (1)); csmaNodes.Create (nCsma);

PointToPointHelperpointToPoint;

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

NetDeviceContainer p2pDevices;

p2pDevices = pointToPoint.Install (p2pNodes);

CsmaHelpercsma;

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

NetDeviceContainercsmaDevices; 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);

UdpEchoServerHelperechoServer (9);

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

serverApps.Stop (Seconds (10.0));

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

ApplicationContainerclientApps = echoClient.Install (p2pNodes.Get (0)); clientApps.Start (Seconds (2.0));

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

pointToPoint.EnablePcapAll ("second"); csma.EnablePcap ("second", csmaDevices.Get (1), true);

// NetAnim Simulation Output AnimationInterfaceanim("second.xml");

// Position point-to-point nodes anim.SetConstantPosition(p2pNodes.Get(0), 10.0, 30.0);

anim.SetConstantPosition(p2pNodes.Get(1), 20.0, 30.0);

// Position CSMA (bus) nodes in a row to visualize the bus topology for (uint32\_t i = 0; i<csmaNodes.GetN(); ++i)

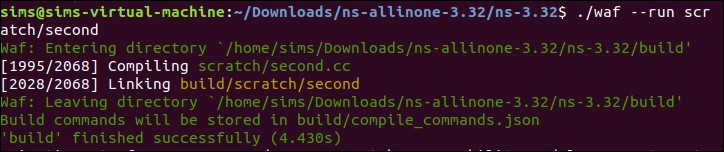
{

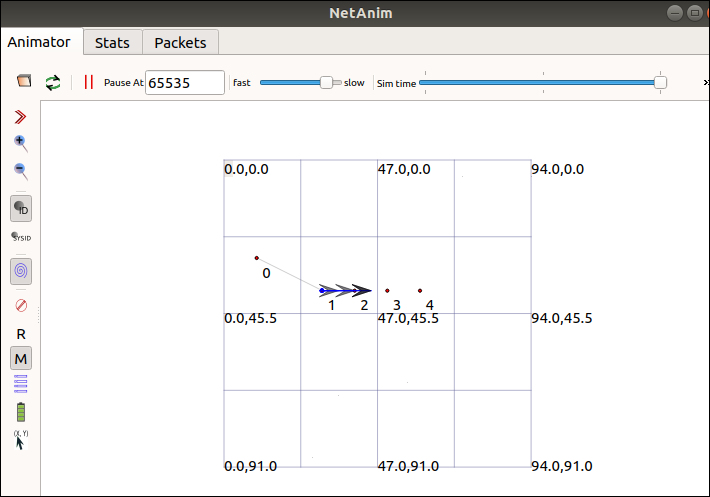
anim.SetConstantPosition(csmaNodes.Get(i), 30.0 + 10 \* i, 40.0);

}

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

}

**Output:**



# Practical No - 5

## Aim: Program to simulate Star topology

**Code:**

**star.cc**

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

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

// Network topology (default)

//

|  |  |  |
| --- | --- | --- |
| //  // | n2 n3 n4  \ | / | .  . |
| // | \|/ | . |
| // | n1--- n0---n5 | . |
| // | /|\ | . |
| // | / | \ | . |
| // | n8 n7 n6 | . |
| // |  |  |

using namespace ns3; NS\_LOG\_COMPONENT\_DEFINE ("Star");

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;

CommandLinecmd ( FILE );

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

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

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)); PacketSinkHelperpacketSinkHelper ("ns3::TcpSocketFactory", hubLocalAddress); ApplicationContainerhubApp = 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 spoke node.

//

OnOffHelperonOffHelper ("ns3::TcpSocketFactory", Address ()); onOffHelper.SetAttribute ("OnTime", StringValue ("ns3::ConstantRandomVariable[Constant=1]")); onOffHelper.SetAttribute ("OffTime", StringValue ("ns3::ConstantRandomVariable[Constant=0]"));

ApplicationContainerspokeApps;

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

{

AddressValueremoteAddress (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.");

Ipv4GlobalRoutingHelper::PopulateRoutingTables (); NS\_LOG\_INFO ("Enable pcap tracing.");

pointToPoint.EnablePcapAll ("star"); AnimationInterfaceanim("NetAnimFolder/star.xml");

anim.SetConstantPosition(star.GetHub(), 50.0, 50.0);

// Arrange spoke nodes in a circle around the hub double radius = 30.0;

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

{

double angle = i \* 2 \* M\_PI / star.SpokeCount();

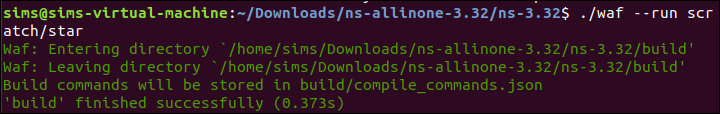
double x = 50.0 + radius \* std::cos(angle); // center at (50,50) double y = 50.0 + radius \* std::sin(angle); anim.SetConstantPosition(star.GetSpokeNode(i), x, y);

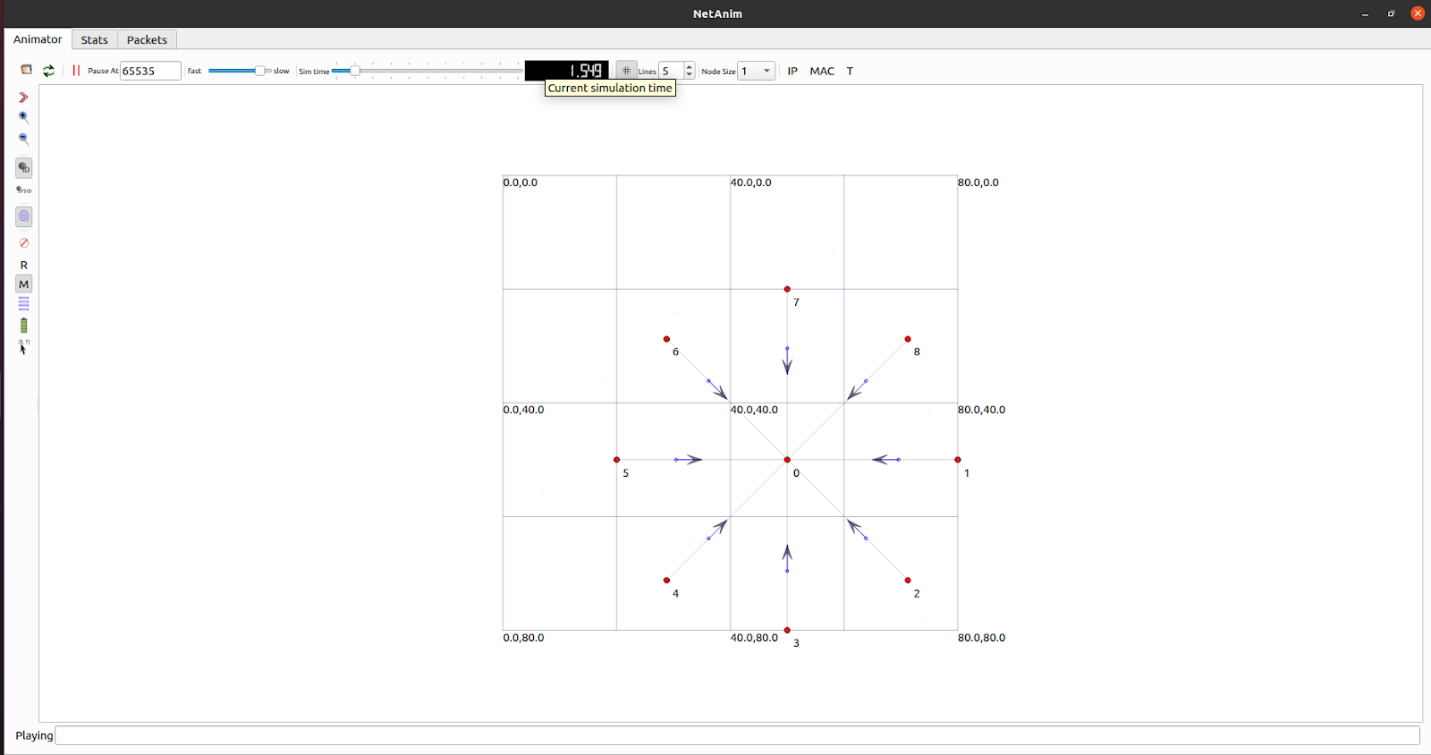
}

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

return 0;

}

**Output:**



# Practical No - 6

## Aim: Program to simulate Mesh topology

**Code:**

**mesh-topology.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"

using namespace ns3;

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

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

{

CommandLinecmd ( FILE ); cmd.Parse (argc, argv);

Time::SetResolution (Time::NS);

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

// Create 4 nodes for the mesh topology (2x2 grid) NodeContainer nodes;

nodes.Create (4);

// Set up point-to-point links between all pairs of nodes PointToPointHelperpointToPoint;

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

NetDeviceContainer devices;

// Create the links between all pairs of nodes for (uint32\_t i = 0; i<nodes.GetN(); ++i)

{

for (uint32\_t j = i + 1; j <nodes.GetN(); ++j)

{

NetDeviceContainerlinkDevices = pointToPoint.Install (nodes.Get (i), nodes.Get (j)); devices.Add (linkDevices);

}

}

// Install the Internet stack on the nodes

InternetStackHelper stack; stack.Install (nodes);

// Assign IP addresses to each device Ipv4AddressHelper address;

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

Ipv4InterfaceContainer interfaces = address.Assign (devices);

// Set up UDP Echo server on the last node (node 3) UdpEchoServerHelperechoServer (9); ApplicationContainerserverApps = echoServer.Install (nodes.Get (3)); serverApps.Start (Seconds (1.0));

serverApps.Stop (Seconds (10.0));

// Set up UDP Echo client on the first node (node 0) UdpEchoClientHelperechoClient (interfaces.GetAddress (3), 9); echoClient.SetAttribute ("MaxPackets", UintegerValue (1)); echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0))); echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

ApplicationContainerclientApps = echoClient.Install (nodes.Get (0)); clientApps.Start (Seconds (2.0));

clientApps.Stop (Seconds (10.0));

// Set mobility model for all nodes (constant position model) MobilityHelper mobility; mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel"); mobility.Install(nodes);

// Set constant positions for the nodes to create a 2x2 grid AnimationInterfaceanim("mesh\_topology.xml");

// Positioning nodes in a 2x2 grid layout

AnimationInterface::SetConstantPosition (nodes.Get(0), 10, 25); // Node 0 at (10, 25)

AnimationInterface::SetConstantPosition (nodes.Get(1), 40, 25); // Node 1 at (40, 25)

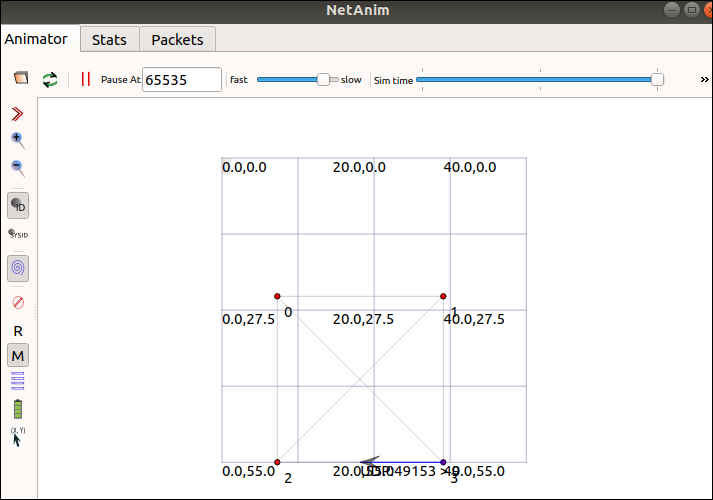
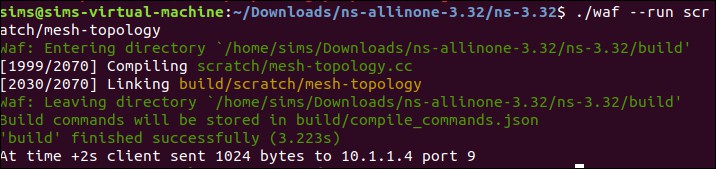
AnimationInterface::SetConstantPosition (nodes.Get(2), 10, 55); // Node 2 at (10, 55)

AnimationInterface::SetConstantPosition (nodes.Get(3), 40, 55); // Node 3 at (40, 55)

// Enable packet metadata in the animation output anim.EnablePacketMetadata(true);

// Run the simulation Simulator::Run (); Simulator::Destroy (); return 0;

}

**Output:**

# Practical No - 7

## Aim: Program to simulate Hybrid topology

**Code:**

**third.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"

#include "ns3/netanim-module.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;

CommandLinecmd ( 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);

PointToPointHelperpointToPoint;

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

NetDeviceContainer p2pDevices;

p2pDevices = pointToPoint.Install (p2pNodes);

NodeContainercsmaNodes; csmaNodes.Add (p2pNodes.Get (1)); csmaNodes.Create (nCsma);

CsmaHelpercsma;

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

NetDeviceContainercsmaDevices; csmaDevices = csma.Install (csmaNodes);

NodeContainerwifiStaNodes; wifiStaNodes.Create (nWifi); NodeContainerwifiApNode = p2pNodes.Get (0);

YansWifiChannelHelper channel = YansWifiChannelHelper::Default (); YansWifiPhyHelperphy = YansWifiPhyHelper::Default (); phy.SetChannel (channel.Create ());

WifiHelperwifi;

wifi.SetRemoteStationManager ("ns3::AarfWifiManager"); WifiMacHelper mac;

Ssidssid = Ssid ("ns-3-ssid"); mac.SetType ("ns3::StaWifiMac",

"Ssid", SsidValue (ssid), "ActiveProbing", BooleanValue (false));

NetDeviceContainerstaDevices;

staDevices = wifi.Install (phy, mac, wifiStaNodes);

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

NetDeviceContainerapDevices;

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);

UdpEchoServerHelperechoServer (9);

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

serverApps.Stop (Seconds (10.0));

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

ApplicationContainerclientApps = 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 == true)

{

pointToPoint.EnablePcapAll ("third"); phy.EnablePcap ("third", apDevices.Get (0));

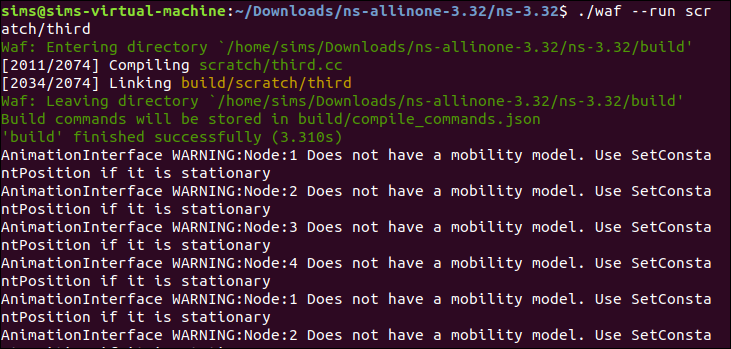
csma.EnablePcap ("third", csmaDevices.Get (0), true);

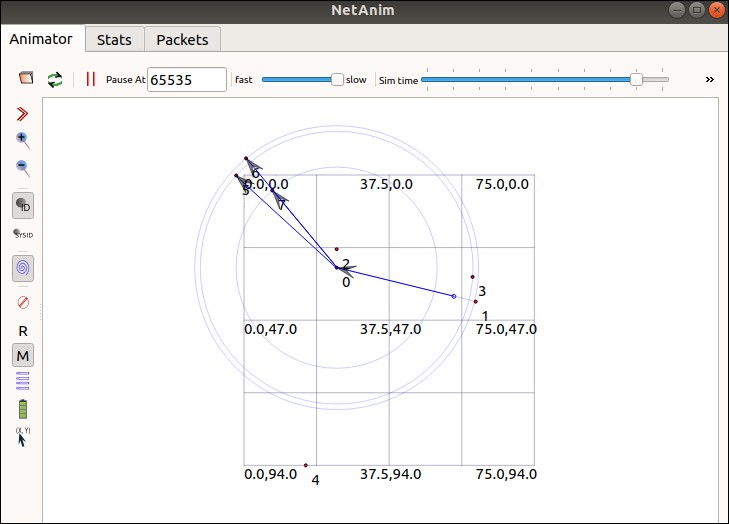
}

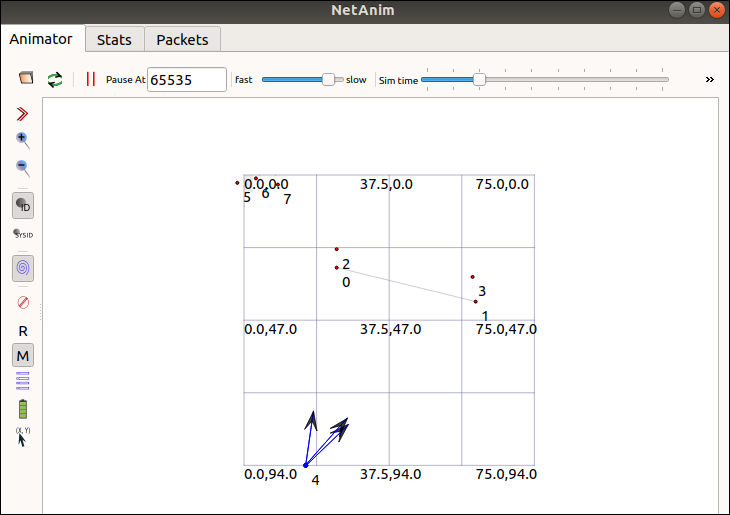
AnimationInterfaceanim("third.xml"); anim.SetConstantPosition(p2pNodes.Get(0), 30.0, 30.0);

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

}

**Output:**



****

# Practical No - 8

## Aim: Program to simulate UDP Client Server

**Code:**

**udp-echo.cc**

// Network topology

//

// n0 n1 n2 n3

// | | | |

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

// LAN

//

// - UDP flows from n0 to n1 and back

// - DropTail queues

// - Tracing of queues and packet receptions to file "udp-echo.tr"

#include <fstream>

#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 ("UdpEchoExample"); int

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

{

// Enable logging

LogComponentEnable ("UdpEchoExample", LOG\_LEVEL\_INFO); LogComponentEnable ("UdpEchoClientApplication", LOG\_LEVEL\_INFO); LogComponentEnable ("UdpEchoServerApplication", LOG\_LEVEL\_INFO); bool useV6 = false;

Address serverAddress; CommandLinecmd ( FILE );

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

NS\_LOG\_INFO ("Creating 4 nodes."); NodeContainer n;

n.Create (4);

InternetStackHelper internet; internet.Install (n);

NS\_LOG\_INFO ("Creating CSMA channel."); CsmaHelpercsma;

csma.SetChannelAttribute ("DataRate", DataRateValue (DataRate (5000000))); csma.SetChannelAttribute ("Delay", TimeValue (MilliSeconds (2))); csma.SetDeviceAttribute ("Mtu", UintegerValue (1400));

NetDeviceContainer d = csma.Install (n);

NS\_LOG\_INFO ("Assigning IP Addresses."); if (!useV6)

{

Ipv4AddressHelper ipv4;

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

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

NS\_LOG\_INFO ("Server IPv4 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 ("Server IPv6 Address: " << i6.GetAddress(1,1));

}

NS\_LOG\_INFO ("Installing UDP Echo Server on Node 1."); uint16\_t port = 9;

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

apps.Stop (Seconds (10.0));

NS\_LOG\_INFO ("Installing UDP Echo Client on Node 0."); uint32\_t packetSize = 1024;

uint32\_t maxPacketCount = 1;

Time interPacketInterval = Seconds (1.0); UdpEchoClientHelper client (serverAddress, port);

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

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

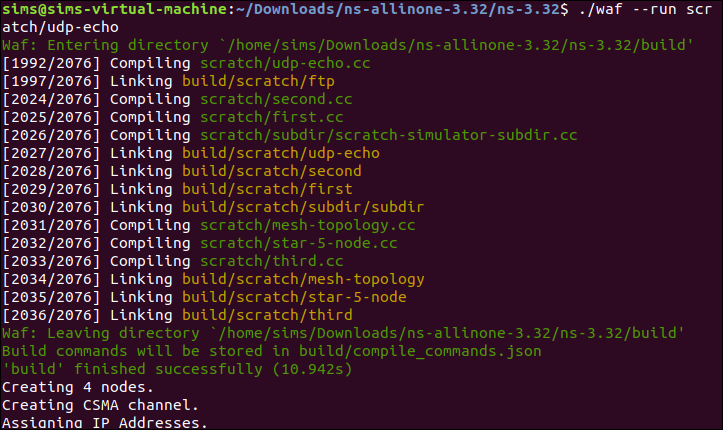
apps.Stop (Seconds (10.0)); NS\_LOG\_INFO ("Tracing enabled."); AsciiTraceHelper ascii;

csma.EnableAsciiAll (ascii.CreateFileStream ("udp-echo.tr")); csma.EnablePcapAll ("udp-echo", false);

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

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

}

**Output:**

# Practical No - 9

## Aim: Program to simulate DHCP Server and Clients

**Code:**

**dhcp.cc**

#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[])

{

CommandLinecmd ( 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); if (verbose)

{

LogComponentEnable ("DhcpExample", LOG\_LEVEL\_INFO); 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 CSMA channels."); CsmaHelpercsma;

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

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

p2pNodes.Create (1);

NS\_LOG\_INFO ("Create point-to-point channel."); PointToPointHelperpointToPoint;

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

NetDeviceContainer p2pDevices;

p2pDevices = pointToPoint.Install (p2pNodes);

NS\_LOG\_INFO ("Install internet stack."); InternetStackHelpertcpip;

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

NS\_LOG\_INFO ("Assign IP to point-to-point link."); Ipv4AddressHelper address;

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

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

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 ("Set up DHCP.");

DhcpHelperdhcpHelper;

Ipv4InterfaceContainer fixedNodes = dhcpHelper.InstallFixedAddress ( devNet.Get (4), Ipv4Address ("172.30.0.17"), Ipv4Mask ("/24")); fixedNodes.Get (0).first->SetAttribute ("IpForward", BooleanValue (true));

ApplicationContainerdhcpServerApp = 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"));

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

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

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

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

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

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

dhcpClients.Stop (stopTime);

NS\_LOG\_INFO ("Set up echo server and client."); UdpEchoServerHelperechoServer (9);

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

serverApps.Stop (stopTime);

UdpEchoClientHelperechoClient (p2pInterfaces.GetAddress (1), 9); echoClient.SetAttribute ("MaxPackets", UintegerValue (100)); echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0))); echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

ApplicationContainerclientApps = echoClient.Install (nodes.Get (1)); clientApps.Start (Seconds (10.0));

clientApps.Stop (stopTime);

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

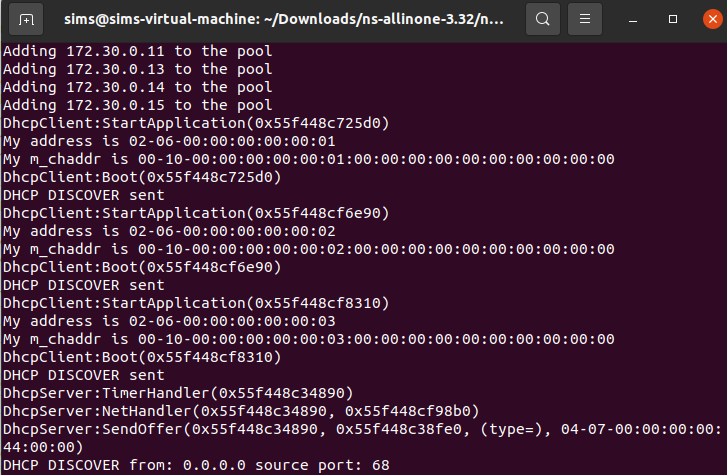
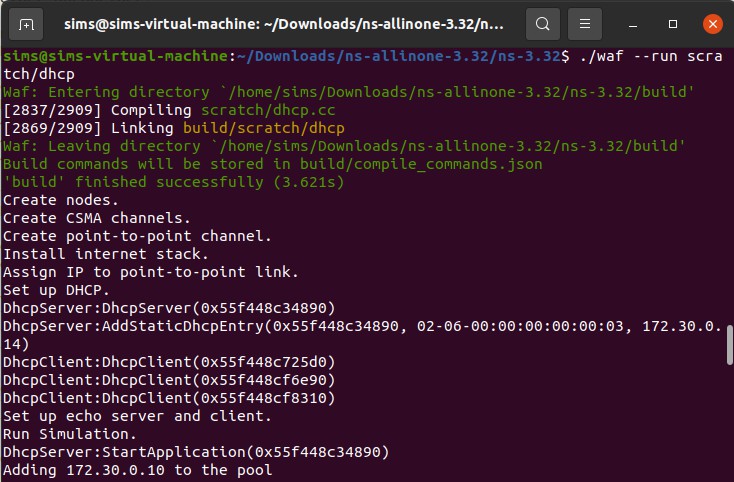
{

NS\_LOG\_INFO ("Enable 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 – 10

## Aim: Program to simulate FTP using TCP

**Code:**

**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

// CommandLinecmd;

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).

// PointToPointHelperpointToPoint;

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 BulkSendHelpersource("ns3::TcpSocketFactory",

InetSocketAddress(i.GetAddress(1), port));

// Set the amount of data to send in bytes. Zero is unlimited. source.SetAttribute("MaxBytes", UintegerValue(maxBytes)); ApplicationContainersourceApps = source.Install(nodes.Get(0)); sourceApps.Start(Seconds(0.0)); sourceApps.Stop(Seconds(10.0));

//

// Create a PacketSinkApplication and install it on node 1

// PacketSinkHelpersink("ns3::TcpSocketFactory",

InetSocketAddress(Ipv4Address::GetAny(), port)); ApplicationContainersinkApps = 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);

}

//

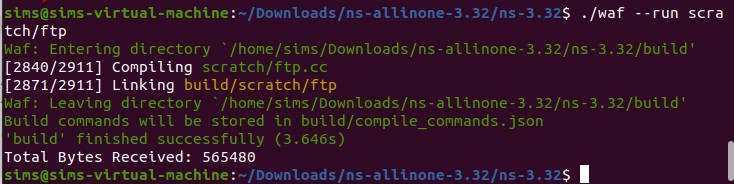
// Now, do the actual simulation.

//

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:**

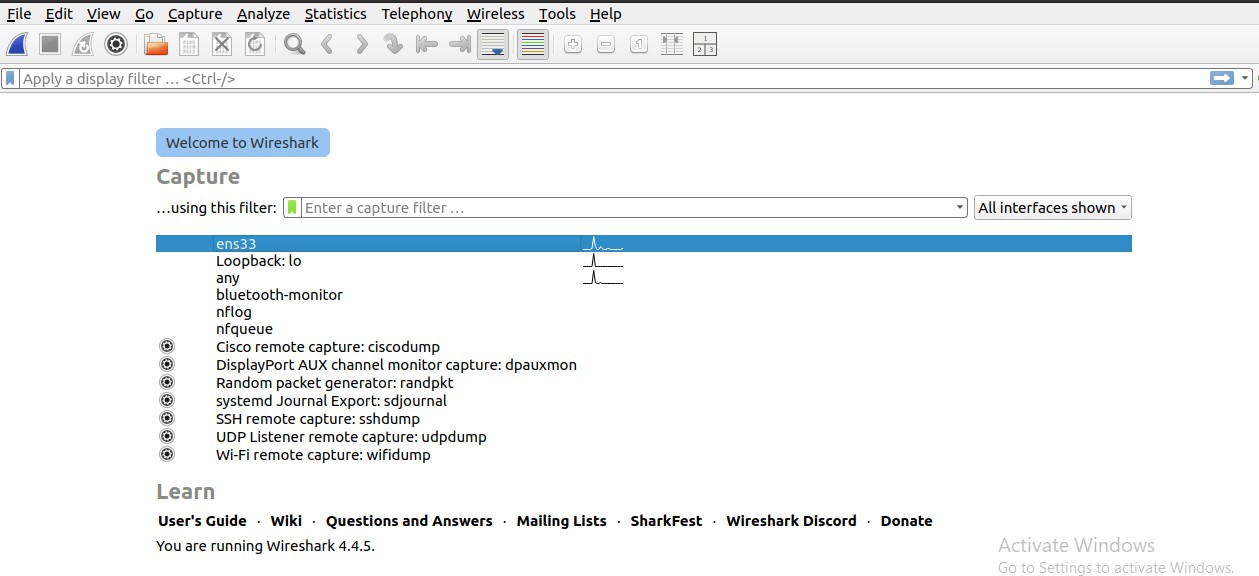
# Practical No - 11

**Aim: Exercises for analyzing network protocols using Wireshark:**

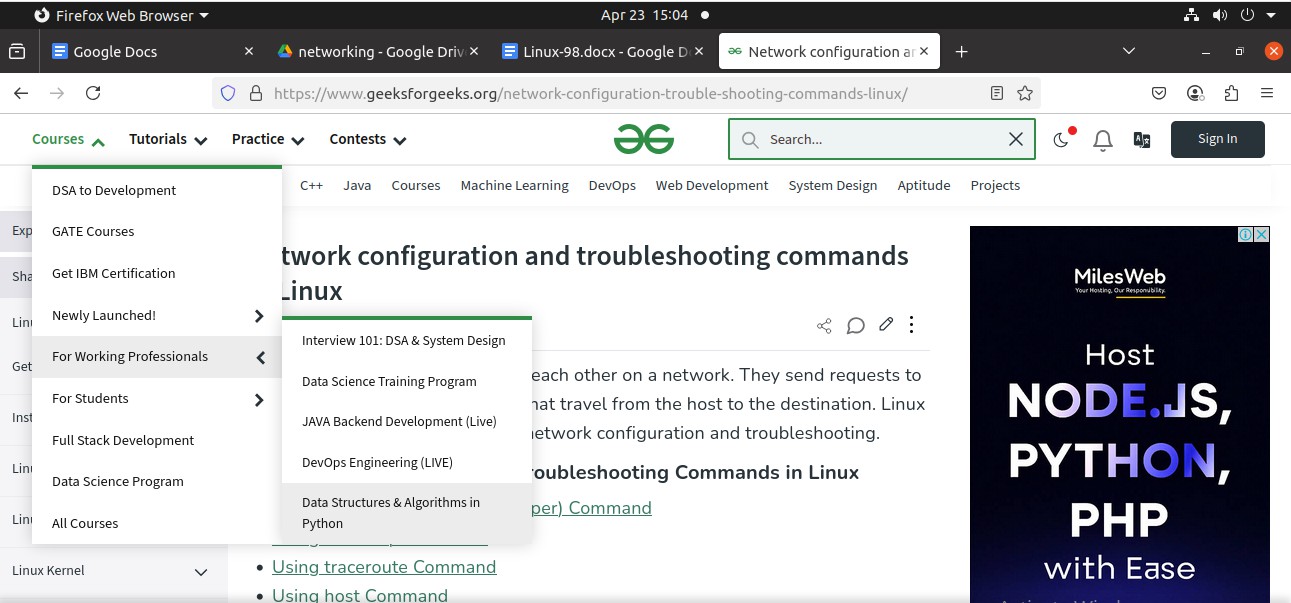
* **Capture the packets while browsing any website**
* **Analyze the header fields of various protocols**

**Code:**

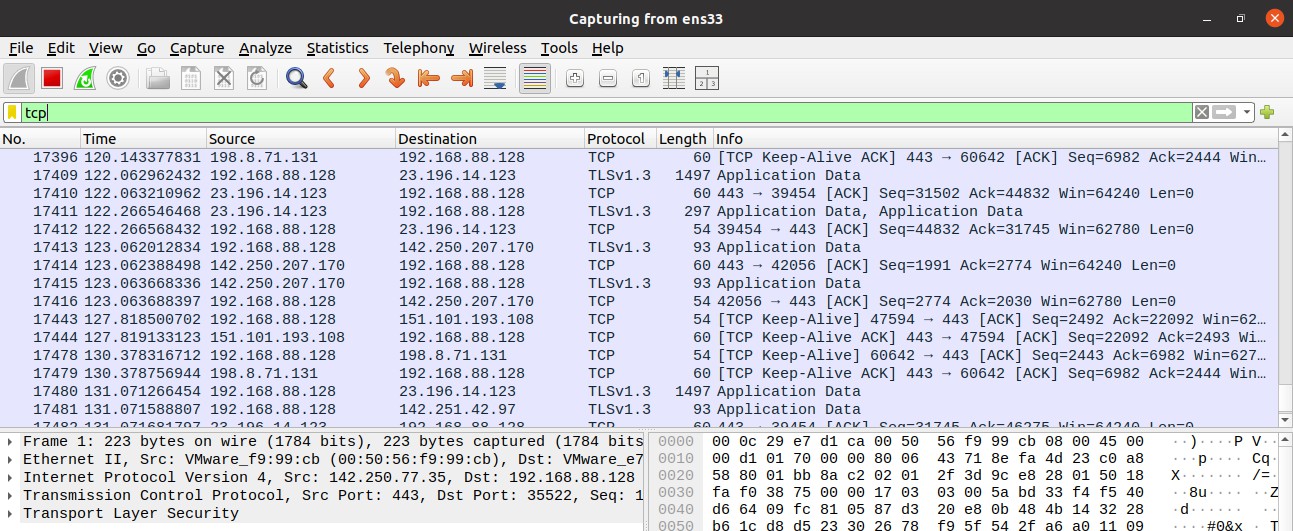
**Start Wireshark**

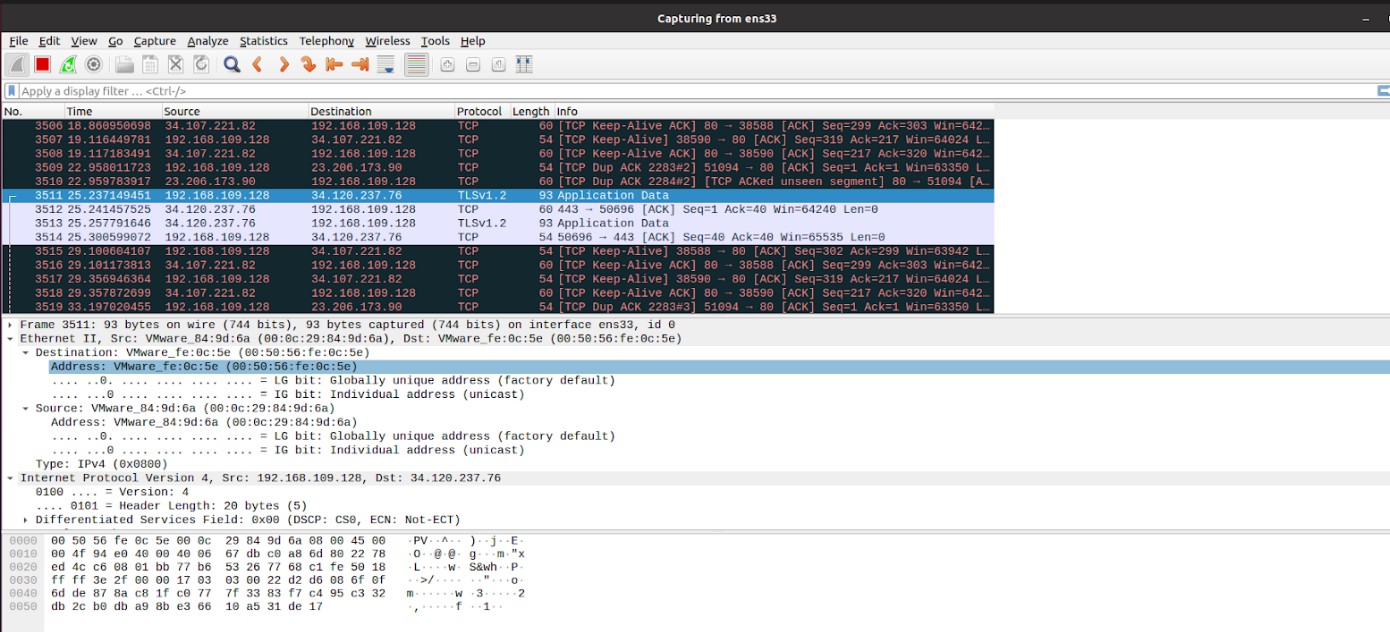


**http**[**s://www.geeksforgeeks.com/**](http://www.geeksforgeeks.com/)



**TCP**



**Header (Ethernet II)**

**Statistics (I/O Graph)**

