

**Deccan Education Society's
Navinchandra Mehta Institute of
Technology and Development**

C E R T I F I C A T E

This is to certify that **Mr. Krutarth Prasad Bodas** of M.C.A. Semester II
with

Roll No.**C23021** has complete All practical's of **MCAL26 – Networking with Linux** under my supervision in this college during the year 2023-2024.

CO	R1: Journal	R2: Performance during lab session	R3: Implementation using different problem solving techniques	R4: Mock Viva	Attendance
CO1					
CO2					
CO3					
CO4					

Practical-in-charge

Head of Department

MCA Department
(NMITD)

Index

Sr. No	Problem Statement	Course Outcome	Date	Sign
1	Installation of NS-3 in Linux	CO1	22-02-24	
2	Installation of NS-3 in Linux	CO1	22-02-24	
3	Installation of NS-3 in Linux	CO1	04-03-24	
4	Program to simulate traffic between two nodes	CO2	04-03-24	
5	Program to simulate star topology	CO2	20-03-24	
6	Program to simulate bus topology	CO2	20-03-24	
7	Program to simulate mesh topology	CO2	27-03-24	
8	Program to simulate hybrid topology	CO2	27-03-24	
9	Program to simulate UDP server	CO2, CO3	02-04-24	
10	Program to simulate UDP server client	CO2, CO3	16-04-24	
11	Program to simulate DHCP server and n clients	CO2, CO3	16-04-24	
12	Animate a simple network using Net Anim in Network Simulator	CO3	11-03-24	

13	Analyze the network traffic using Wire Shark	CO4	11-03-24	
14	Mini Project of Write a program to demonstrate Hybrid Topology in ns3	CO4	11-03-24	

Practical 1: Installing NS-3 in Ubuntu

Network simulator is a tool used for simulating the real world network on one computer by writing scripts in C++ or Python. Normally if we want to perform experiments, to see how our network works using various parameters. We don't have required number of computers and routers for making different topologies. Even if we have these resources it is very expensive to build such a network for experiment purposes.

So to overcome these drawbacks we used NS3, which is a discrete event network simulator for Internet. NS3 helps to create various virtual nodes (i.e., computers in real life) and with the help of various Helper classes it allows us to install devices, internet stacks, application, etc to our nodes.

Using NS3 we can create PointToPoint, Wireless, CSMA, etc connections between nodes. PointToPoint connection is same as a LAN connected between two computers. Wireless connection is same as WiFi connection between various computers and routers. CSMA connection is same as bus topology between computers. After building connections we try to install NIC to every node to enable network connectivity.

When network cards are enabled in the devices, we add different parameters in the channels (i.e., real world path used to send data) which are data-rate, packet size, etc. Now we use Application to generate traffic and send the packets using these applications.

Steps for installing NS-3 in Ubuntu

Step 1: Update the system

```
$ sudo apt update
```

Step 2: Prerequisites for installing NS-3

```
$ sudo apt install build-essential autoconf automake libxml2-dev g++ python3
python3-dev pkg-config sqlite3 cmake python3-setuptools git qtbase5-dev qtchooser
qt5-qmake qtbase5-dev-tools gir1.2-goocanvas-2.0 python3-gi python3-gi-cairo
python3-pygraphviz gir1.2-gtk-3.0 ipython3 openmpi-bin openmpi-common
openmpi-doc libopenmpi-dev autoconf cvs bzip2 unrar gsl-bin libgsl-dev libgslcblas0
wireshark tcpdump sqlite3 libsqlite3-dev libxml2 libxml2-dev libc6-dev libc6-
dev-i386 libclang-dev llvm-dev automake python3-pip libxml2 libxml2-dev libboost-
all-dev
```

Step 3: Now download the ns3 3.35 from <https://nsnam.org>

Step 4 : Copy the softwares from the Downloads/ folder to the home folder (in my case its /home/ns-3/)

Now extract both the versions using the GUI method.

Just right click and click "Extract Here"

Now we will install ns-3.35

Step 5 :

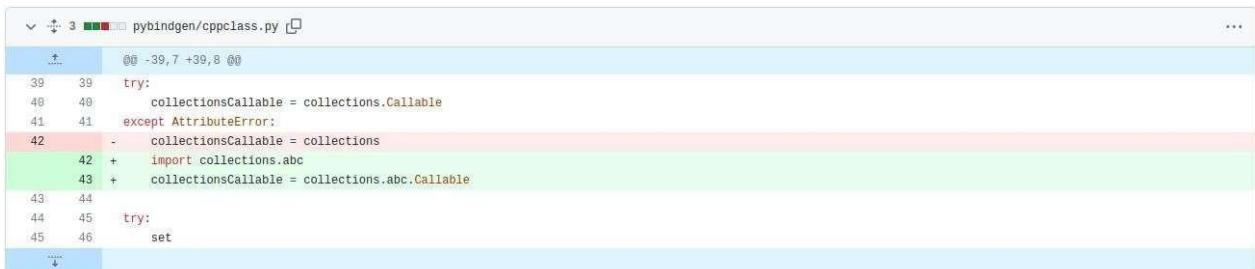
```
$ cd
```

```
$ cd ns-allinone-3.35/
```

```
$ ./build.py --enable-examples --enable-tests
```

In case, if you get the following error pybindgen(ns3 module antenna)

Do this step and repeat the above step



```
v 3 pybindgen/cppclass.py
 39 39 try:
 40 40     collectionsCallable = collections.Callable
 41 41 except AttributeError:
 42 -     collectionsCallable = collections
 42 +     import collections.abc
 43 +     collectionsCallable = collections.abc.Callable
 43 44
 44 45     try:
 45 46         set
```

We have installed two version of ns-3.35 successfully in Ubuntu

Practical 2: Install NetAnim in Ubuntu

NetAnim is an offline network animator tool that now ships along with the ns-allinone-3.xx package.

It can animate the ns-3 network simulation using an XML trace file that is generated as an output during simulation.

So, the necessary steps for creating this XML trace file and setting its related attributes should be done in the ns-3 simulation code itself.

For a beginner who starts learning ns2, the animation that is created by NetAnim will be a more attractive factor in learning and understanding a simulation. Undoubtedly, each and every beginner of ns-3 will be very happy while seeing the first animator output of their simulation on NetAnim for the very first time. So let us enjoy seeing our first network animation on NetAnim.

Steps to Install NetAnim

You can directly install NetAnim

Otherwise, you have to execute some commands but for this we need NS3 installed or compiled.

Step1: sudo apt-get install NetAnim

Step2: NetAnim file.xml

Step3: Select Xml File

Step4: Run the simulation by clicking, NS3 NetAnim successfully.

Practical 3: Install Wireshark In Ubuntu

Wireshark is an open-source packet analyzer, which is used for education, analysis, software development, communication protocol development, and network troubleshooting.

It is used to track the packets so that each one is filtered to meet our specific needs. It is commonly called as a sniffer, network protocol analyzer, and network analyzer. It is also used by network security engineers to examine security problems.

Wireshark is a free to use application which is used to apprehend the data back and forth. It is often called as a free packet sniffer computer application. It puts the network card into an unselective mode, i.e., to accept all the packets which it receives

Install WireShark

Step 1: Add the stable official PPA. To do this, go to terminal by pressing Ctrl+Alt+T and run:

sudo add-apt-repository ppa:wireshark-dev/stable

Step 2: Update the repository:

sudo apt-get update

Step 3: Install wireshark 2.0:

sudo apt-get install wireshark

Step 4: Run wireshark:

sudo wireshark

If you get a error couldn't run /usr/bin/dumpcap in child process: Permission Denied. go to the terminal again and run:

sudo dpkg-reconfigure wireshark-common

Say YES to the message box. This adds a wireshark group. Then add user to the group by typing

sudo adduser \$USER wireshark

Practical 4: Program to simulate traffic between two nodes

Code:

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

```

```
#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"
//netanimation
#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);
```

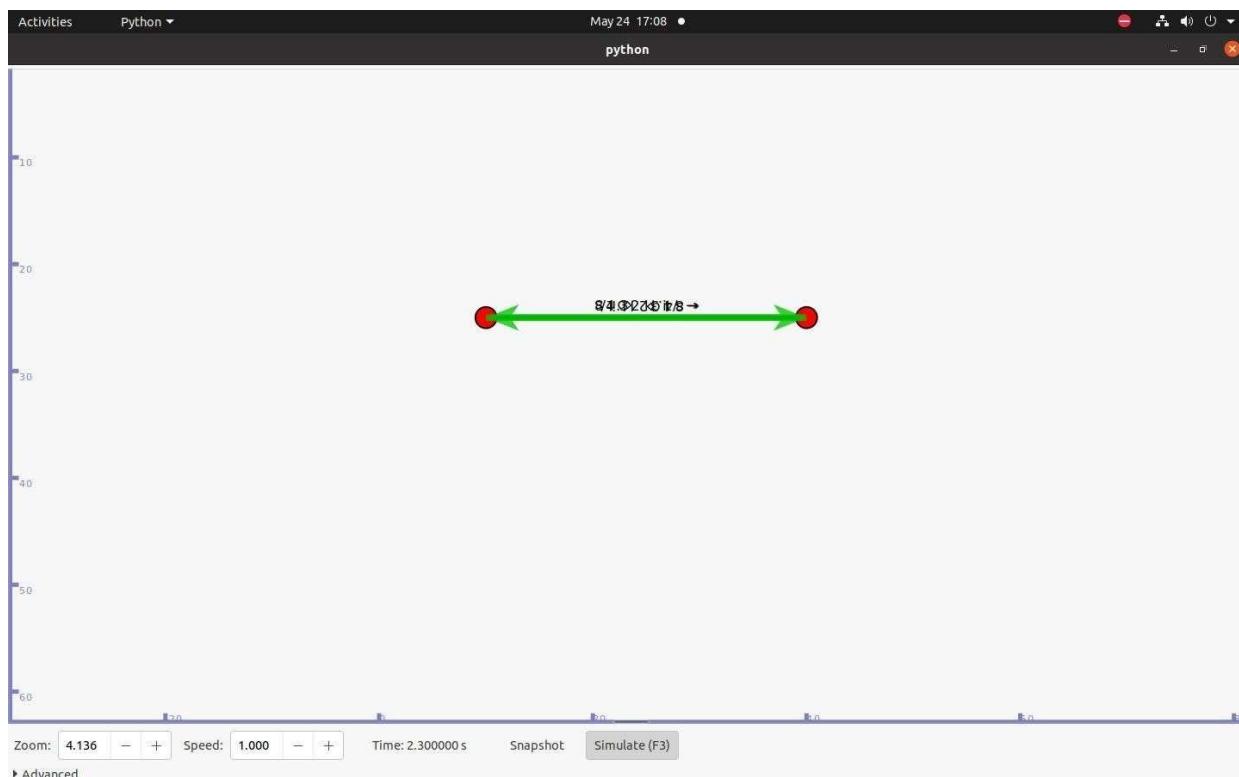
```
ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));  
serverApps.Start (Seconds (1.0));  
serverApps.Stop (Seconds (10.0));
```

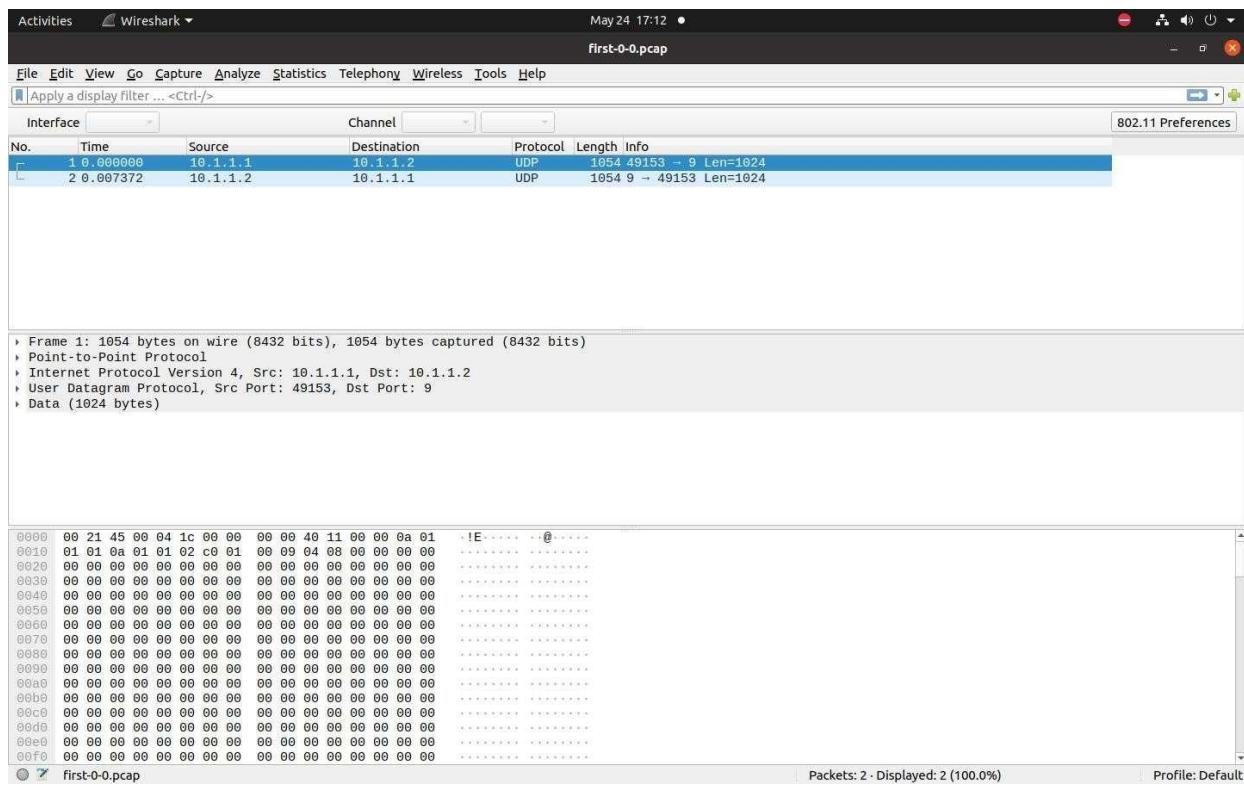
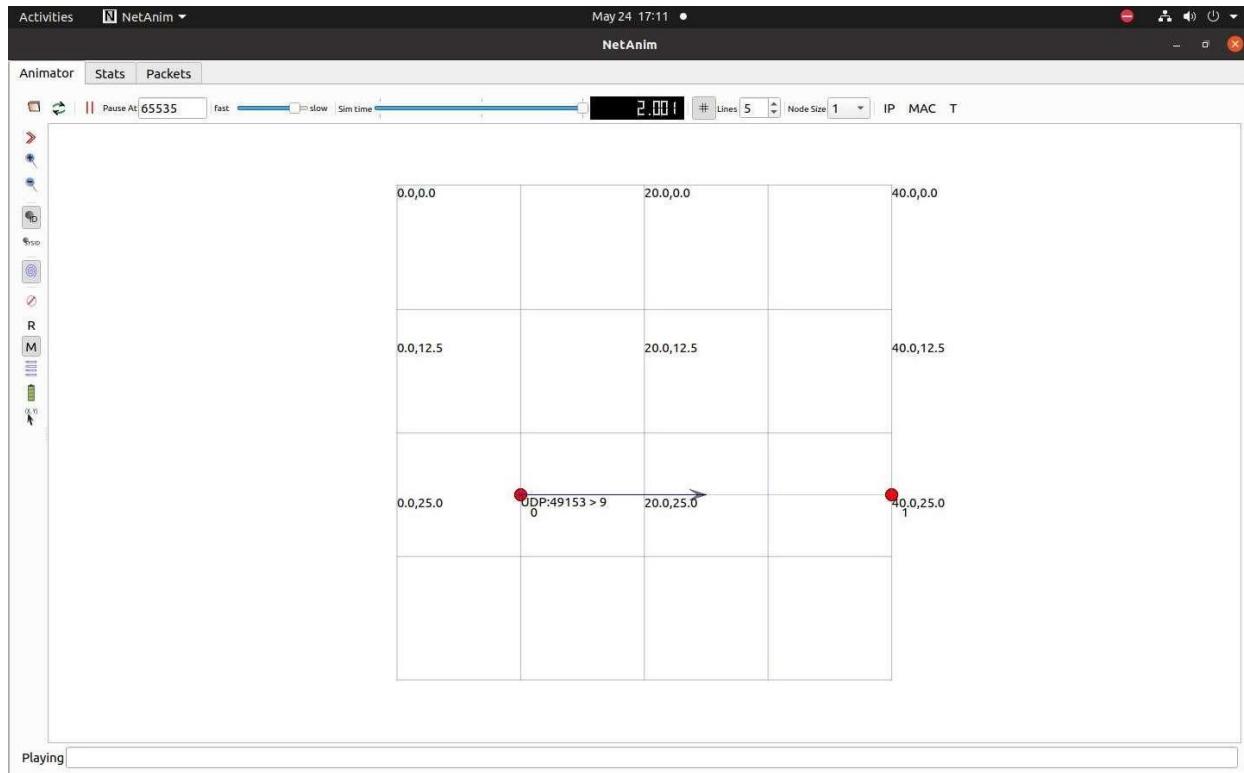
```
UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);  
echoClient.SetAttribute ("MaxPackets", UintegerValue (1));  
echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));  
echoClient.SetAttribute ("PacketSize", UintegerValue (1024));
```

```
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("first");  
Simulator::Run();  
Simulator::Destroy();  
return 0;  
}
```

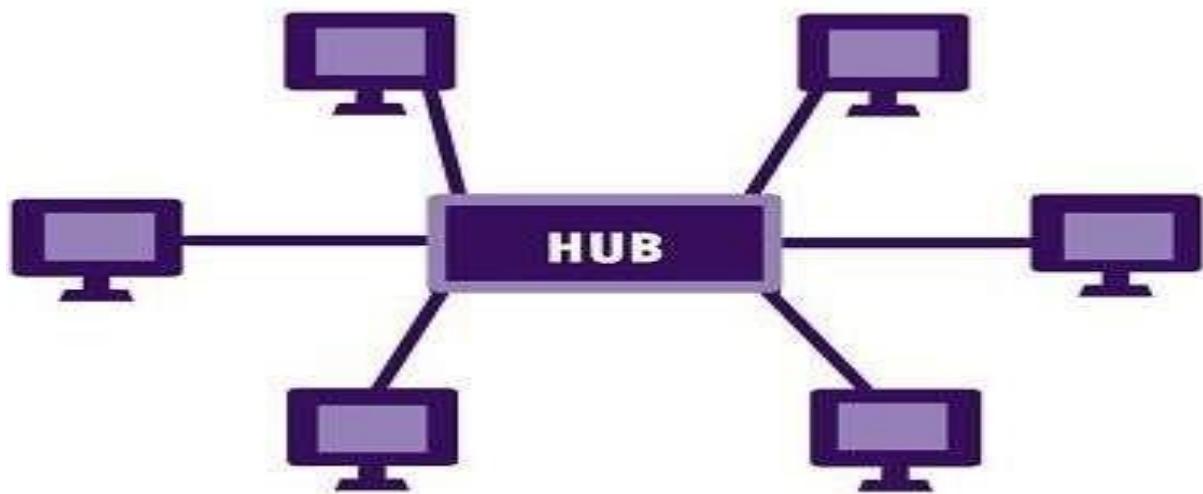
OutPut:





Practical 5: Program to simulate star topology

A star topology, sometimes known as a star network, is a network topology in which each device is connected to a central hub. It is one of the most prevalent computer network configurations, and it's by far the most popular Network Topology. In this network arrangement, all devices linked to a central network device are displayed as a star.



Code:

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 */
```

```
*  
* You should have received a copy of the GNU General Public License  
* along with this program; if not, write to the Free Software  
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA  
*  
*/
```

```
#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;

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

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"));

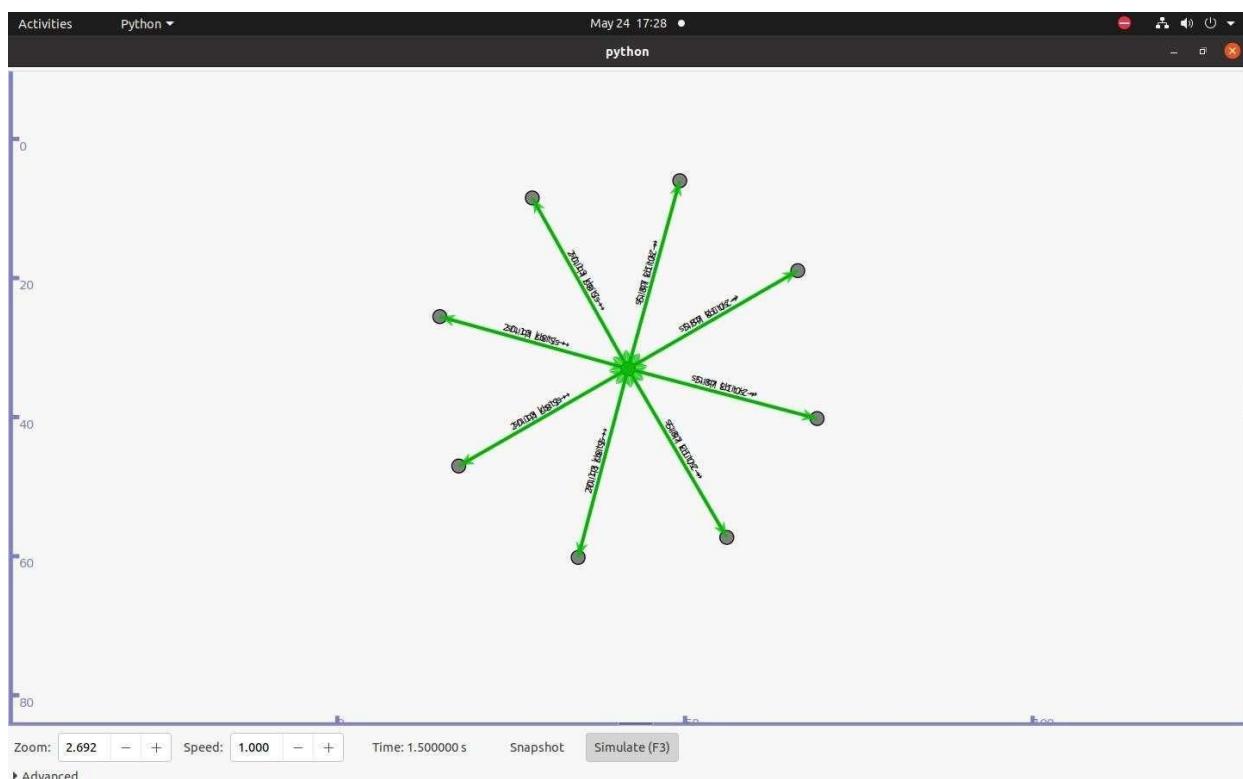
NS_LOG_INFO ("Create applications.");
//
// Create a packet sink on the star "hub" to receive packets.
//
uint16_t port = 50000;
Address hubLocalAddress (InetSocketAddress (Ipv4Address::GetAny (), port));
PacketSinkHelper packetSinkHelper ("ns3::TcpSocketFactory", hubLocalAddress);
ApplicationContainer hubApp = packetSinkHelper.Install (star.GetHub ());
hubApp.Start (Seconds (1.0));
hubApp.Stop (Seconds (10.0));

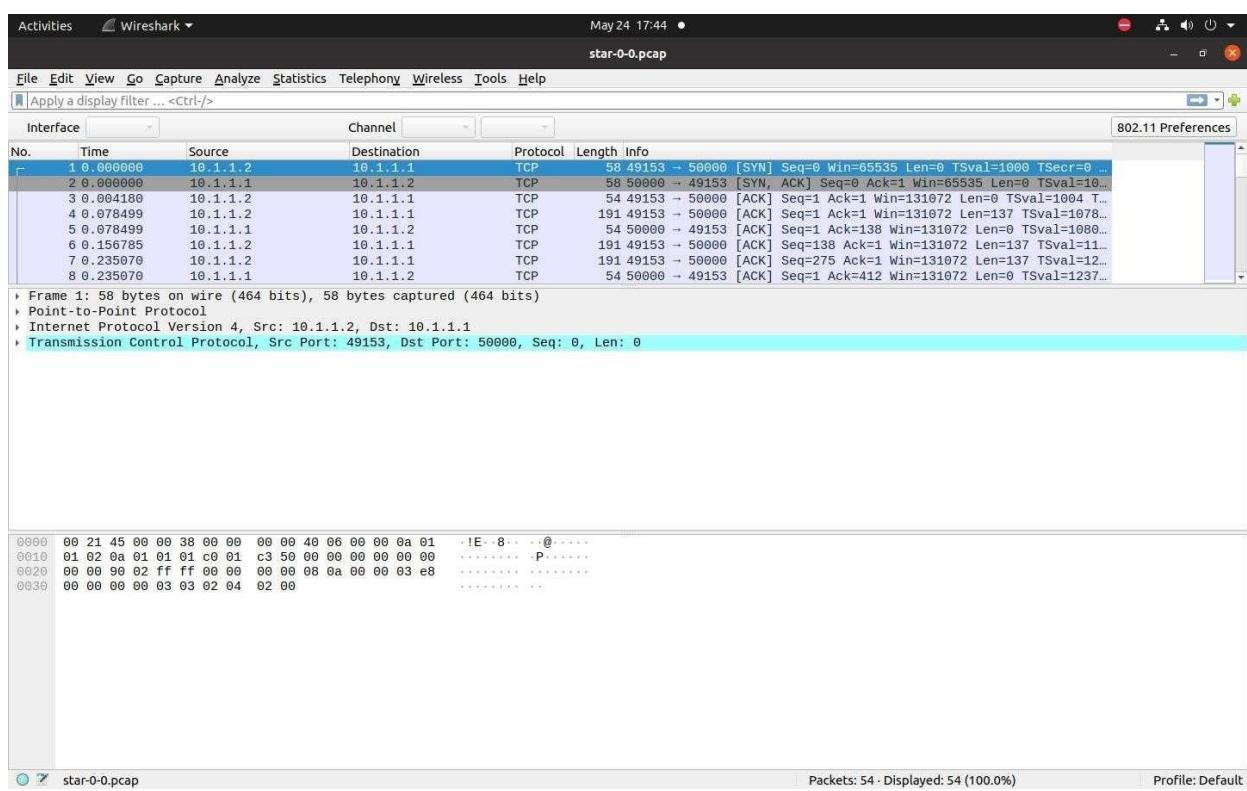
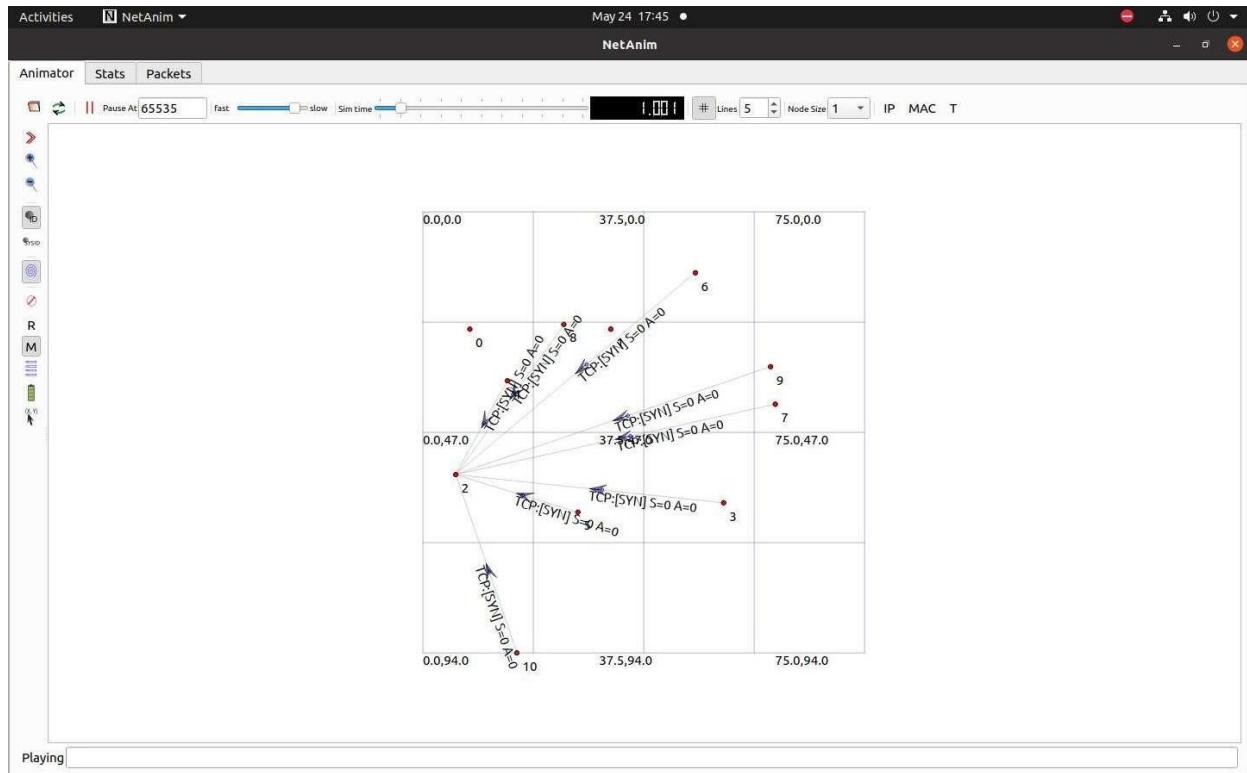
//
// Create OnOff applications to send TCP to the hub, one on each spoke node.
//
OnOffHelper onOffHelper ("ns3::TcpSocketFactory", Address ());
```

```
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 the star.  
//  
Ipv4GlobalRoutingHelper::PopulateRoutingTables ();  
  
NS_LOG_INFO ("Enable pcap tracing.");  
//  
// Do pcap tracing on all point-to-point devices on all nodes.  
//  
pointToPoint.EnablePcapAll ("star");
```

```
NS_LOG_INFO ("Run Simulation.");
Simulator::Run ();
Simulator::Destroy ();
NS_LOG_INFO ("Done.");
return 0;
}
```

OutPut:

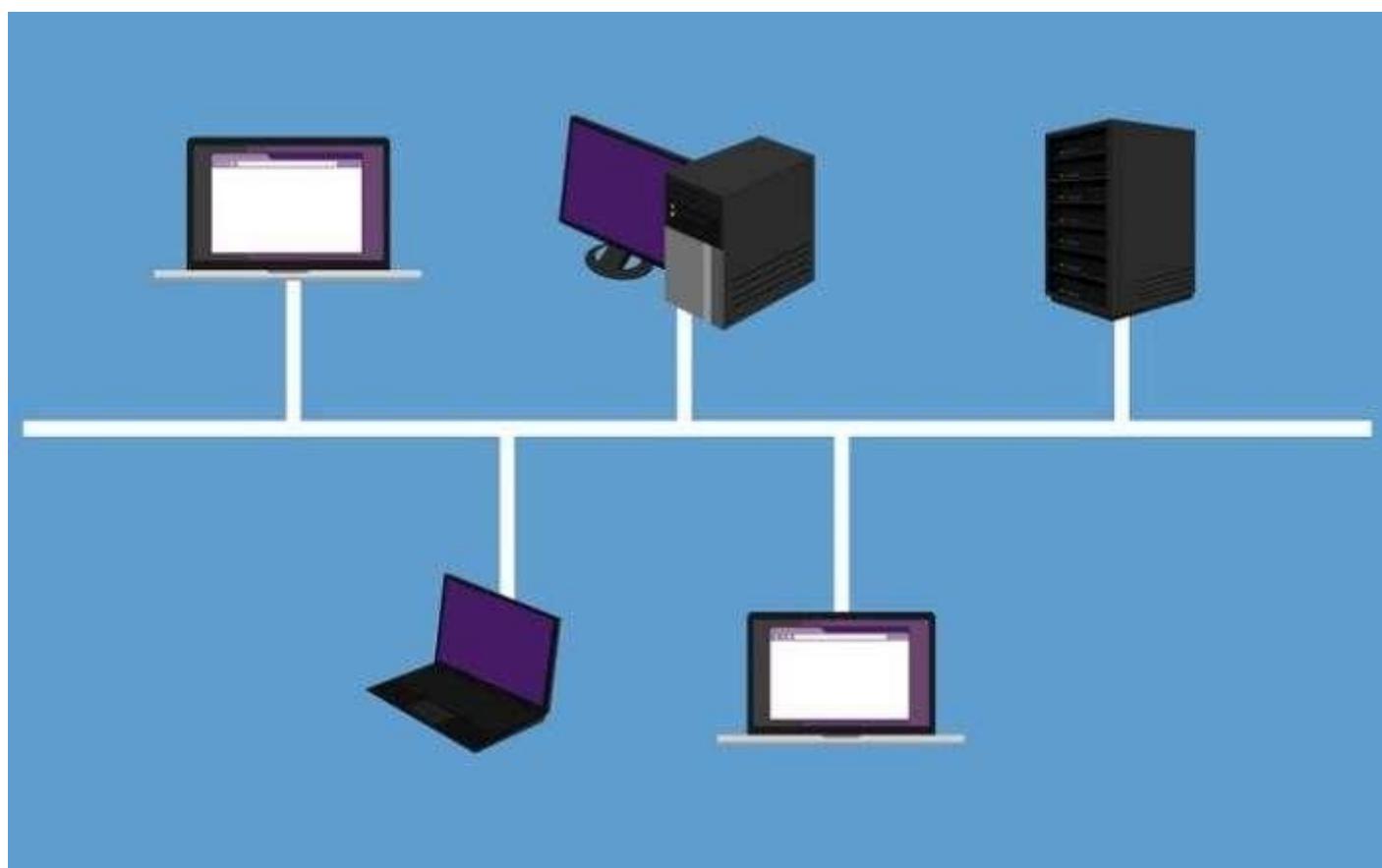




Practical 6: Program to simulate bus topology

The bus network topology is also referred to as horizontal topology. This topology is very common among local area networks. In this network, every computer is linked to a single connection line or cable through an interface. Thus each computer can directly communicate with other computer or device in the network.

Each work station or node has a particular address, and to access a specific node, a user just needs to know its address. There is no concentration point in the network, and thus the problem of resolution is severe. Traffic flow among various computers in the network is relatively simple because it grants all stations to get each transmission. So a single station can broadcast to multiple stations.



Code:

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

```
/*
```

- * This program is free software; you can redistribute it and/or modify
- * it under the terms of the GNU General Public License version 2 as
- * published by the Free Software Foundation;

*

* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.

*

* You should have received a copy of the GNU General Public License
* along with this program; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

```
#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"
//netanimation
#include "ns3/netanim-module.h"
#include "ns3/mobility-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 nodes;  
nodes.Create (2);
```

```
NodeContainer csmaNodes;  
csmaNodes.Add (nodes.Get (1));  
csmaNodes.Create (nCsma);
```

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

```
NetDeviceContainer p2pDevices;  
p2pDevices = pointToPoint.Install (nodes);
```

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

```
NetDeviceContainer csmaDevices;  
csmaDevices = csma.Install (csmaNodes);
```

```
InternetStackHelper stack;  
stack.Install (nodes.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 (nodes.Get (0));
```

```
clientApps.Start (Seconds (2.0));
```

```
clientApps.Stop (Seconds (10.0));
```

```
Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
```

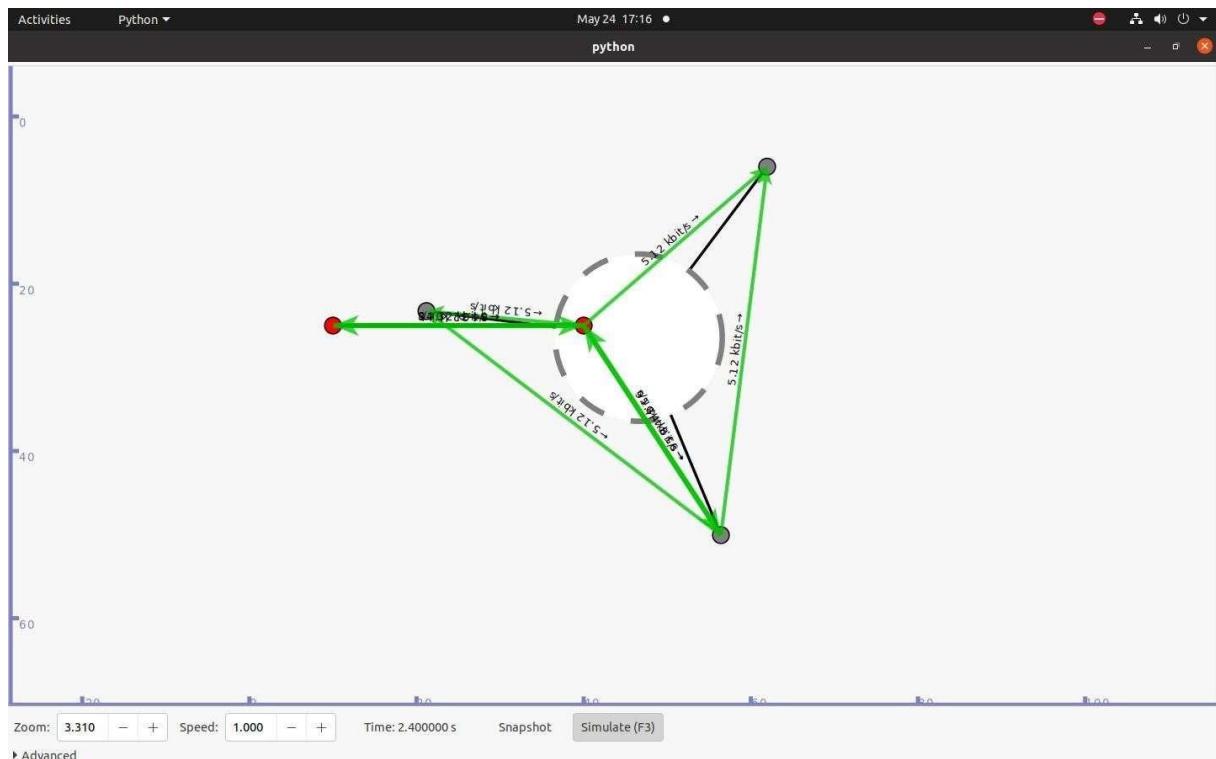
```
MobilityHelper mobility;
```

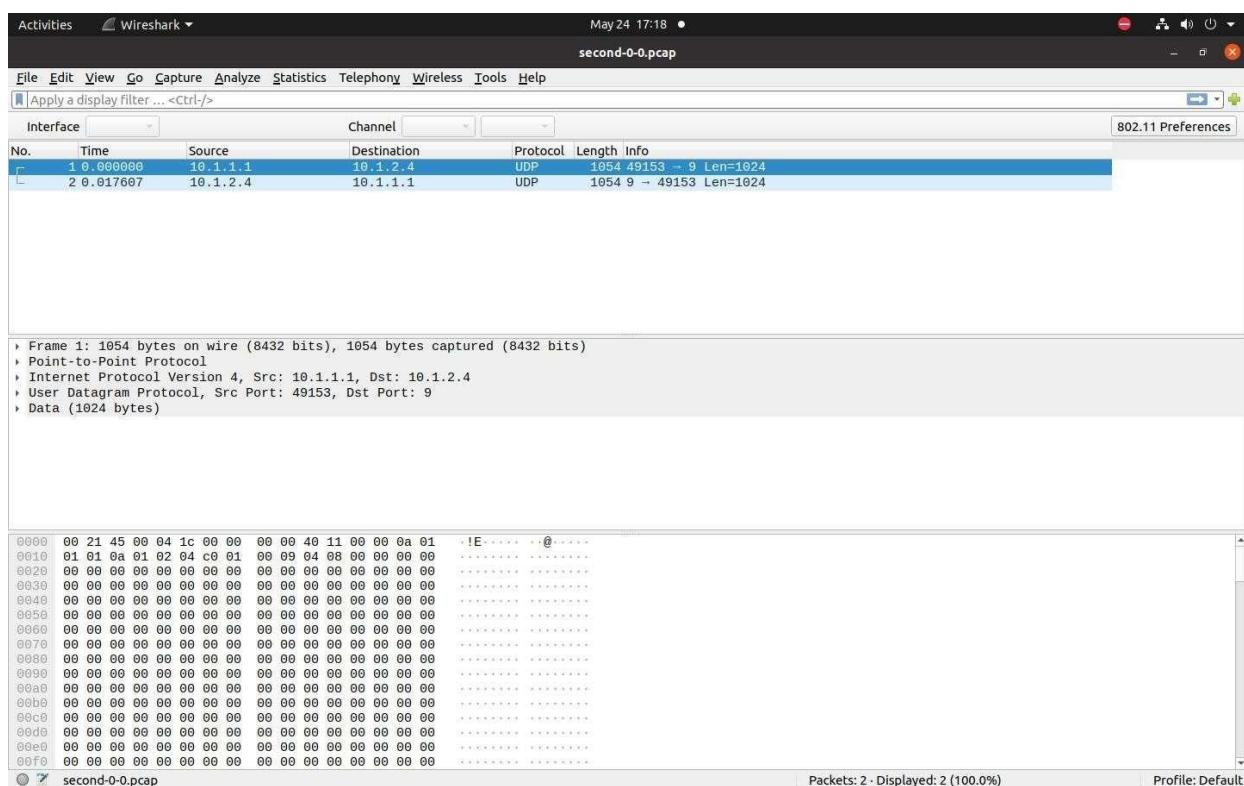
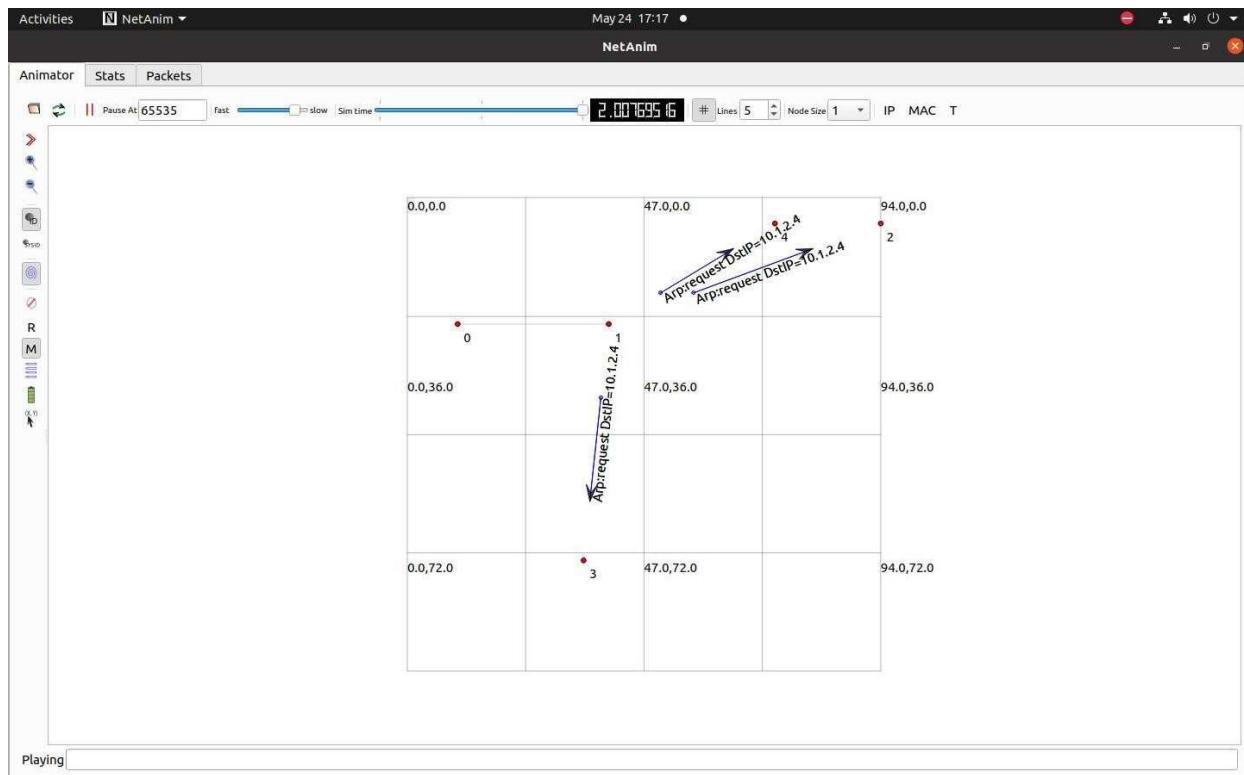
```
mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
```

```
mobility.Install(nodes);

AnimationInterface anim("second.xml");
AnimationInterface::SetConstantPosition(nodes.Get(0),10,25);
AnimationInterface::SetConstantPosition(nodes.Get(1),40,25);
anim.EnablePacketMetadata(true);
pointToPoint.EnablePcapAll ("second");
csma.EnablePcap ("second", csmaDevices.Get (1), true);
Simulator::Run ();
Simulator::Destroy ();
return 0;
}
```

OutPut

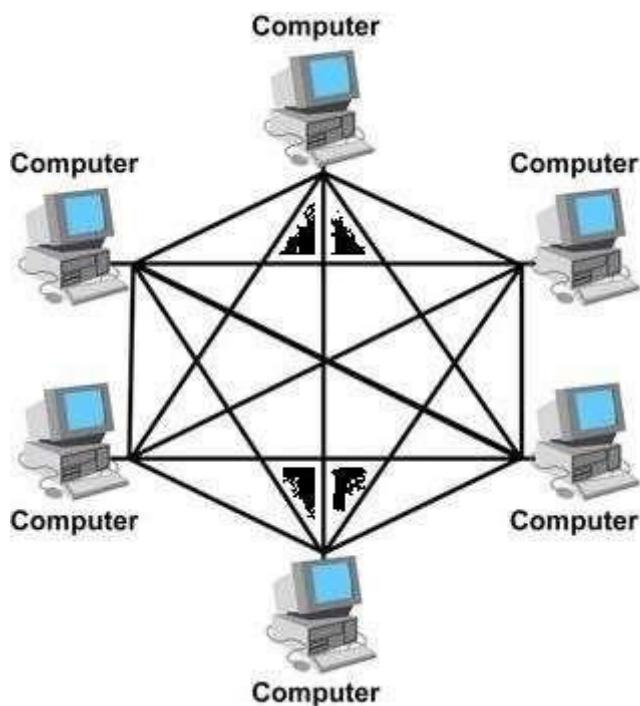




Practical 7:Program to simulate mesh topology

Mesh topology is a type of networking in which all the computers are inter-connected to each other. In Mesh Topology, the connections between devices take place randomly. The connected nodes can be computers, switches, hubs, or any other devices. In this topology setup, even if one of the connections goes down, it allows other nodes to be distributed. This type of topology is very expensive and does not have any hierarchy, interdependency, and uniform pattern between nodes. The connections of the mesh topology are not easier to establish.

All computers, in a mesh topology, not only responsible for sending their own signals but also relays for other nodes. Usually, it is used for wireless networks, and its connections can be wired or wireless. Also, there is a point-to-point connection between all nodes in the mesh topology setup. The below picture is an instance of a mesh topology network.



Code:

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2008,2009 IITP RAS
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
```

* published by the Free Software Foundation;

*

* This program is distributed in the hope that it will be useful,

* but WITHOUT ANY WARRANTY; without even the implied warranty of

* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

* GNU General Public License for more details.

*

* You should have received a copy of the GNU General Public License

* along with this program; if not, write to the Free Software

* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

*

* Author: Kirill Andreev <andreev@iitp.ru>

*

*

* By default this script creates m_xSize * m_ySize square grid topology with

* IEEE802.11s stack installed at each node with peering management

* and HWMP protocol.

* The side of the square cell is defined by m_step parameter.

* When topology is created, UDP ping is installed to opposite corners

* by diagonals. packet size of the UDP ping and interval between two

* successive packets is configurable.

*

* m_xSize * step

* |<____>|

* step

* |<-->|

* * --- * --- * <---Ping sink _

```
* | \ | /|      ^
* | \ | / |      |
* * --- * --- * m_ySize * step |
* | /| \ |      |
* | / | \ |      |
* * --- * --- *
* ^ Ping source
*
* See also MeshTest::Configure to read more about configurable
* parameters.
*/
```

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

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("TestMeshScript");
```

```
/***
 * \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 (100.0),  
    m_randomStart (0.1),  
    m_totalTime (100.0),  
    m_packetInterval (0.1),  
    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);
}

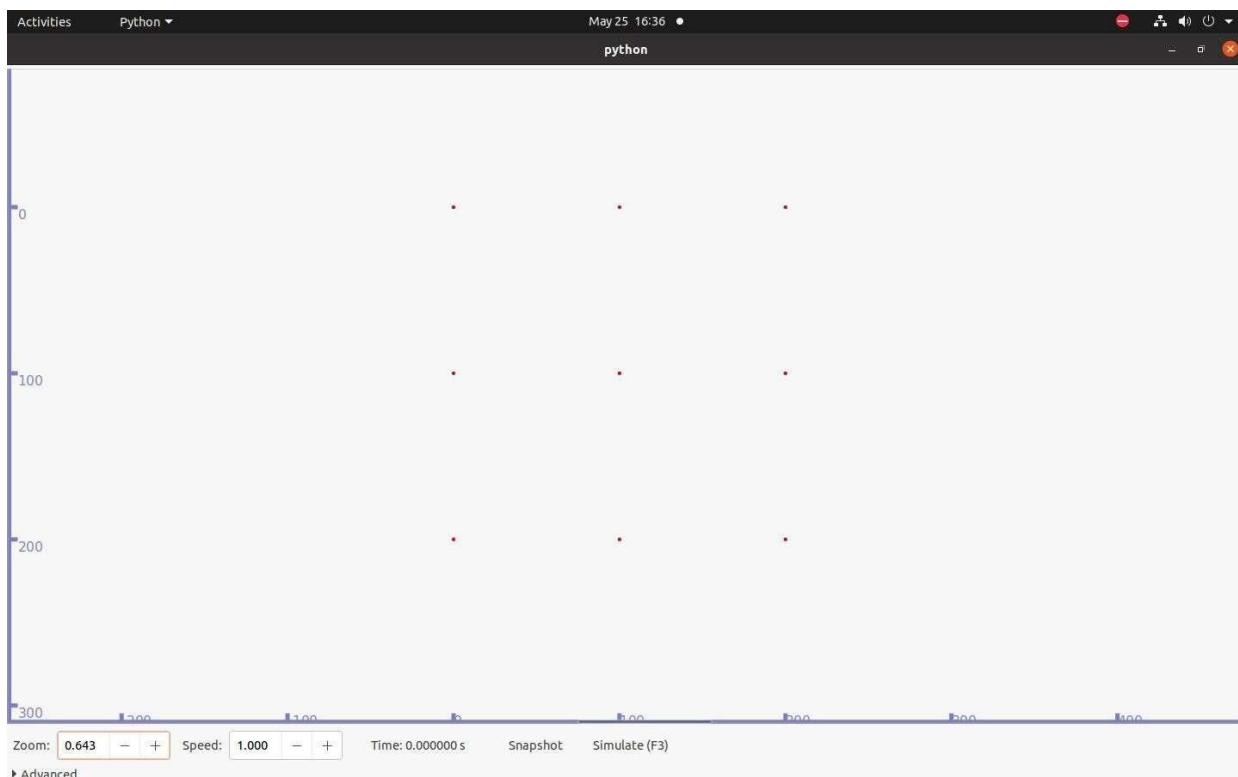
mesh.SetMacType ("RandomStart", TimeValue (Seconds (m_randomStart)));
// Set number of interfaces - default is single-interface mesh point
mesh.SetNumberOfInterfaces (m_nIfaces);
// Install protocols and return container if MeshPointDevices
meshDevices = mesh.Install (wifiPhy, nodes);
// Setup mobility - 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 ()  
{  
    UdpEchoServerHelper echoServer (9);  
    ApplicationContainer serverApps = echoServer.Install (nodes.Get (0));  
    serverApps.Start (Seconds (0.0));  
    serverApps.Stop (Seconds (m_totalTime));  
    UdpEchoClientHelper echoClient (interfaces.GetAddress (0), 9);  
    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 (m_xSize*m_ySize-  
1));  
    clientApps.Start (Seconds (0.0));  
    clientApps.Stop (Seconds (m_totalTime));  
}  
int  
MeshTest::Run ()  
{
```

```
CreateNodes ();  
InstallInternetStack ();  
InstallApplication ();  
Simulator::Schedule (Seconds (m_totalTime), &MeshTest::Report, this);  
Simulator::Stop (Seconds (m_totalTime));  
Simulator::Run ();  
Simulator::Destroy ();  
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);
```

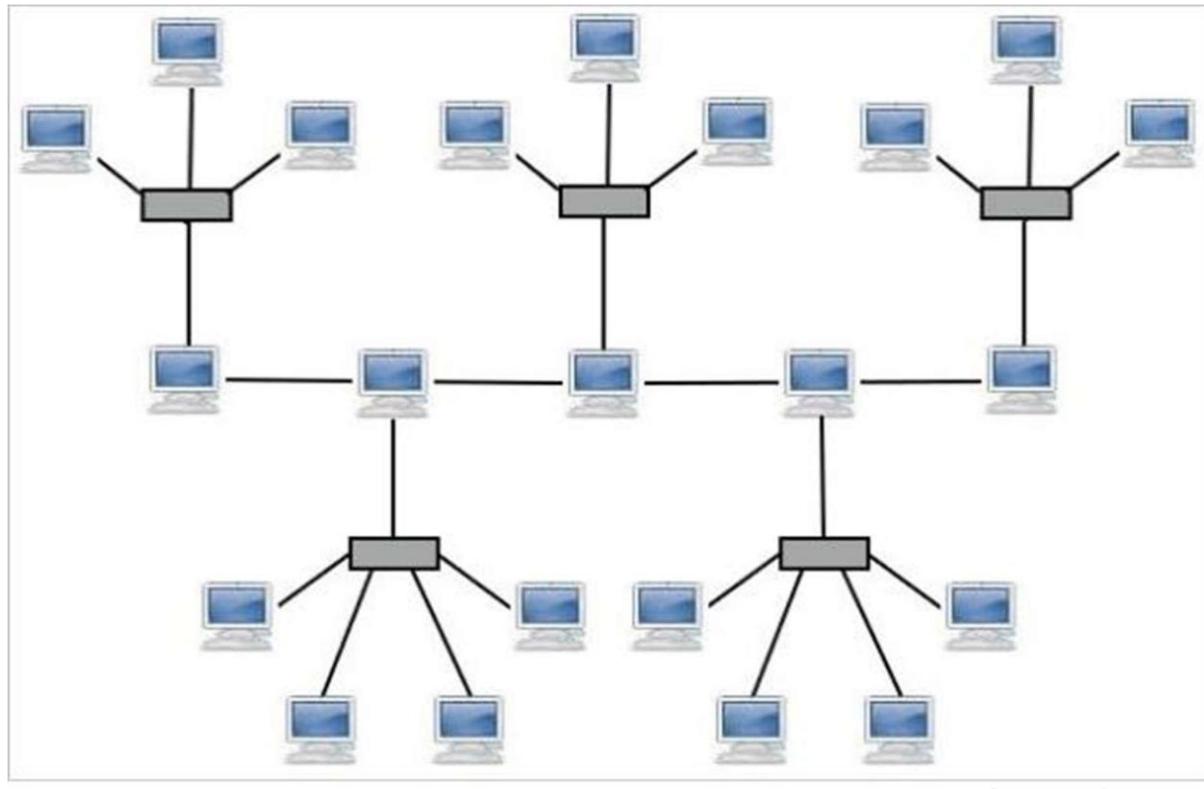
```
        of.close ();  
    }  
}  
int  
main (int argc, char *argv[])  
{  
    MeshTest t;  
    t.Configure (argc, argv);  
    return t.Run ();  
}
```

OutPut:



Practical 8: Program to simulate hybrid topology

A hybrid topology is a kind of network topology that is a combination of two or more network topologies, such as mesh topology, bus topology, and ring topology. Its usage and choice are dependent on its deployments and requirements like the performance of the desired network, and the number of computers, their location.



Code:

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

```

```
* GNU General Public License for more details.  
*  
* You should have received a copy of the GNU General Public License  
* along with this program; if not, write to the Free Software  
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA  
*/
```

```
#include <fstream>  
  
#include "ns3/core-module.h"  
  
#include "ns3/network-module.h"  
  
#include "ns3/internet-module.h"  
  
#include "ns3/point-to-point-module.h"  
  
#include "ns3/applications-module.h"  
  
//netanimation  
  
#include "ns3/netanim-module.h"  
  
#include "ns3/mobility-module.h"  
  
  
using namespace ns3;  
  
  
NS_LOG_COMPONENT_DEFINE ("FifthScriptExample");  
  
  
//  
=====
```

```
//      node 0          node 1  
// +_____+ +_____+  
// | ns-3 TCP | | ns-3 TCP |
```

```
// +_____+ +_____+
// | 10.1.1.1 | | 10.1.1.2 |
// +_____+ +_____+
// | point-to-point | | point-to-point |
// +.....+ +.....+
// |           |
// +.....+
//      5 Mbps, 2 ms
```

```
// We want to look at changes in the ns-3 TCP congestion window. We need
// to crank up a flow and hook the CongestionWindow attribute on the socket
// of the sender. Normally one would use an on-off application to generate a
// flow, but this has a couple of problems. First, the socket of the on-off
// application is not created until Application Start time, so we wouldn't be
// able to hook the socket (now) at configuration time. Second, even if we
// could arrange a call after start time, the socket is not public so we
// couldn't get at it.
```

```
// So, we can cook up a simple version of the on-off application that does what
// we want. On the plus side we don't need all of the complexity of the on-off
// application. On the minus side, we don't have a helper, so we have to get
// a little more involved in the details, but this is trivial.
```

```
// So first, we create a socket and do the trace connect on it; then we pass
// this socket into the constructor of our simple application which we then
// install in the source node.
```

```
//  
=====  
=====  
//  
  
class MyApp : public Application  
{  
public:  
  
    MyApp ();  
    virtual ~MyApp();  
  
    void Setup (Ptr<Socket> socket, Address address, uint32_t packetSize, uint32_t  
nPackets, DataRate dataRate);  
  
private:  
    virtual void StartApplication (void);  
    virtual void StopApplication (void);  
  
    void ScheduleTx (void);  
    void SendPacket (void);  
  
    Ptr<Socket>    m_socket;  
    Address        m_peer;  
    uint32_t       m_packetSize;  
    uint32_t       m_nPackets;  
    DataRate       m_dataRate;  
    EventId        m_sendEvent;  
    bool           m_running;
```

```
    uint32_t      m_packetsSent;  
};
```

```
MyApp::MyApp()
```

```
    : m_socket(0),  
      m_peer(),  
      m_packetSize(0),  
      m_nPackets(0),  
      m_dataRate(0),  
      m_sendEvent(),  
      m_running(false),  
      m_packetsSent(0)  
{  
}
```

```
MyApp::~MyApp()
```

```
{  
    m_socket = 0;  
}
```

```
void
```

```
MyApp::Setup(Ptr<Socket> socket, Address address, uint32_t packetSize, uint32_t  
nPackets, DataRate dataRate)
```

```
{  
    m_socket = socket;  
    m_peer = address;  
    m_packetSize = packetSize;
```

```
m_nPackets = nPackets;  
m_dataRate = dataRate;  
}  
  
void  
MyApp::StartApplication (void)  
{  
    m_running = true;  
    m_packetsSent = 0;  
    m_socket->Bind ();  
    m_socket->Connect (m_peer);  
    SendPacket ();  
}  
  
void  
MyApp::StopApplication (void)  
{  
    m_running = false;  
  
    if (m_sendEvent.IsRunning ())  
    {  
        Simulator::Cancel (m_sendEvent);  
    }  
  
    if (m_socket)  
    {  
        m_socket->Close ();  
    }
```

```
    }

}

void
MyApp::SendPacket (void)
{
    Ptr<Packet> packet = Create<Packet> (m_packetSize);
    m_socket->Send (packet);

    if (++m_packetsSent < m_nPackets)
    {
        ScheduleTx ();
    }
}

void
MyApp::ScheduleTx (void)
{
    if (m_running)
    {
        Time tNext (Seconds (m_packetSize * 8 / static_cast<double>
(m_dataRate.GetBitRate ())));
        m_sendEvent = Simulator::Schedule (tNext, &MyApp::SendPacket, this);
    }
}

static void
```

```
CwndChange (uint32_t oldCwnd, uint32_t newCwnd)
{
    NS_LOG_UNCOND (Simulator::Now ().GetSeconds () << "\t" << newCwnd);
}

static void
RxDrop (Ptr<const Packet> p)
{
    NS_LOG_UNCOND ("RxDrop at " << Simulator::Now ().GetSeconds ());
}

int
main (int argc, char *argv[])
{
    CommandLine cmd (__FILE__);
    cmd.Parse (argc, argv);

    NodeContainer nodes;
    nodes.Create (2);

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

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

```
Ptr<RateErrorModel> em = CreateObject<RateErrorModel>();  
em->SetAttribute ("ErrorRate", DoubleValue (0.00001));  
devices.Get (1)->SetAttribute ("ReceiveErrorModel", PointerValue (em));
```

```
InternetStackHelper stack;  
stack.Install (nodes);
```

```
Ipv4AddressHelper address;  
address.SetBase ("10.1.1.0", "255.255.255.252");  
Ipv4InterfaceContainer interfaces = address.Assign (devices);
```

```
uint16_t sinkPort = 8080;  
Address sinkAddress (InetSocketAddress (interfaces.GetAddress (1), sinkPort));  
PacketSinkHelper packetSinkHelper ("ns3::TcpSocketFactory", InetSocketAddress  
(Ipv4Address::GetAny (), sinkPort));  
ApplicationContainer sinkApps = packetSinkHelper.Install (nodes.Get (1));  
sinkApps.Start (Seconds (0.));  
sinkApps.Stop (Seconds (20.));
```

```
Ptr<Socket> ns3TcpSocket = Socket::CreateSocket (nodes.Get (0),  
TcpSocketFactory::GetTypeId ());  
ns3TcpSocket->TraceConnectWithoutContext ("CongestionWindow", MakeCallback  
(&CwndChange));
```

```
Ptr<MyApp> app = CreateObject<MyApp>();  
app->Setup (ns3TcpSocket, sinkAddress, 1040, 1000, DataRate ("1Mbps"));  
nodes.Get (0)->AddApplication (app);  
app->SetStartTime (Seconds (1.));
```

```
app->SetStopTime (Seconds (20.));
```

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

```
MobilityHelper mobility;
```

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

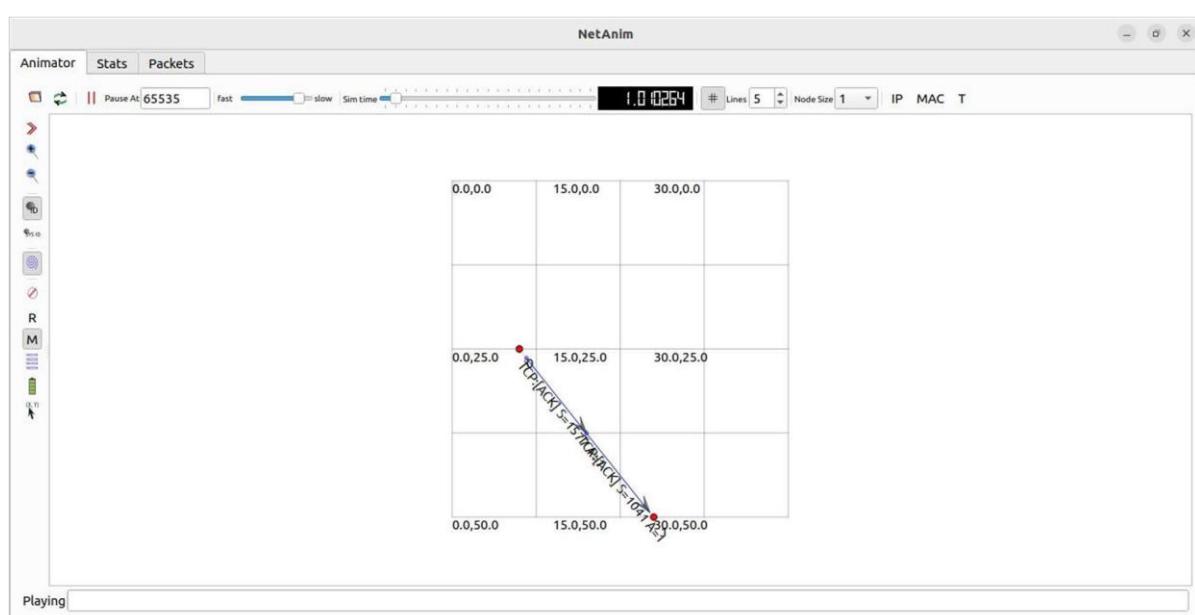
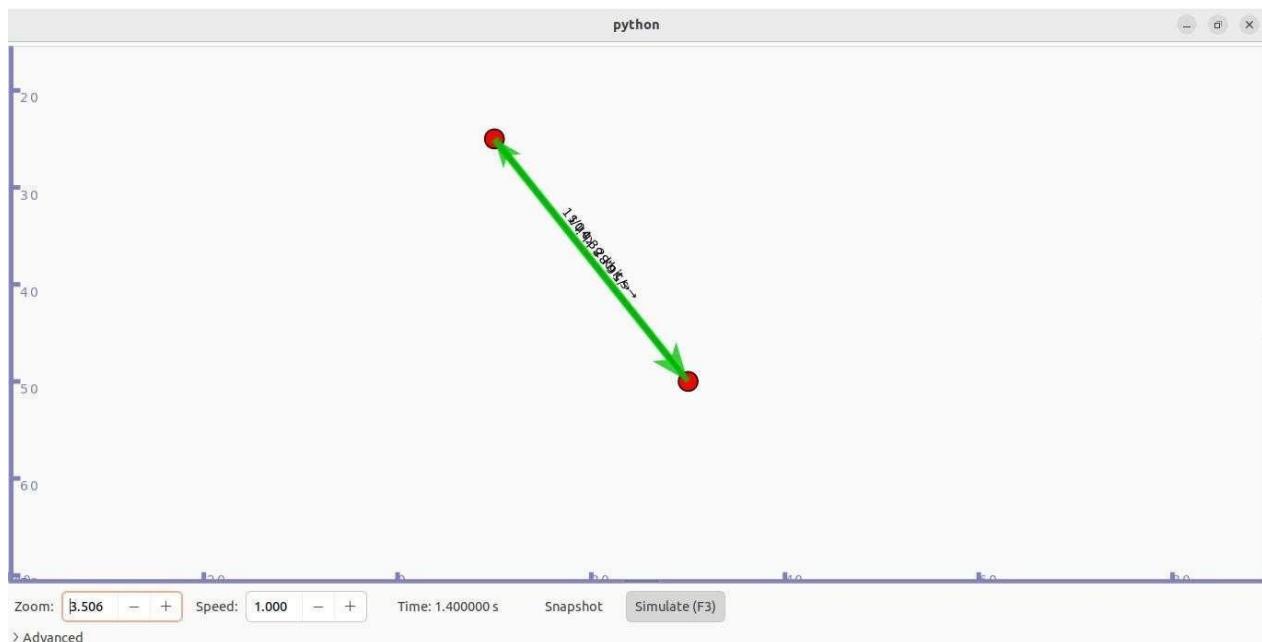
```
AnimationInterface anim("fifth.xml");  
AnimationInterface::SetConstantPosition(nodes.Get(0),10,25);  
AnimationInterface::SetConstantPosition(nodes.Get(1),30,50);  
anim.EnablePacketMetadata(true);
```

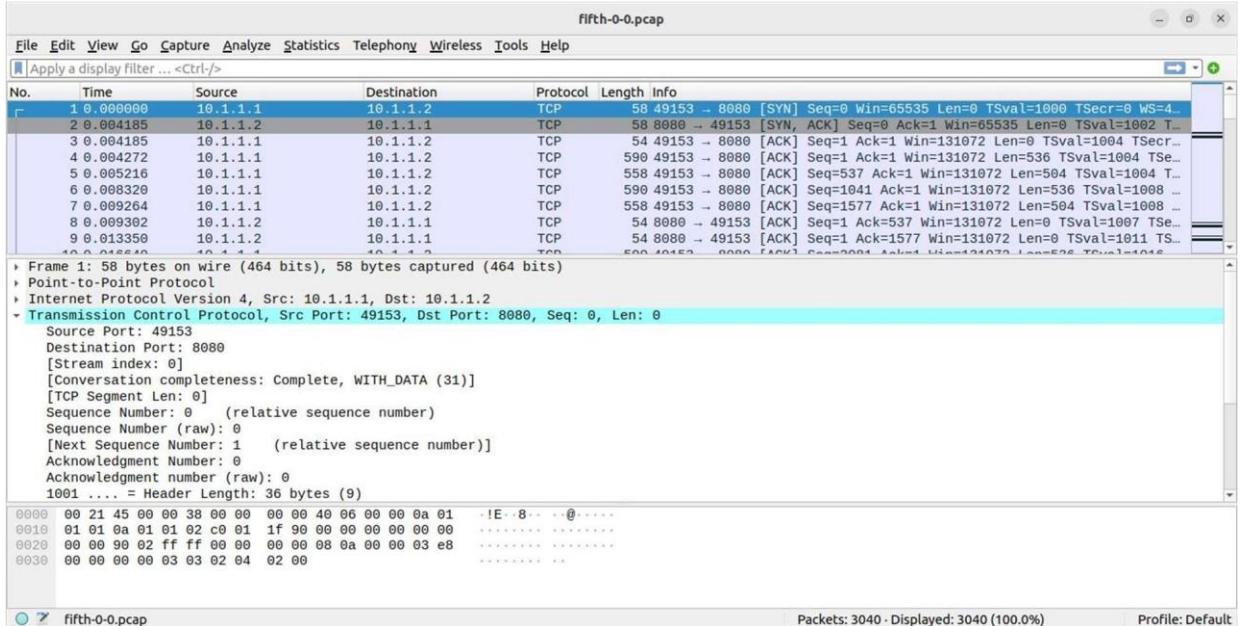
```
pointToPoint.EnablePcapAll("fifth");  
Simulator::Stop (Seconds (20));  
Simulator::Run ();  
Simulator::Destroy ();
```

```
return 0;
```

```
}
```

OutPut:





Practical 9: Program to simulate UDP server

User Datagram Protocol (UDP) is a communications protocol that is primarily used to establish low-latency and loss-tolerating connections between applications on the internet.

UDP speeds up transmissions by enabling the transfer of data before an agreement is provided by the receiving party. As a result, UDP is beneficial in time-sensitive communications, including voice over IP (VoIP), domain name system (DNS) lookup, and video or audio playback.

Code:

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

```

```
// 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[])  
{  
//  
// Users may find it convenient to turn on explicit debugging  
// for selected modules; the below lines suggest how to do this  
//  
#if 0  
    LogComponentEnable ("UdpEchoExample", LOG_LEVEL_INFO);  
    LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_ALL);  
    LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_ALL);
```

```
#endif

// Allow the user to override any of the defaults and the above Bind() at
// run-time, via command-line arguments
//

bool useV6 = false;
Address serverAddress;

CommandLine cmd (__FILE__);
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 (4);

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 a UdpEchoServer application on node one.  
//
```

```
uint16_t port = 9; // well-known echo port number
UdpEchoServerHelper server (port);
ApplicationContainer apps = server.Install (n.Get (1));
apps.Start (Seconds (1.0));
apps.Stop (Seconds (10.0));

//
// Create a UdpEchoClient application to send UDP datagrams from node zero to
// node one.
//
uint32_t packetSize = 1024;
uint32_t maxPacketCount = 1;
Time interPacketInterval = Seconds (1.);
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));

#if 0
//
// Users may find it convenient to initialize echo packets with actual data;
// the below lines suggest how to do this
//
client.SetFill (apps.Get (0), "Hello World");

```

```
client.SetFill (apps.Get (0), 0xa5, 1024);

uint8_t fill[] = { 0, 1, 2, 3, 4, 5, 6};

client.SetFill (apps.Get (0), fill, sizeof(fill), 1024);

#endif

AsciiTraceHelper ascii;

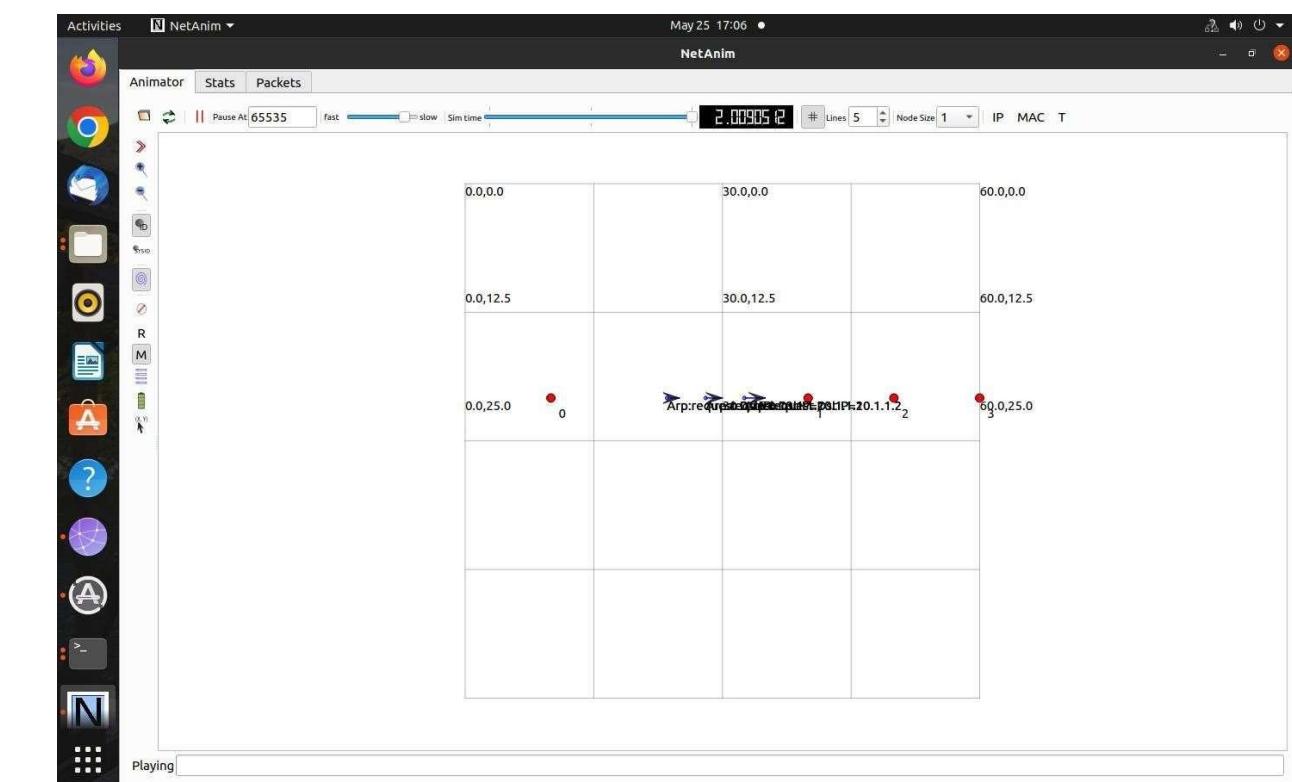
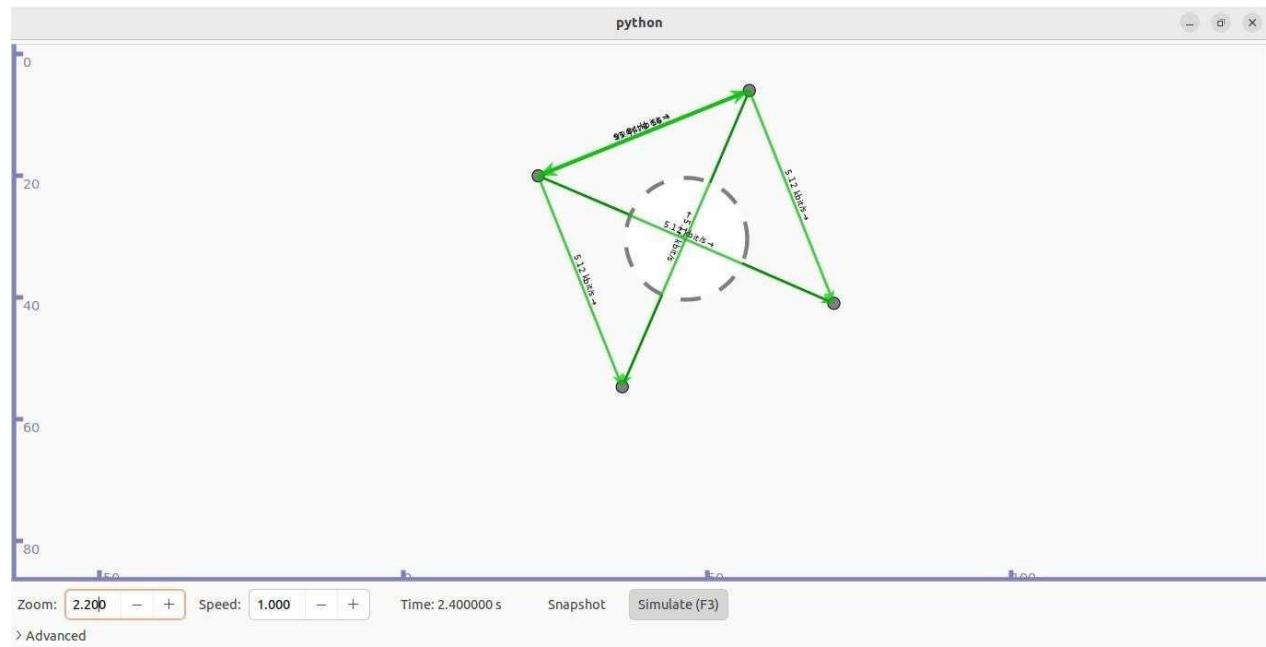
csma.EnableAsciiAll (ascii.CreateFileStream ("udp-echo.tr"));

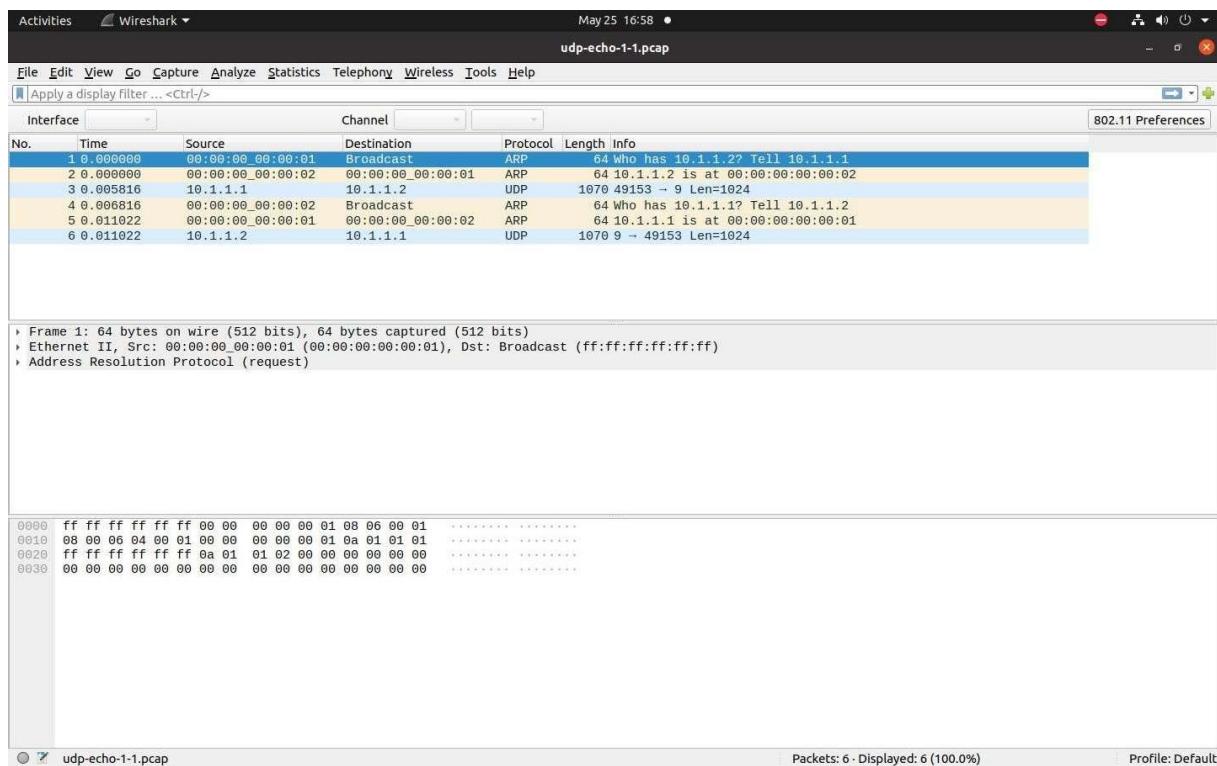
csma.EnablePcapAll ("udp-echo", false);

// Now, do the actual simulation.

// NS_LOG_INFO ("Run Simulation.");
Simulator::Run ();
Simulator::Destroy ();
NS_LOG_INFO ("Done.");
}
```

OutPut:





Practical 10:Program to simulate UDP server client

Code:

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2009 INRIA
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * Author: Mohamed Amine Ismail <amine.ismail@sophia.inria.fr>
 */

// Network topology
//
//   n0   n1
//   |   |

```

```
// =====  
// LAN (CSMA)  
  
// - UDP flow from n0 to n1 of 1024 byte packets at intervals of 50 ms  
// - maximum of 320 packets sent (or limited by simulation duration)  
// - option to use IPv4 or IPv6 addressing  
// - option to disable logging statements
```

```
#include <fstream>  
  
#include "ns3/core-module.h"  
#include "ns3/csma-module.h"  
#include "ns3/applications-module.h"  
#include "ns3/internet-module.h"  
#include "ns3/point-to-point-module.h"  
  
//netanimation  
#include "ns3/netanim-module.h"  
#include "ns3/mobility-module.h"  
  
using namespace ns3;  
  
NS_LOG_COMPONENT_DEFINE ("UdpClientServerExample");
```

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

```
// Declare variables used in command-line arguments
bool useV6 = false;
bool logging = true;
Address serverAddress;

CommandLine cmd (__FILE__);
cmd.AddValue ("useIpv6", "Use Ipv6", useV6);
cmd.AddValue ("logging", "Enable logging", logging);
cmd.Parse (argc, argv);

if (logging)
{
    LogComponentEnable ("UdpClient", LOG_LEVEL_INFO);
    LogComponentEnable ("UdpServer", LOG_LEVEL_INFO);
}

NS_LOG_INFO ("Create nodes in above topology.");
NodeContainer n;
n.Create (2);

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

NetDeviceContainer devices;
devices = pointToPoint.Install (n);
```

```
InternetStackHelper internet;
internet.Install (n);

NS_LOG_INFO ("Create channel between the two nodes.");
CsmaHelper csma;
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 ("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 UdpServer application on node 1.");
```

```
uint16_t port = 4000;
```

```
UdpServerHelper server (port);
```

```
ApplicationContainer apps = server.Install (n.Get (1));
```

```
apps.Start (Seconds (1.0));
```

```
apps.Stop (Seconds (10.0));
```

```
NS_LOG_INFO ("Create UdpClient application on node 0 to send to node 1.");
```

```
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));
```

```
NS_LOG_INFO ("Run Simulation.");
```

```
MobilityHelper mobility;
```

```
mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
```

```
mobility.Install(n);
```

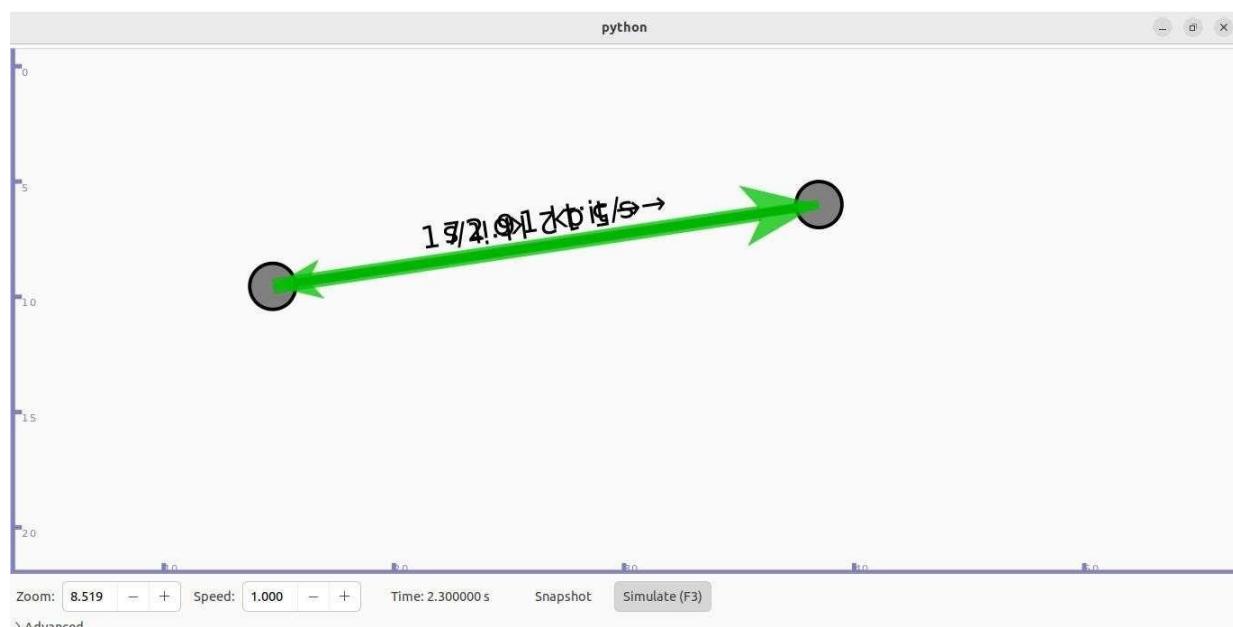
```
AnimationInterface anim("udp-client-server.xml");
```

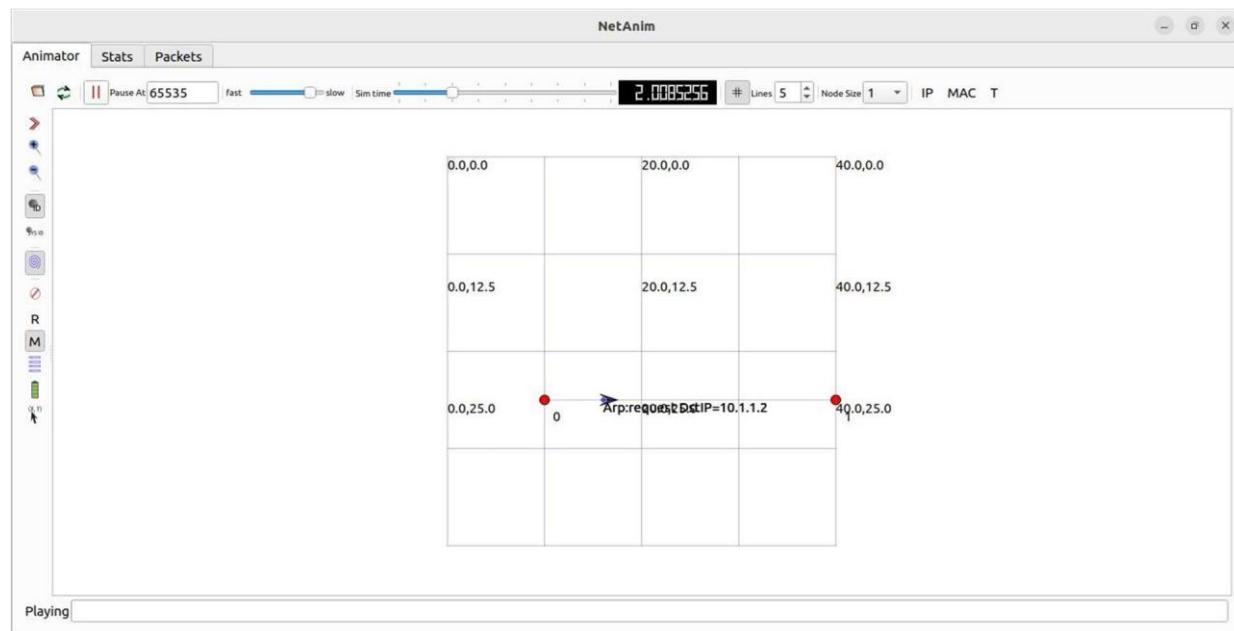
```
AnimationInterface::SetConstantPosition(n.Get(0),10,25);  
AnimationInterface::SetConstantPosition(n.Get(1),40,25);  
anim.EnablePacketMetadata(true);
```

```
pointToPoint.EnablePcapAll("udp-client-server");
```

```
Simulator::Run ();  
Simulator::Destroy ();  
NS_LOG_INFO ("Done.");  
}
```

Output:





Practical 11: Program to simulate DHCP server and n clients

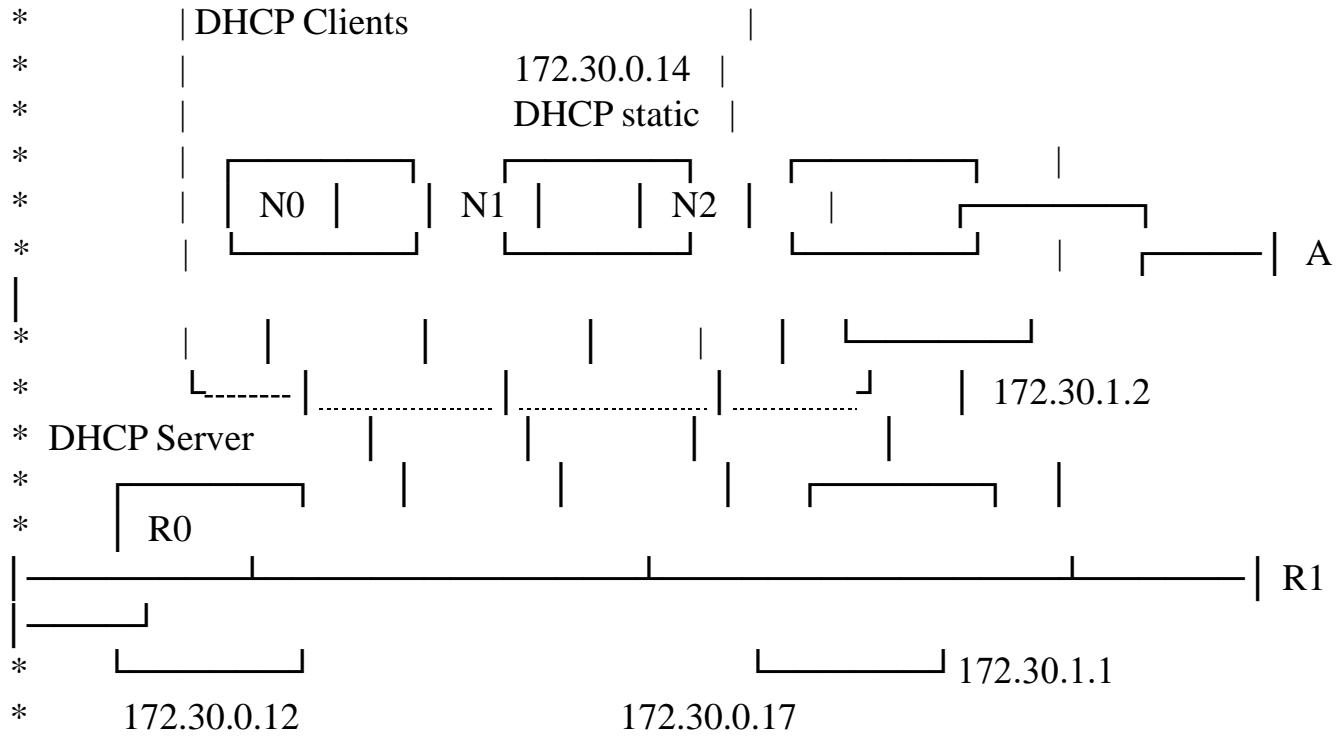
DHCP stands for Dynamic Host Configuration Protocol. It is the critical feature on which the users of an enterprise network communicate. DHCP helps enterprises to smoothly manage the allocation of IP addresses to the end-user clients' devices such as desktops, laptops, cellphones.

Code:

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2011 UPB
 * Copyright (c) 2017 NITK Surathkal
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * Author: Radu Lupu <rлуpu@elcom.pub.ro>
 *        Ankit Deepak <adadeepak8@gmail.com>
 *        Deepti Rajagopal <deeptir96@gmail.com>
 *
 */
/*
 * Network layout:
 *
 * R0 is a DHCP server. The DHCP server announced R1 as the default router.
 * Nodes N1 will send UDP Echo packets to node A.
 *
 *
 */

```





- * Things to notice:

- * 1) The routes in A are manually set to have R1 as the default router,
* just because using a dynamic outing in this example is an overkill.
- * 2) R1's address is set statically though the DHCP server helper interface.
* This is useful to prevent address conflicts with the dynamic pool.
* Not necessary if the DHCP pool is not conflicting with static addresses.
- * 3) N2 has a dynamically-assigned, static address (i.e., a fixed address assigned via
DHCP).

- *

- */

```
#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"

//netanim code
#include "ns3/netanim-module.h"
#include "ns3/mobility-module.h"
```

```
using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("DhcpExample");

int
main (int argc, char *argv[])
{
    CommandLine cmd (__FILE__);

    bool verbose = false;
    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);
```

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

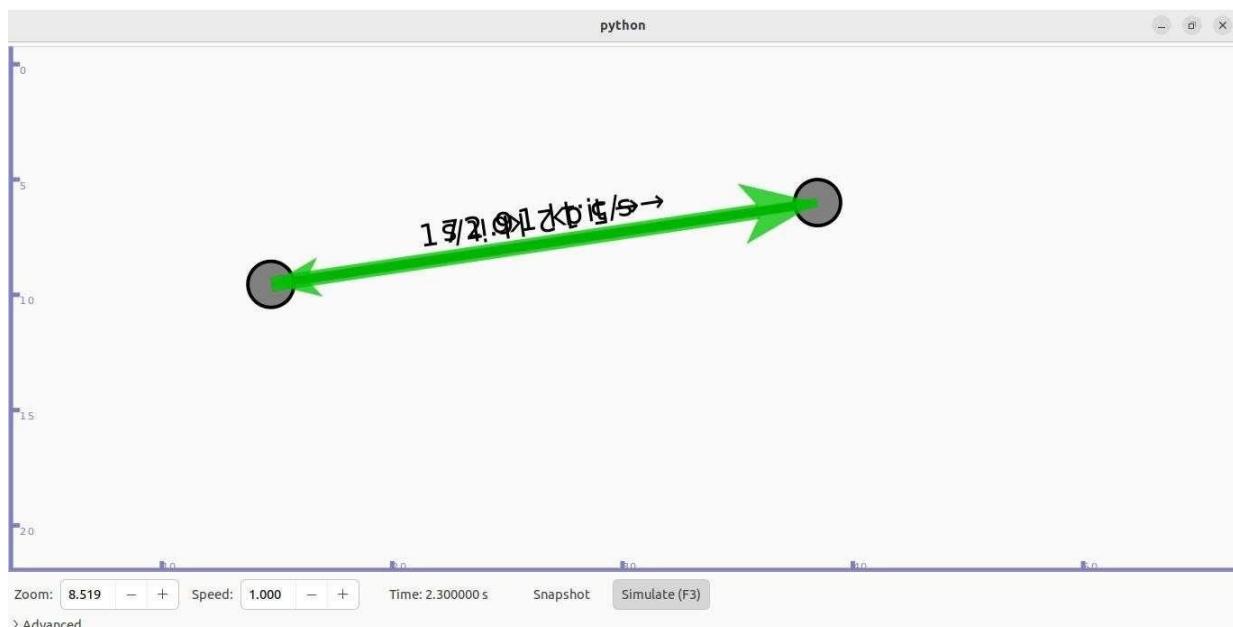
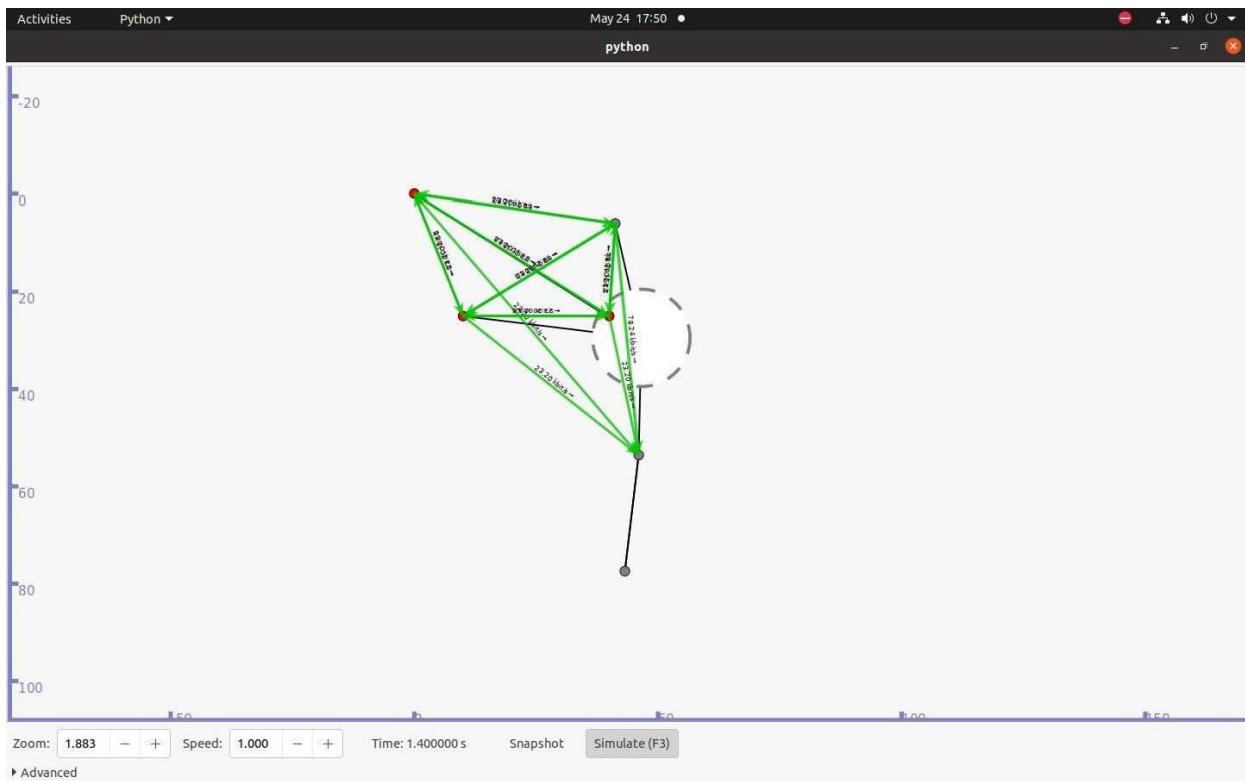
```
AnimationInterface anim("pranay.xml");
AnimationInterface::SetConstantPosition(nodes.Get(0),10,25);
AnimationInterface::SetConstantPosition(nodes.Get(1),40,25);
anim.EnablePacketMetadata(true);
```

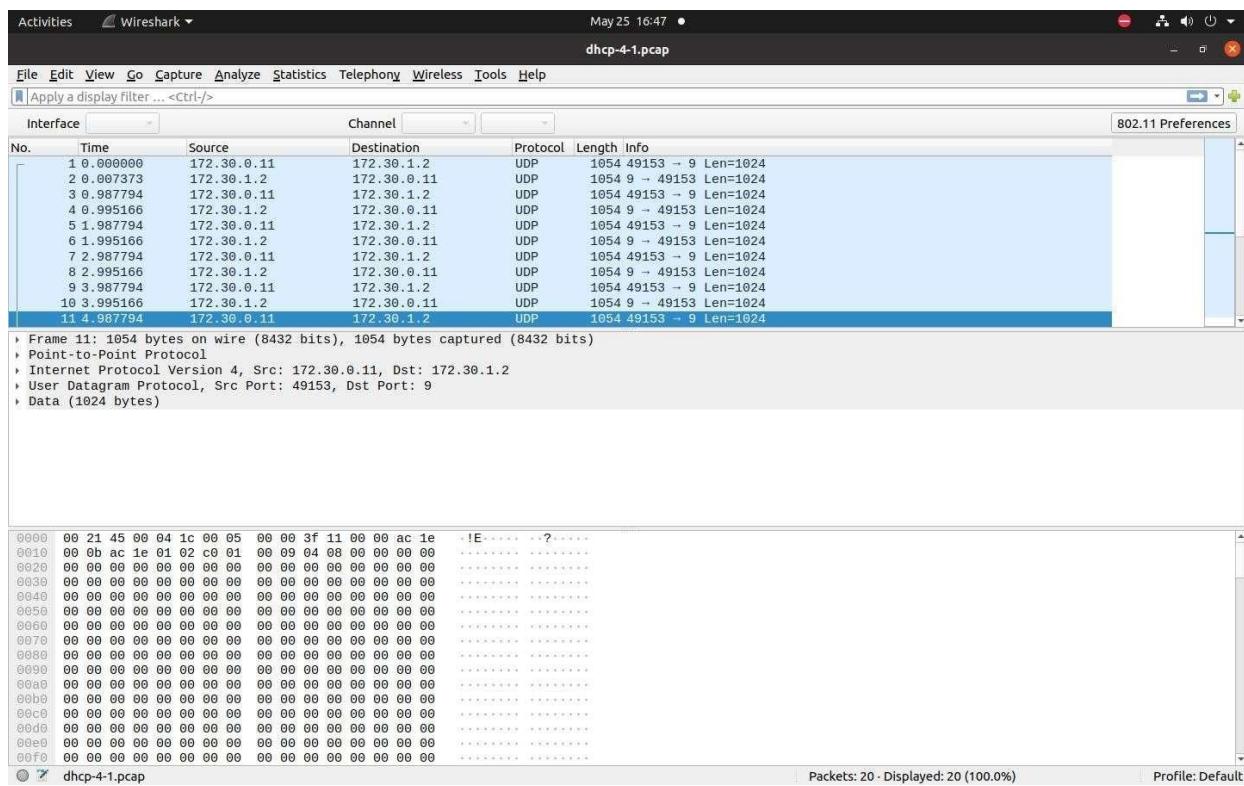
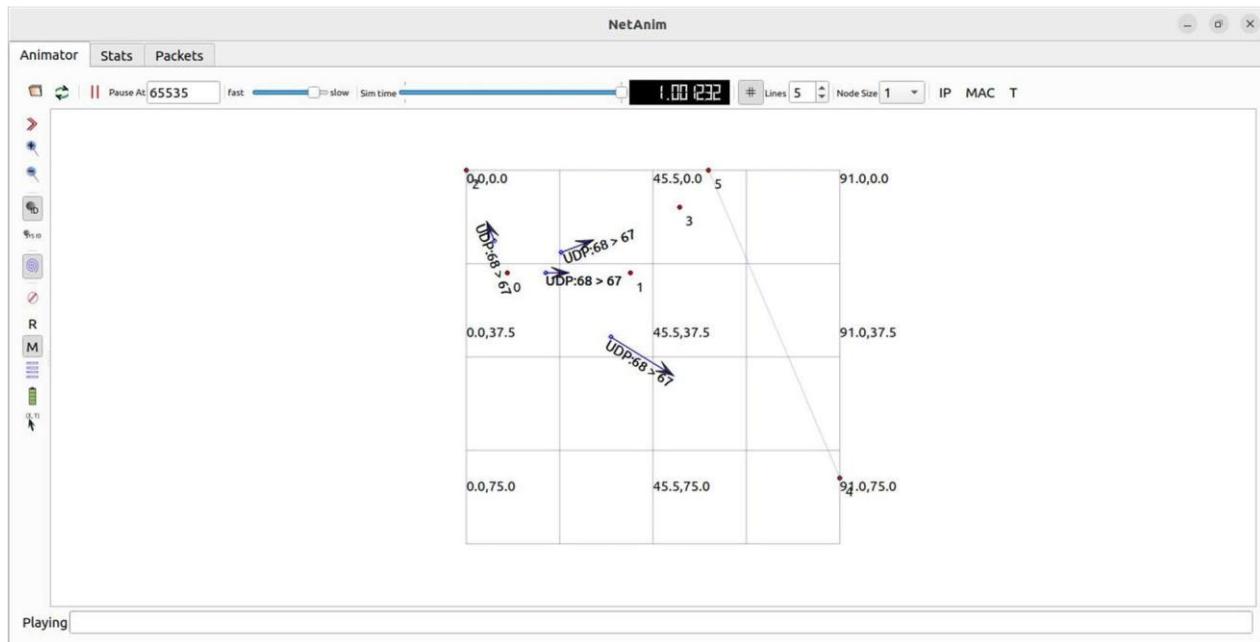
```
Simulator::Stop (stopTime + Seconds (10.0));
```

```
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 12: Animate a simple Network using NetAnim in Network Stimulator

Step 1: Generate the XML file using Mobility helper in first.cc and the code below to create the XML file

```
79  
80 MobilityHelper mobility;  
81 mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");  
82 mobility.Install(nodes);  
83  
84 AnimationInterface anim("first.xml");  
85 AnimationInterface::SetConstantPosition(nodes.Get(0),10,25);  
86 AnimationInterface::SetConstantPosition(nodes.Get(1),40,25);  
87 anim.EnablePacketMetadata(true);  
88  
89 pointToPoint.EnablePcapAll("first");  
90  
91 Simulator::Run ();  
92 Simulator::Destroy ();  
93 return 0;  
94 }
```

C++ ▾ Tab Width: 8 ▾ Ln 89, Col 39 ▾ INS

Step 2: Then change directory to NetAnim folder and open terminal and enter ./NetAnim to start NetAnim



Step 3: Open the XML File that was just generated now and click the play button to start the stimulation

Activities NetAnim May 25 22:43

Animator Cancel Searching in ns-3.35 Open

Name Location Size Type Modified

- Recent
- Home
- Documents
- Downloads
- Music
- Pictures
- Videos
- netanim-3.108
- + Other Locations

xml

Name	Location	Size	Type	Modified
dhcp.xml		23.0 kB	Markup	16 May
fifth.xml		1.3 MB	Markup	21:43
First.xml		1.9 kB	Markup	Fri
second.xml		14.3 kB	Markup	Fri
sixth.xml		343.2 kB	Markup	Fri
third.xml		111.8 kB	Markup	Fri
udp-client-server.xml		118.9 kB	Markup	01:24

NetAnim

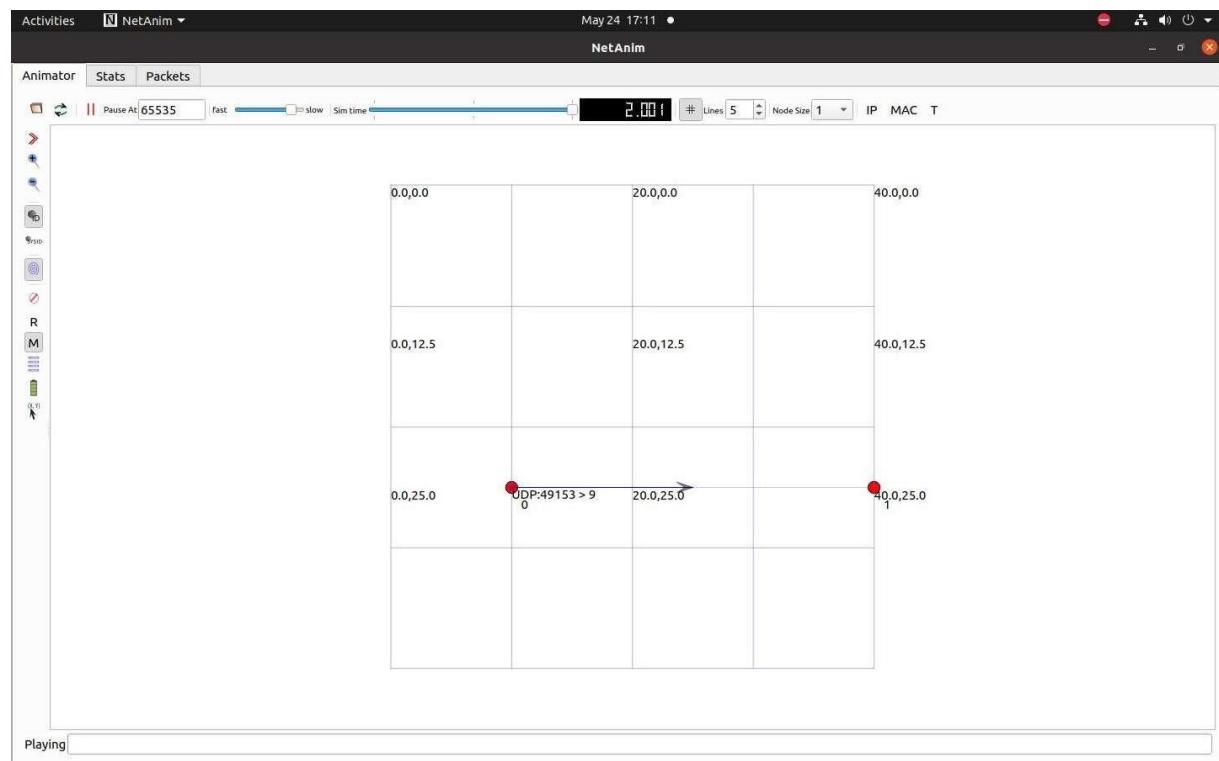
Animator Stats Packets

Pause At: 65535 | Fast | slow | Sim time | Lines: 5 | Node Size: 1 | IP MAC T

0.0.0 20.0.0 40.0.0
0.0.12.5 20.0.12.5 40.0.12.5
0.0.25.0 0 20.0.25.0 40.0.25.0

Parsing complete: Click Play

The screenshot shows the NetAnim application interface. On the left is a vertical toolbar with icons for various tools like Animator, Stats, and Packets. The main window has three tabs: Animator (selected), Stats, and Packets. The Animator tab displays a network diagram with four nodes labeled 0.0.0.0, 20.0.0.0, 40.0.0.0, 0.0.12.5, 20.0.12.5, 40.0.12.5, 0.0.25.0, and 20.0.25.0. Nodes 0.0.0.0, 20.0.0.0, and 40.0.0.0 are at the top in a row, while others are below them. Nodes 0.0.12.5, 20.0.12.5, and 40.0.12.5 are in the middle row, and 0.0.25.0 and 20.0.25.0 are at the bottom. Red dots are placed on nodes 0.0.0.0, 20.0.0.0, and 40.0.0.0. Below the diagram is a status bar with the message "Parsing complete: Click Play". To the right of the main window is a file browser titled "NetAnim" showing a list of XML files in the ns-3.35 directory. The file "First.xml" is selected and highlighted in orange.



Practical 13: Network Traffic Analysis using Wireshark

Wireshark is a network protocol analyzer, or an application that captures packets from a network connection, such as from your computer to your home office or the internet. Packet is the name given to a discrete unit of data in a typical Ethernet network.

Wireshark is the most often-used packet sniffer in the world. Like any other packet sniffer, Wireshark does three things:

1. **Packet Capture:** Wireshark listens to a network connection in real time and then grabs entire streams of traffic – quite possibly tens of thousands of packets at a time.
2. **Filtering:** Wireshark is capable of slicing and dicing all of this random live data using filters. By applying a filter, you can obtain just the information you need to see.
3. **Visualization:** Wireshark, like any good packet sniffer, allows you to dive right into the very middle of a network packet. It also allows you to visualize entire conversations and network streams.

How to Install Wireshark on Linux

To install Wireshark using the following sequence (notice that you'll need to have root permissions):

```
$ sudo apt-get install wireshark  
$ sudo dpkg-reconfigure wireshark-common  
$ sudo usermod -a -G wireshark $USER  
$ newgrp wireshark
```

Once you have completed the above steps, you then log out and log back in, and then start Wireshark:

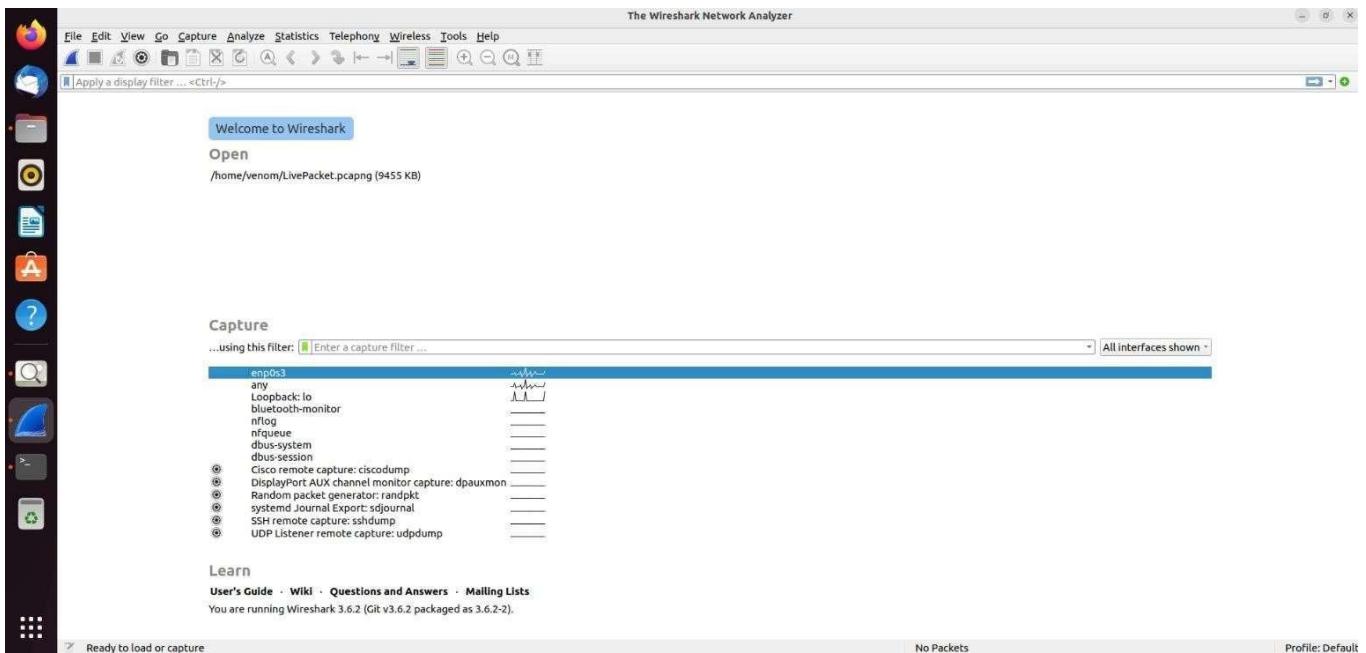
```
$ sudo wireshark
```

How to Capture Packets Using Wireshark

Once you've installed Wireshark, you can start grabbing network traffic. But remember: To capture any packets, you need to have proper permissions on your computer to put Wireshark into promiscuous mode.

- In a Linux system, it usually means that you have root access.

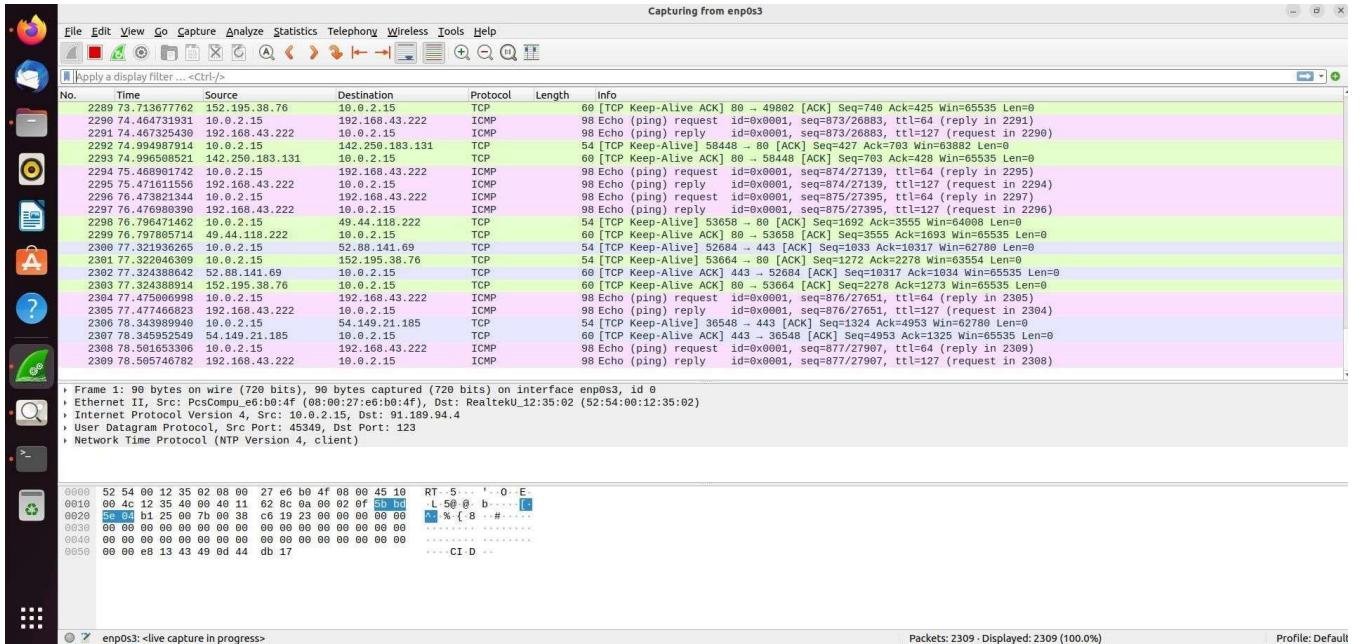
As long as you have the right permissions, you have several options to actually start the capture. Perhaps the best is to select Capture >> Options from the main window. This will bring up the Capture Interfaces window, as shown below



This window will list all available interfaces. In this case, Wireshark provides several to choose from.

For this example, we'll select the wlp interface, which is the most active interface. Wireshark visualizes the traffic by showing a moving line, which represents the packets on the network.

Once the network interface is selected, you simply click the Start button to begin your capture. As the capture begins, it's possible to view the packets that appear on the screen, as shown in image below



In the above image we can see all the packets that are transferred using the current network.

What the Color Coding Means in Wireshark

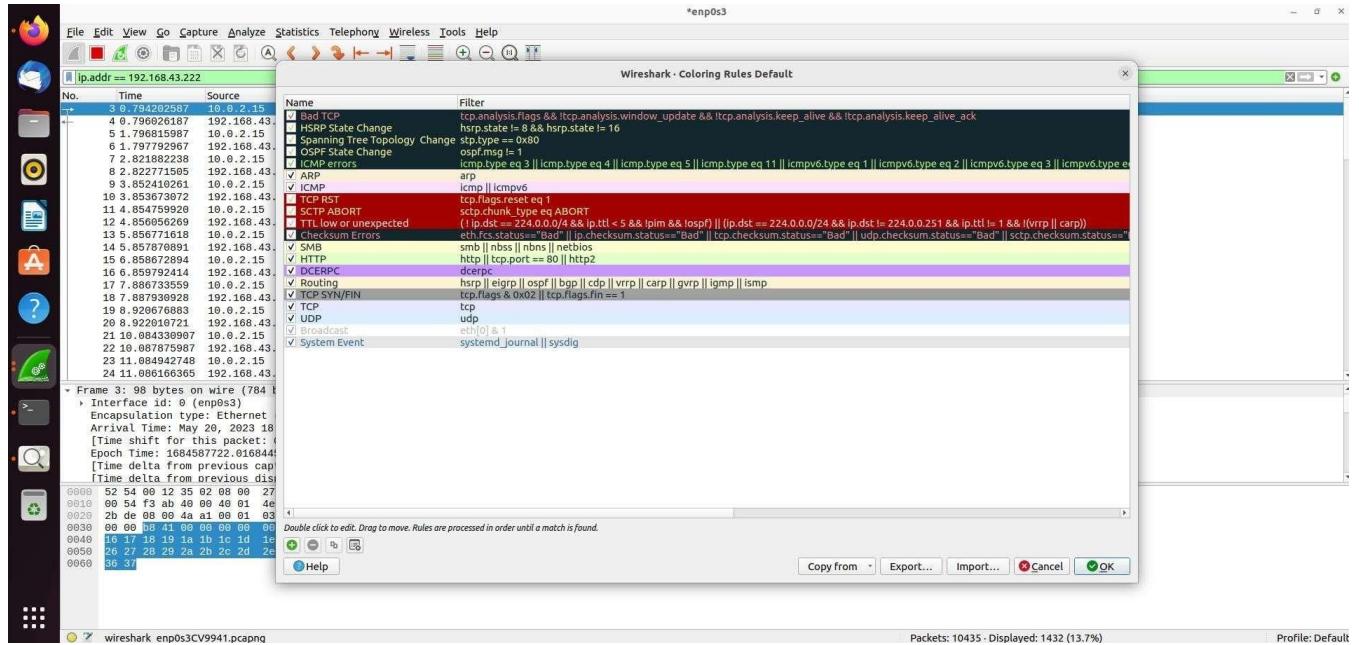
Now that you have some packets, it's time to figure out what they mean. Wireshark tries to help you identify packet types by applying common-sense color coding. The table below describes the default colors given to major packet types.

Color in Wireshark

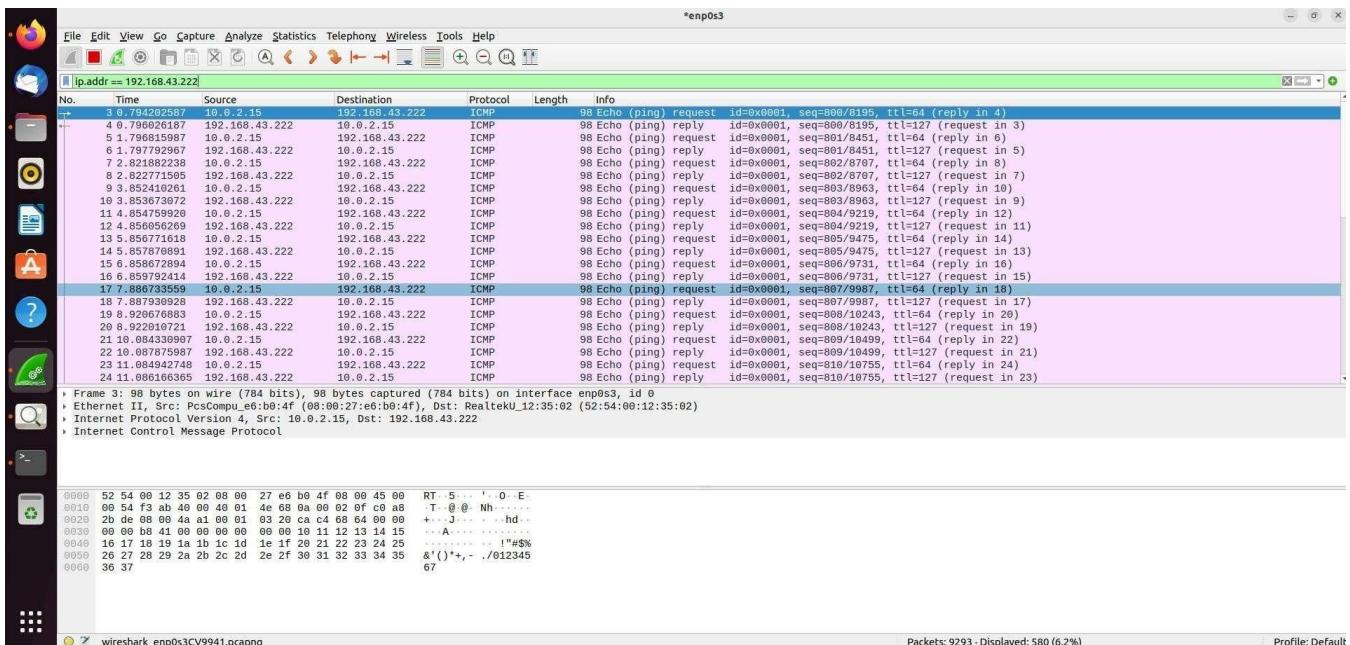
Packet Type

Light purple	TCP
Light blue	UDP
Black	Packets with errors
Light green	HTTP traffic
Light yellow	Windows-specific traffic, including Server Message Blocks (SMB) and NetBIOS
Dark yellow	Routing
Dark gray	TCP SYN, FIN and ACK traffic

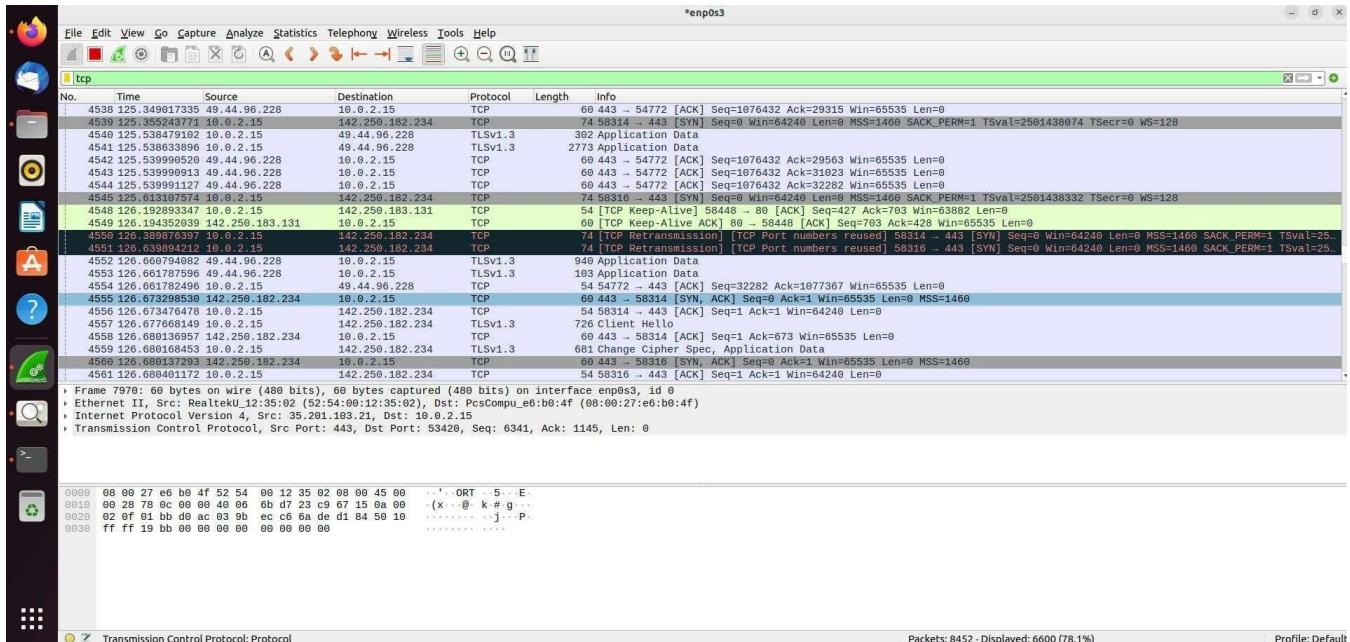
The default coloring scheme is shown below in image below. You can view this by going to View >> Coloring Rules.



You can even change the defaults or apply a custom rule. If you don't want any coloring at all, go to View, then click Colorize Packet List. It's a toggle, so if you want the coloring back, simply go back and click Colorize Packet List again. It's possible, even, to colorize specific conversations between computers.

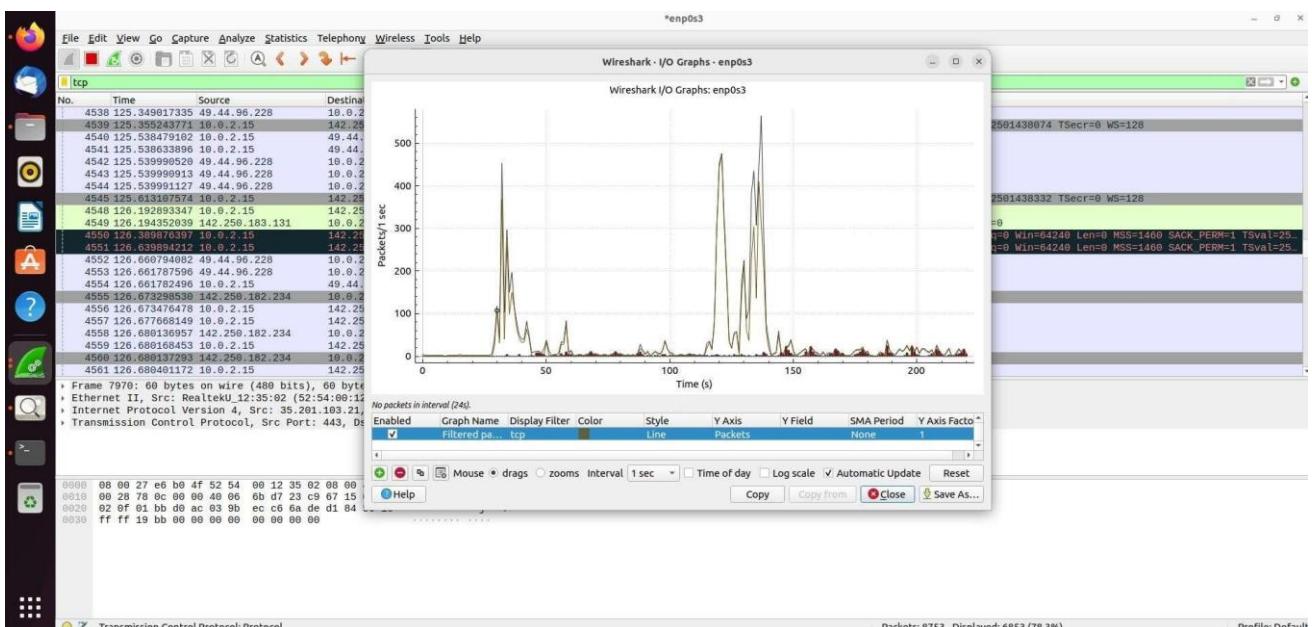


In the above screenshot we can see that there are light blue which indicate TPC protocols, UDP(light blue), TCP (light purple), TCP handshake (dark gray), HTTP (Green), Bad TCP(Black)



However, you're not limited to just interpreting by color. It's possible to view the input/output (I/O)statistics of an entire packet capture.

In Wireshark, just go to Statistics >> I/O Graph, and you'll see a graph similar to the one shown inImage below



How to Filter and Inspect Packets in Wireshark

You can apply Wireshark filters in two ways:

1. In the Display Filter window, at the top of the screen
2. By highlighting a packet (or a portion of a packet) and right-clicking

on the packetWireshark filters use key phrases, such as the following:

ip.addr Specifies an IPv4 address

ipv6.addr Specifies an IPv6 address

src Source - where the packet came
from

dst Destination - where the packet is
going

You can also use the following values:

& Means “and,” as in, “Choose the IP address of 192.168.2.1 and
192.168.2.2”

== Means “equals,” as in “Choose only IP address 192.168.2.1”

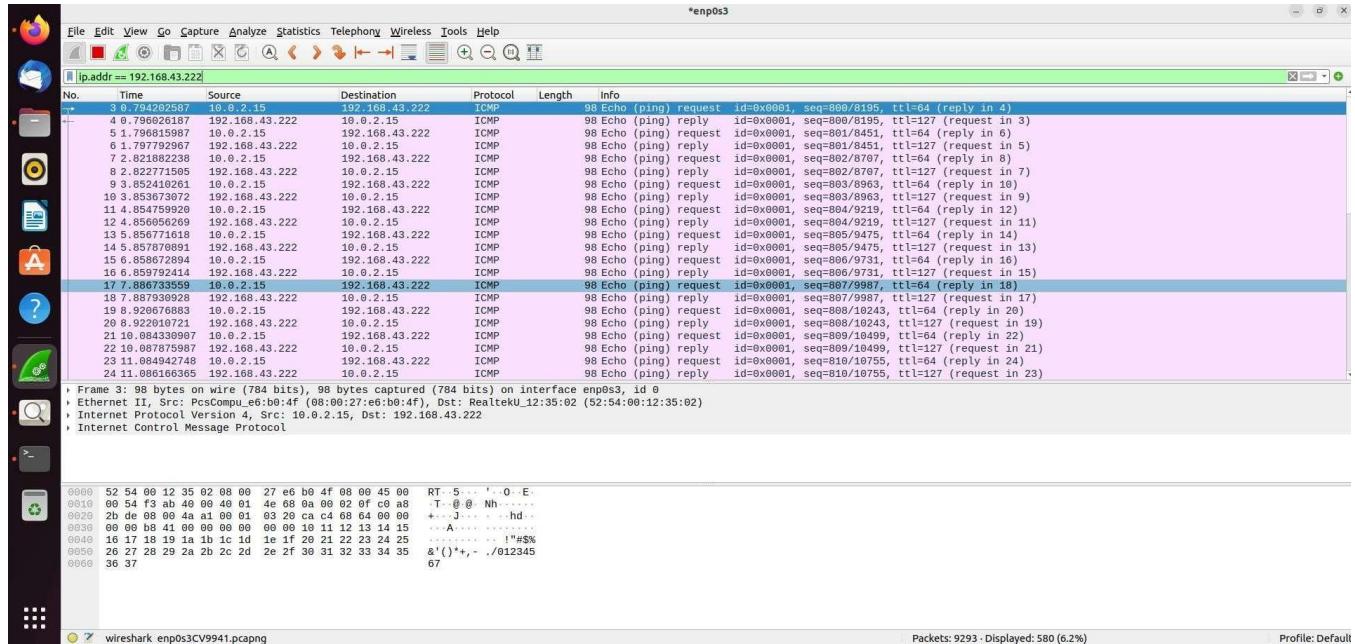
! Means “not,” as in, do not show a particular IP address or source
port

Valid filter rules are always colored green. If you make a mistake on a filter rule, the box will turn avivid pink.

Let’s start with a couple of basic rules. For example, let’s say you want to see packets that have only the IP address of 192.168.53.216 somewhere inside. You would create the following commandline, and put it into the Filter window:

```
ip.addr == 192.168.43.222
```

The image below shows the results of adding that filter:



In this example we have tried to make a connection between two devices using

ping command ping 192.168.43.222

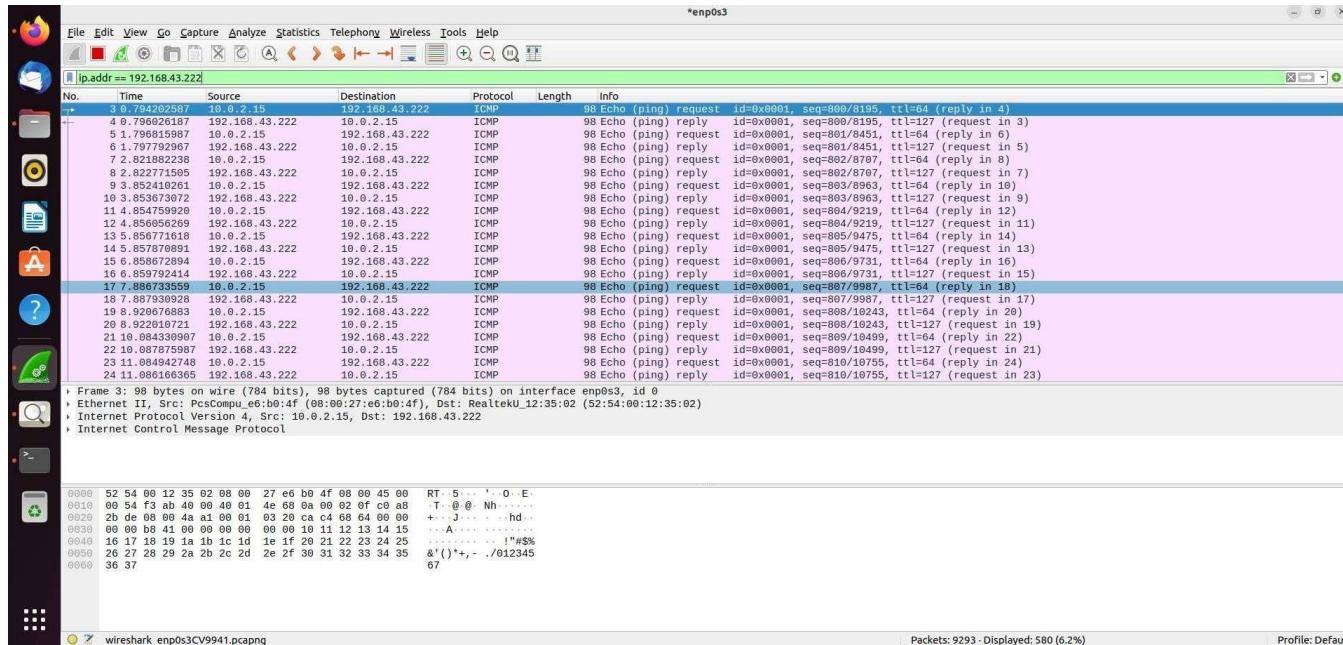
this allows us to send ping request to the device with the IP address 192.168.42.1

In the screenshot below we can observe that the connection have successfully achived using ping

```
venom@venom: ~
64 bytes from 192.168.43.222: icmp_seq=95 ttl=127 time=2.67 ms
64 bytes from 192.168.43.222: icmp_seq=96 ttl=127 time=1.95 ms
64 bytes from 192.168.43.222: icmp_seq=97 ttl=127 time=1.91 ms
64 bytes from 192.168.43.222: icmp_seq=98 ttl=127 time=2.07 ms
64 bytes from 192.168.43.222: icmp_seq=99 ttl=127 time=1.66 ms
64 bytes from 192.168.43.222: icmp_seq=100 ttl=127 time=1.46 ms
64 bytes from 192.168.43.222: icmp_seq=101 ttl=127 time=3.16 ms
64 bytes from 192.168.43.222: icmp_seq=102 ttl=127 time=1.22 ms
64 bytes from 192.168.43.222: icmp_seq=103 ttl=127 time=1.45 ms
64 bytes from 192.168.43.222: icmp_seq=104 ttl=127 time=0.994 ms
64 bytes from 192.168.43.222: icmp_seq=105 ttl=127 time=1.09 ms
64 bytes from 192.168.43.222: icmp_seq=106 ttl=127 time=1.01 ms
64 bytes from 192.168.43.222: icmp_seq=107 ttl=127 time=3.79 ms
64 bytes from 192.168.43.222: icmp_seq=108 ttl=127 time=1.67 ms
64 bytes from 192.168.43.222: icmp_seq=109 ttl=127 time=2.03 ms
64 bytes from 192.168.43.222: icmp_seq=110 ttl=127 time=1.37 ms
64 bytes from 192.168.43.222: icmp_seq=111 ttl=127 time=7.94 ms
64 bytes from 192.168.43.222: icmp_seq=112 ttl=127 time=2.46 ms
64 bytes from 192.168.43.222: icmp_seq=113 ttl=127 time=3.99 ms
64 bytes from 192.168.43.222: icmp_seq=114 ttl=127 time=1.07 ms
64 bytes from 192.168.43.222: icmp_seq=115 ttl=127 time=0.970 ms
64 bytes from 192.168.43.222: icmp_seq=116 ttl=127 time=1.56 ms
64 bytes from 192.168.43.222: icmp_seq=117 ttl=127 time=1.61 ms
64 bytes from 192.168.43.222: icmp_seq=118 ttl=127 time=0.792 ms
64 bytes from 192.168.43.222: icmp_seq=119 ttl=127 time=1.23 ms
64 bytes from 192.168.43.222: icmp_seq=120 ttl=127 time=1.05 ms
64 bytes from 192.168.43.222: icmp_seq=121 ttl=127 time=5.18 ms
64 bytes from 192.168.43.222: icmp_seq=122 ttl=127 time=1.20 ms
64 bytes from 192.168.43.222: icmp_seq=123 ttl=127 time=1.58 ms
64 bytes from 192.168.43.222: icmp_seq=124 ttl=127 time=0.913 ms
64 bytes from 192.168.43.222: icmp_seq=125 ttl=127 time=0.709 ms
64 bytes from 192.168.43.222: icmp_seq=126 ttl=127 time=0.610 ms
64 bytes from 192.168.43.222: icmp_seq=127 ttl=127 time=2.60 ms
64 bytes from 192.168.43.222: icmp_seq=128 ttl=127 time=1.00 ms
64 bytes from 192.168.43.222: icmp_seq=129 ttl=127 time=0.915 ms
64 bytes from 192.168.43.222: icmp_seq=130 ttl=127 time=3.88 ms
64 bytes from 192.168.43.222: icmp_seq=131 ttl=127 time=0.981 ms
64 bytes from 192.168.43.222: icmp_seq=132 ttl=127 time=1.18 ms
64 bytes from 192.168.43.222: icmp_seq=133 ttl=127 time=2.79 ms
64 bytes from 192.168.43.222: icmp_seq=134 ttl=127 time=0.929 ms
64 bytes from 192.168.43.222: icmp_seq=135 ttl=127 time=0.986 ms
64 bytes from 192.168.43.222: icmp_seq=136 ttl=127 time=4.28 ms
64 bytes from 192.168.43.222: icmp_seq=137 ttl=127 time=0.43 ms
64 bytes from 192.168.43.222: icmp_seq=138 ttl=127 time=1.08 ms
64 bytes from 192.168.43.222: icmp_seq=139 ttl=127 time=4.08 ms
64 bytes from 192.168.43.222: icmp_seq=140 ttl=127 time=1.15 ms
```

Now we analyze the ping request using wireshark. In the below image we can now filter using ip.addr == 192.168.42.1 here we can now see all he packets that are getting transferred between the two devices.

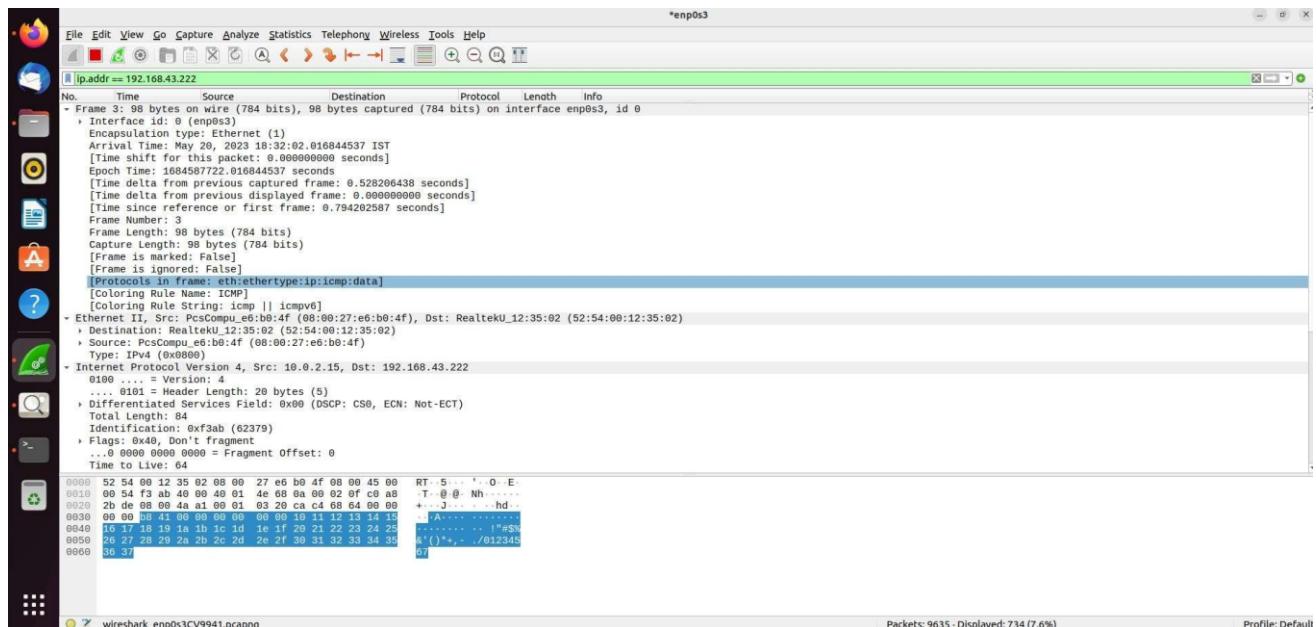
Here there are requests and responses that we can now use to monitor the connection between these two devices.



Request

The below image is the visual representation of the request that was generated. Here we can see the following data

1. Source IP: 192.168.42.222
2. Destination IP: 192.168.42.1
3. Length of the packet: 98 bytes
4. Type of the message: Echo (ping) request
5. Frame number: 2



Response

The below image is the visual representation of the response that was generated. Here we can see the following data

1. Source IP: 192.168.53.85
2. Destination IP: 192.168.53.216
3. Length of the packet: 98 bytes
4. Type of the message: Echo (ping) response
5. Request frame: 3
6. Response time: 206.677 ms

Frame 3: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface enp0s3, id 0
 Interface id: 0 (enp0s3)
 Encapsulation type: Ethernet (1)
 Arrival Time: May 20, 2023 18:32:02.016844537 IST
 [Time shift for this packet: 0.000000000 seconds]
 Epoch Time: 1684587722.016844537 seconds
 [Time delta from previous captured frame: 0.528206438 seconds]
 [Time delta from previous displayed frame: 0.000000000 seconds]
 [Time since reference or first frame: 0.794202587 seconds]
Frame Number: 3
Frame Length: 98 bytes (784 bits)
Capture Length: 98 bytes (784 bits)
[Frame is marked: False]
[Frame is ignored: False]
[Protocols in frame: eth:etherype:ip:icmp:data]
[Coloring Rule Name: ICMP]
[Coloring Rule String: icmp || icmpv6]
Ethernet II, Src: PcsCompu_e6:b0:4f (08:00:27:e6:b0:4f), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)
 Destination: RealtekU_12:35:02 (52:54:00:12:35:02)
 Source: PcsCompu_e6:b0:4f (08:00:27:e6:b0:4f)
 Type: IPv4 (0x0800)
Internet Protocol Version 4, Src: 10.0.2.15, Dst: 192.168.43.222
 Version = Version: 4
 0101 = Header Length: 20 bytes (5)
 Differentiated Services Field: 0x00 (DSSCP: CS0, ECN: Not-ECT)
 Total Length: 84
 Identification: 0xf3ab (62379)
 Flags: 0x40, Don't fragment
 ...0 0000 0000 0000 = Fragment Offset: 0
 Time to Live: 64
 TTL = Time to Live: 64
 0101 = Header Length: 20 bytes (5)
 Differentiated Services Field: 0x00 (DSSCP: CS0, ECN: Not-ECT)
 Total Length: 84
 Identification: 0xf3ab (62379)
 Flags: 0x40, Don't fragment
 ...0 0000 0000 0000 = Fragment Offset: 0
 Time to Live: 64
 T - @ @. Nh.....
 + - J .. .hd
 . F.....
 !#%
 %') *+, - ./012345
 . 67
0000 52 54 00 12 35 02 08 00 27 e6 b0 4f 08 00 45 00 RT 5 .. ' . 0 E.
0001 00 54 f3 ab 40 00 40 01 4e 68 0a 00 02 0f c0 a8 T @ @. Nh.....
0002 2b de 08 00 4a a1 00 01 03 20 ca c4 68 04 00 00 + - J .. .hd
0003 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . F.....
0004 10 38 00 19 1a 1b 1c 1d 1e 1f 20 11 1b 13 14 15 !#%
0005 26 27 28 29 2b 2b 2c 2d 2e 2f 29 31 32 33 34 35 %') *+, - ./012345
0006 36 37 67

Wireshark enp0s3CV9941.pcapng

Packets: 11517 - Displayed: 2420 (21.0%)

Profile: Default

14. Mini Project of Write a program to demonstrate Hybrid Topology in ns3

- Hybrid topology refers to a network configuration that combines multiple types of network topologies. It is a combination of two or more basic network topologies, such as star, ring, bus, mesh, or tree topologies. The aim of using a hybrid topology is to leverage the advantages of different topologies and create a network that suits specific requirements.
- In a hybrid topology, different sections of the network may be connected using different topologies, resulting in a more flexible and scalable network infrastructure. For example, a network might have a star topology in one area, a ring topology in another, and a bus topology connecting these sections. This allows for efficient use of resources, better fault tolerance, improved performance, and increased network capacity.
- Some of the benefits of a hybrid topology include:
- Scalability: Hybrid topologies can be easily expanded by adding or modifying specific sections of the network without affecting the entire infrastructure. This flexibility allows for growth and adaptation to changing network needs.
- Fault tolerance: By combining different topologies, a hybrid network can provide redundancy and fault tolerance. If one section of the network fails, the remaining sections can continue to function, minimizing downtime and improving reliability.
- Performance optimization: Different topologies excel in different aspects. By combining them strategically, a hybrid topology can optimize network performance. For example, a star topology can provide centralized management and easy connectivity, while a mesh topology can offer robust communication between nodes.
- Segmentation: Hybrid topologies allow for network segmentation, enabling specific sections of the network to have their own topologies suited to their requirements. This can enhance security, isolate network issues, and improve overall network management.
- Hybrid topologies are commonly used in large-scale networks, such as enterprise networks, data centers, and telecommunications infrastructure. They provide a versatile and adaptable network architecture that can meet diverse needs and support various applications and services.

Tools used: Ns3 , NetAnim,Wireshark

Code:

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

```

#include "ns3/network-module.h"
#include "ns3/applications-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"
#include "ns3/mobility-module.h"

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 ());

WifiHelper wifi;
wifi.SetRemoteStationManager ("ns3::AarfWifiManager");

WifiMacHelper mac;
Ssid ssid = Ssid ("ns-3-ssid");
mac.SetType ("ns3::StaWifiMac",
    "Ssid", SsidValue (ssid),
    "ActiveProbing", BooleanValue (false));

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

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

NetDeviceContainer apDevices;
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));

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

AnimationInterface anim("ram.xml");
AnimationInterface::SetConstantPosition (csmaNodes.Get(0),10,25);
AnimationInterface::SetConstantPosition (csmaNodes.Get(1),40,25);
anim.EnablePacketMetadata(true);

PcapHelper pcapHelper;
Ptr<PcapFileWrapper> file = pcapHelper.CreateFile ("ram.pcap", std::ios::out, PcapHelper::DLT_PPP);

```

```

Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

Simulator::Stop (Seconds (10.0));

if (tracing)
{
    phy.SetPcapDataLinkType (WifiPhyHelper::DLT_IEEE802_11_RADIO);
    pointToPoint.EnablePcapAll ("ram");
    phy.EnablePcap ("ram", apDevices.Get (0));
    csma.EnablePcap ("ram", csmaDevices.Get (0), true);
}

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

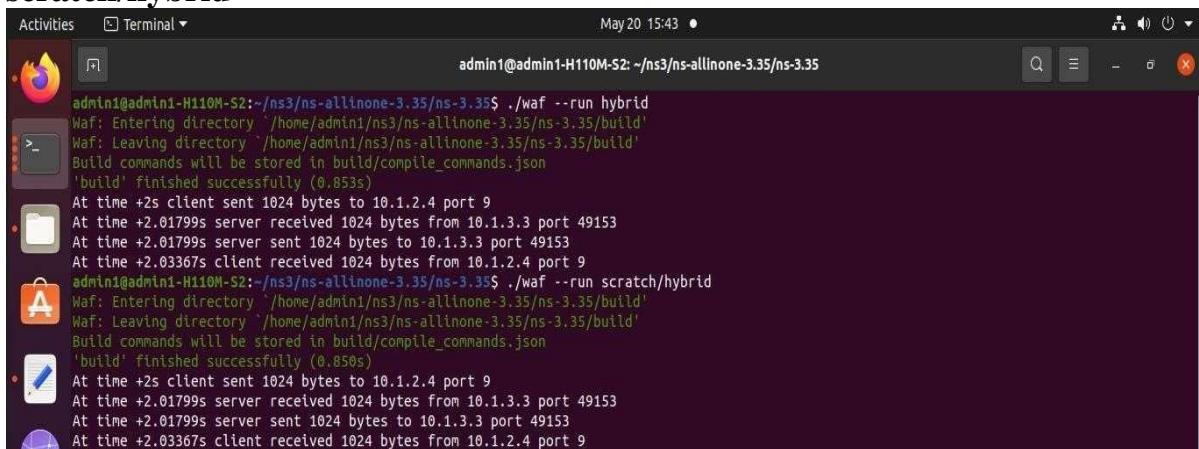
```

Step1: Open terminal and type the following command

./waf –run hybrid

Step2: After successful build the file needs to be run using the following command:

**./waf –run
scratch/hybrid**



```

Activities Terminal May 20 15:43 •
admin1@admin1-H110M-S2:~/ns3/ns-allinone-3.35$ ./waf --run hybrid
Waf: Entering directory '/home/admin1/ns3/ns-allinone-3.35/ns-3.35/build'
Waf: Leaving directory '/home/admin1/ns3/ns-allinone-3.35/ns-3.35/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (0.853s)
At time +2s client sent 1024 bytes to 10.1.2.4 port 9
At time +2.01799s server received 1024 bytes from 10.1.3.3 port 49153
At time +2.01799s server sent 1024 bytes to 10.1.3.3 port 49153
At time +2.03367s client received 1024 bytes from 10.1.2.4 port 9
admin1@admin1-H110M-S2:~/ns3/ns-allinone-3.35/ns-3.35$ ./waf --run scratch/hybrid
Waf: Entering directory '/home/admin1/ns3/ns-allinone-3.35/ns-3.35/build'
Waf: Leaving directory '/home/admin1/ns3/ns-allinone-3.35/ns-3.35/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (0.850s)
At time +2s client sent 1024 bytes to 10.1.2.4 port 9
At time +2.01799s server received 1024 bytes from 10.1.3.3 port 49153
At time +2.01799s server sent 1024 bytes to 10.1.3.3 port 49153
At time +2.03367s client received 1024 bytes from 10.1.2.4 port 9

```

Setp 3: In order to activate the pyvis the following command needs to be entered

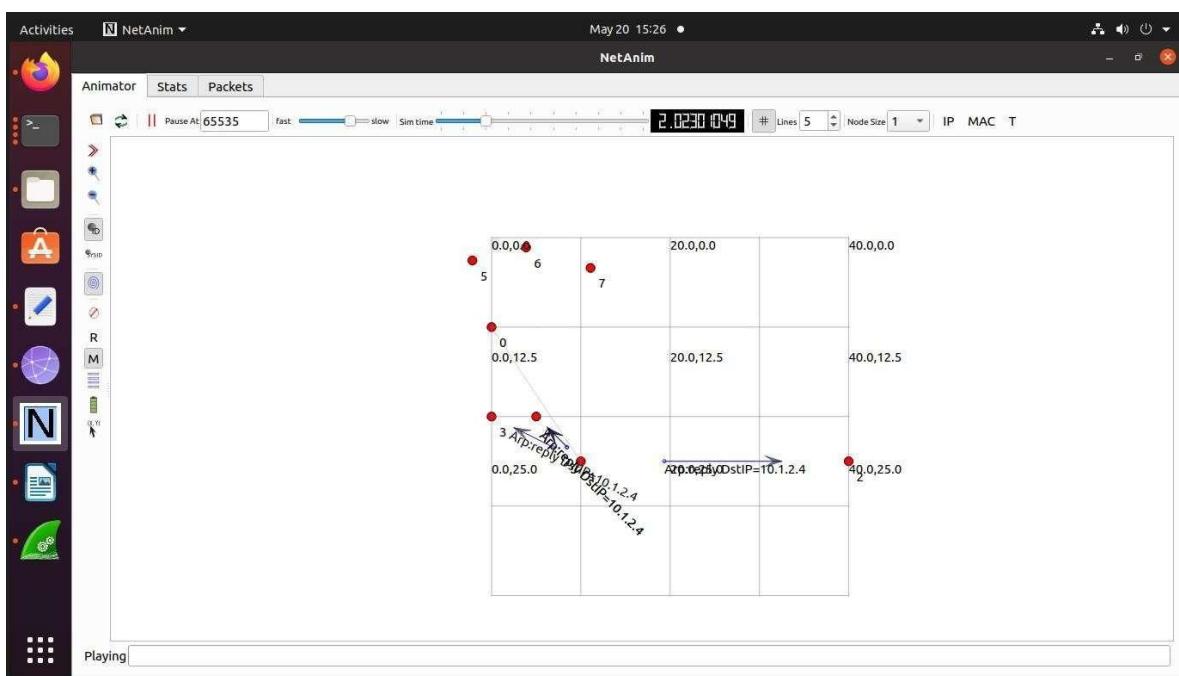
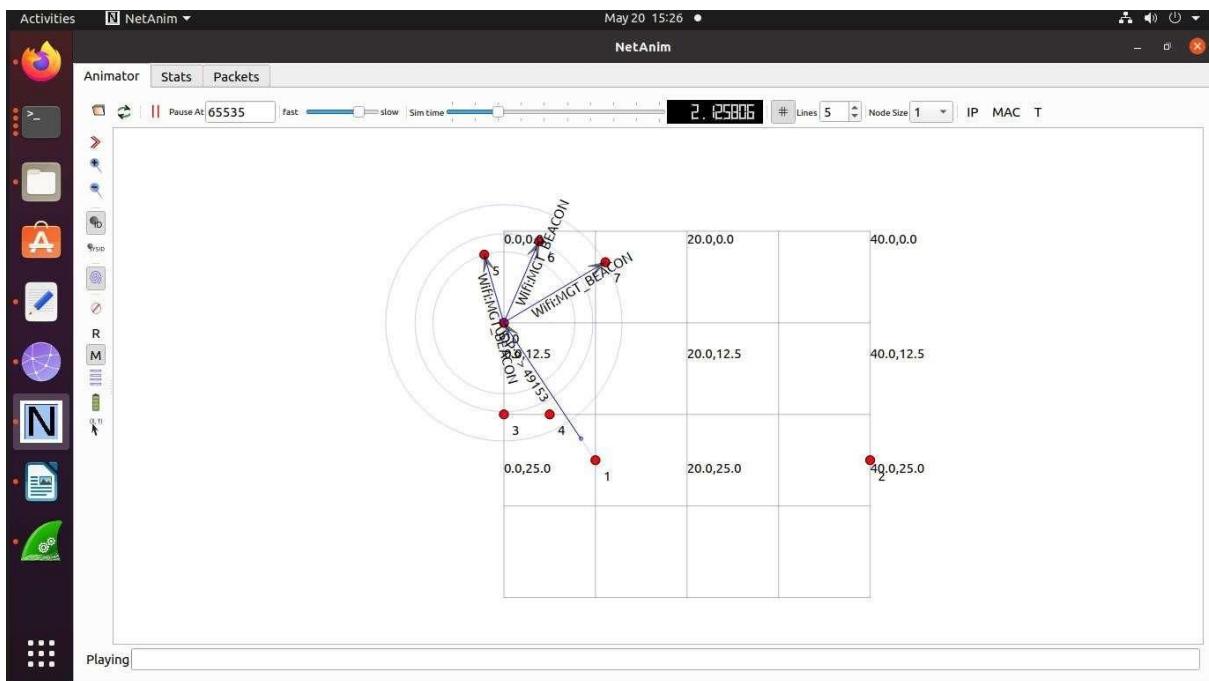
“./waf –run scratch/filename --vis”

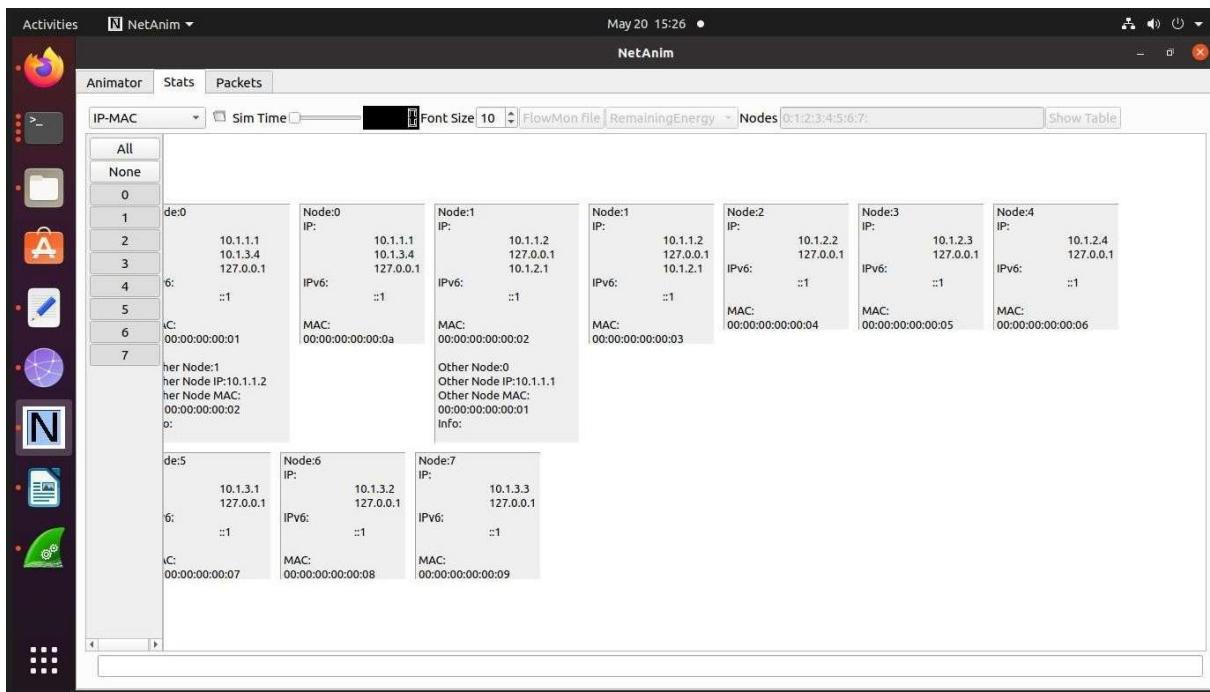
```
admin1@admin1-H110M-S2:~/ns3/ns-allinone-3.35/ns-3.35$ ./waf --run scratch/hybrid --vis
Waf: Entering directory '/home/admin1/ns3/ns-allinone-3.35/ns-3.35/build'
Waf: Leaving directory '/home/admin1/ns3/ns-allinone-3.35/ns-3.35/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (0.989s)
```



Step 4 : Move to netanim folder and type following command and open hybrid.xml

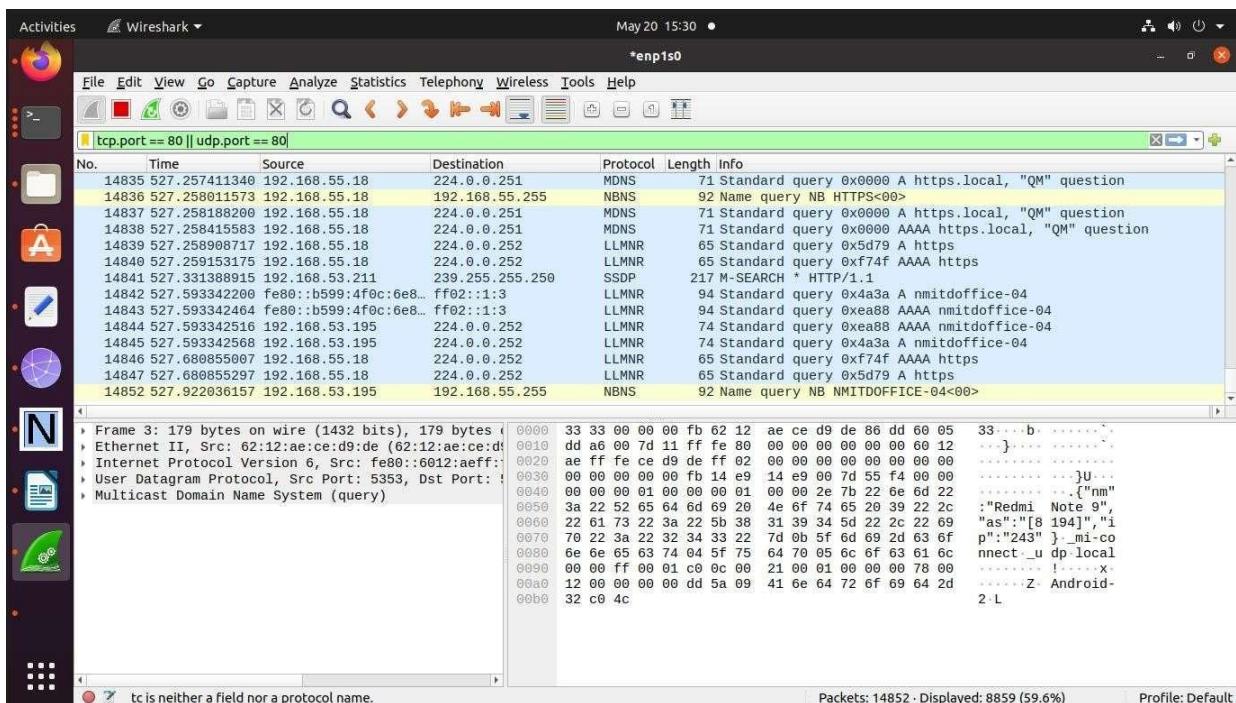
./NetAnim



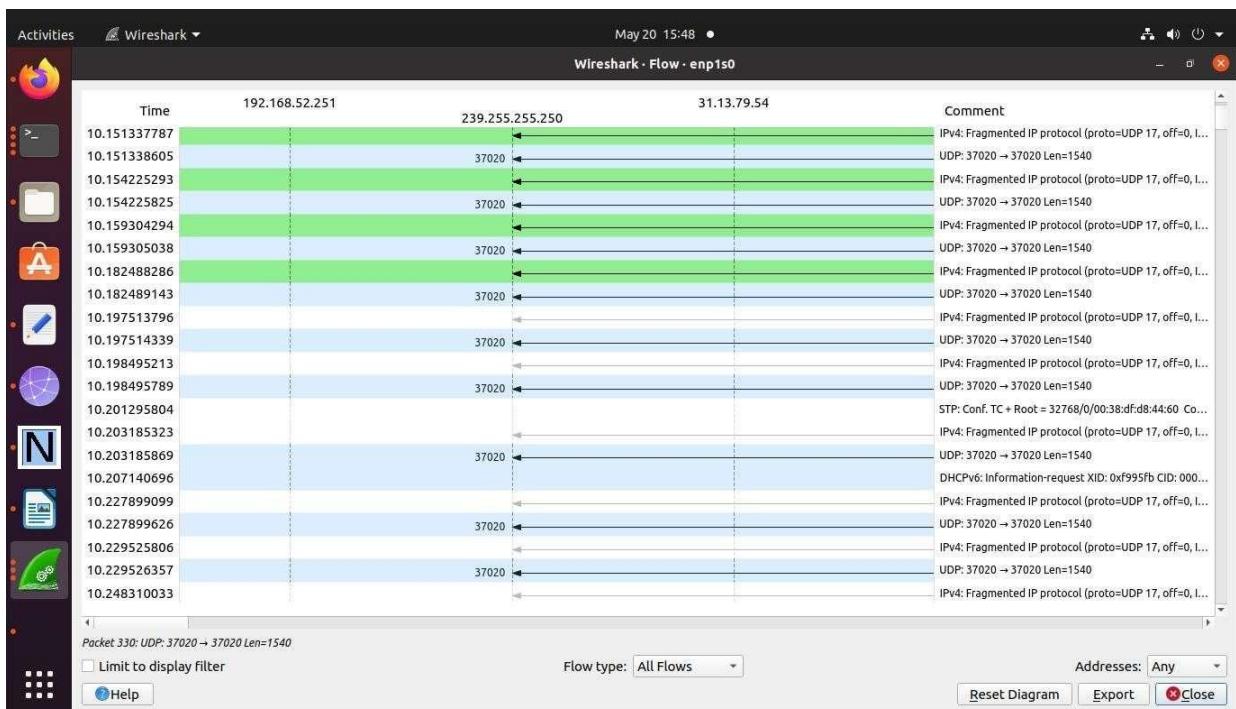
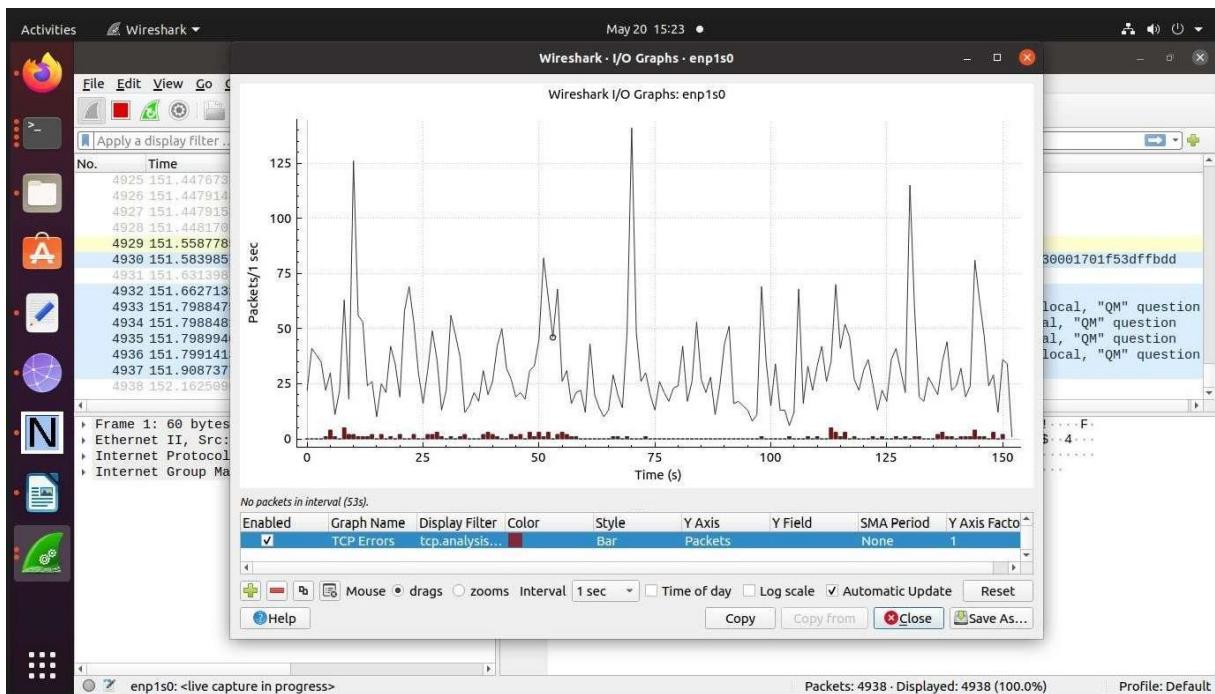


Step5: In Terminal Type wireshark and open pcap file generated

And Click On Start Capturing



Network throughput:



Conclusion

It proves to be more flexible, reliable, effective and scalable than other networking topologies but at the same time difficult to manage and

costly also. Although its complex structure can create hazards due to its robust and effective features gained a lot of popularity.