**Practical No. 1**

**Aim:** Installation of NS-3 in Linux.

**Commands:**

List of Packages for Installing ns-3 in Ubantu/Mint Systems

1. sudo apt upgrade

2. Sudo apt update

3. Minimal requirements for C++ users

apt-get install g++ python3

4. Minimal requirements for Python API users

apt-get install g++ python3 python3-dev pkg-config sqlite3

5. Netanim animator: qt5 development tools are needed for Netanim animator;

apt-get install qt5-default mercurial

6. ns-3-pyviz visualizer

apt-get install gir1.2-goocanvas-2.0 python-gi python-gi-cairo python-

pygraphviz python3-gi python3-gi-cairo python3-pygraphviz gir1.2-gtk-3.0

ipython ipython3

7. Debugging:

8. apt-get install gdbvalgrind

9. Doxygen and related inline documentation:

apt-get install doxygengraphvizimagemagick

apt-get install texlivetexlive-extra-utilstexlive-latex-extra texlive-font-

utilsdvipnglatexmk

10. The ns-3 manual and tutorial are written in reStructuredText for Sphinx

(doc/tutorial, doc/manual, doc/models), and figures typically in dia (also

needs the texlive packages above):

apt-get install python3-sphinx dia

11. To read pcap packet traces

apt-get install tcpdump

12. Support for generating modified python bindings

apt-get install cmake libc6-dev libc6-dev-i386 libclang-6.0-dev llvm-6.0-dev

automake python3-pip

python3 -m pip install --user cxxfilt

After installing the required packages, create a folder named workspace in thehome

directory and then put the NS3 tar package into the workspace.

Go to terminal and input these commands consecutively after each commandfinishes

executing:

cd

cd workspace

tar xjf<name of NS3 Downloaded file name>

cd <name of extracted NS3>

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

It takes time be patient !!

Test the NS3 build and installation success by running test.py in the ns

directory using the following commands:

cd ns-<version number>

./test.py

**Practical No. 2**

**Aim:** Installation of NetAnim.

**Process:**

NetAnim is a Network Animator that comes preloaded with ns3.  During compilation of ns3, #NetAnim may not compile.  
  
There are other alternatives for installation of NetAnim, for more info, see this post  
  
<http://www.nsnam.com/2014/08/installing-netanim-software-for-ns3-in.html>  
  
This post will tell you to compile NetAnim that comes along with #ns3. Sometimes the above methods might not work, but the bundled netanim will work sure.  
Here is the way to do...  
  
**$] sudo apt-get update**  
**$] sudo apt-get install qt4-dev-tools qt4-qmake, qt4-qmake qt4-default**  
**$] cd ns-allinone-3.24.1/netanim-3.106/**  
**$] make clean**  
**$] make**  
**$] qmake NetAnim.pro**  
**$] ./NetAnim file.xml**

**Practical No. 3**

**Aim:** Program to simulate traffic.

**Program:**

**First.cc**

/\* -\*- 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"

// 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));

//pointToPoint.EnablePcapAll("first");

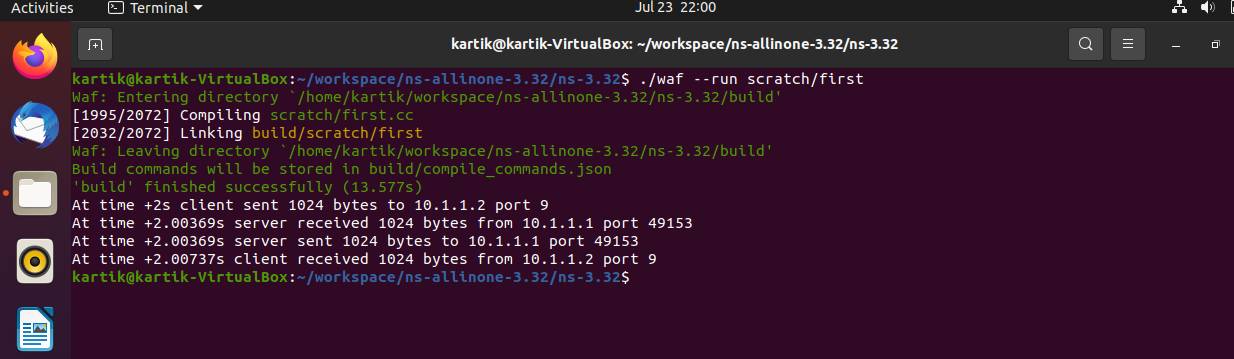
Simulator::Run ();

Simulator::Destroy ();

return 0;

}

**Output:**

****

**Practical No. 4**

**Aim:** Program to simulate traffic between two nodes.

**Program:**

**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"

#include "ns3/mobility-module.h"

#include "ns3/onoff-application.h"

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));

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);

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

clientApps.Start (Seconds (2.0));

clientApps.Stop (Seconds (10.0));

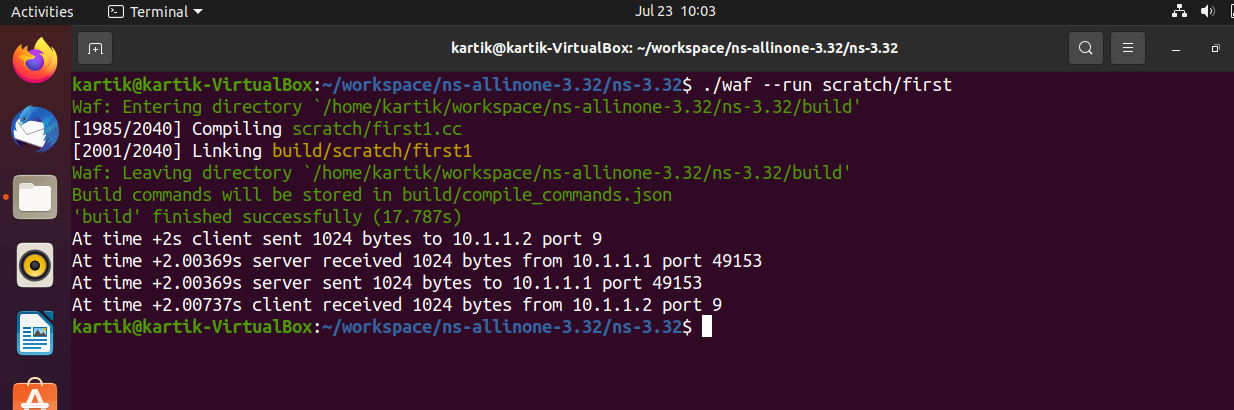
Simulator::Run ();

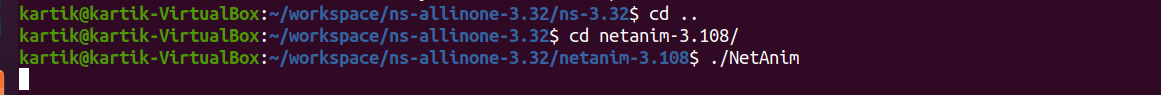
Simulator::Destroy ();

return 0;

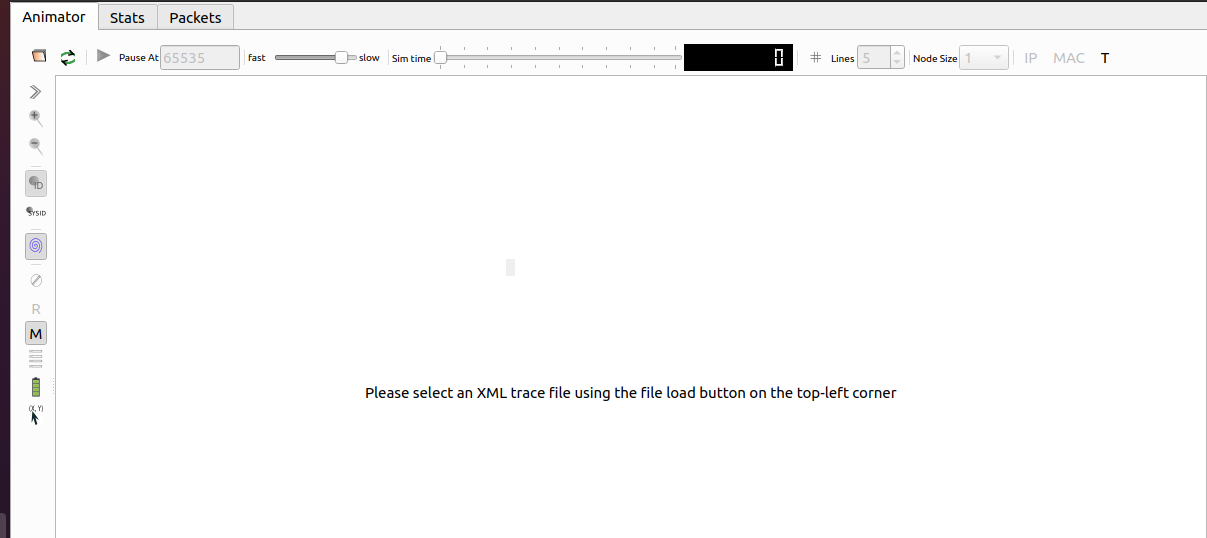
}

**Output:**

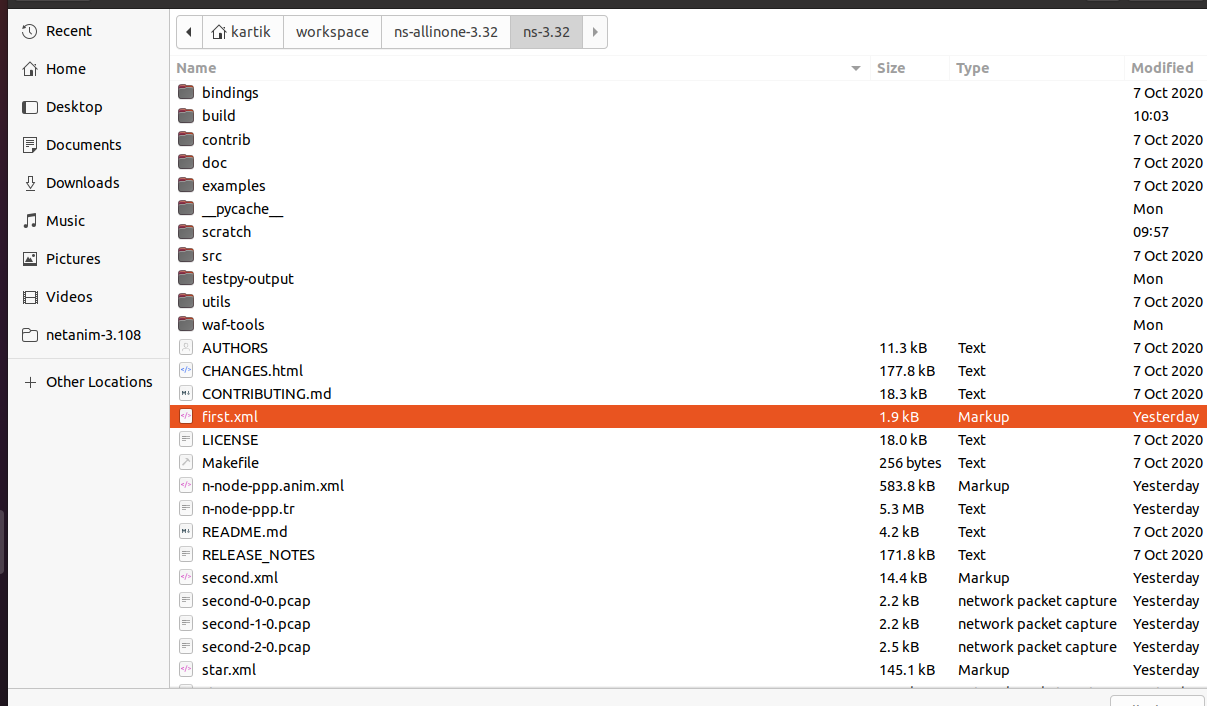




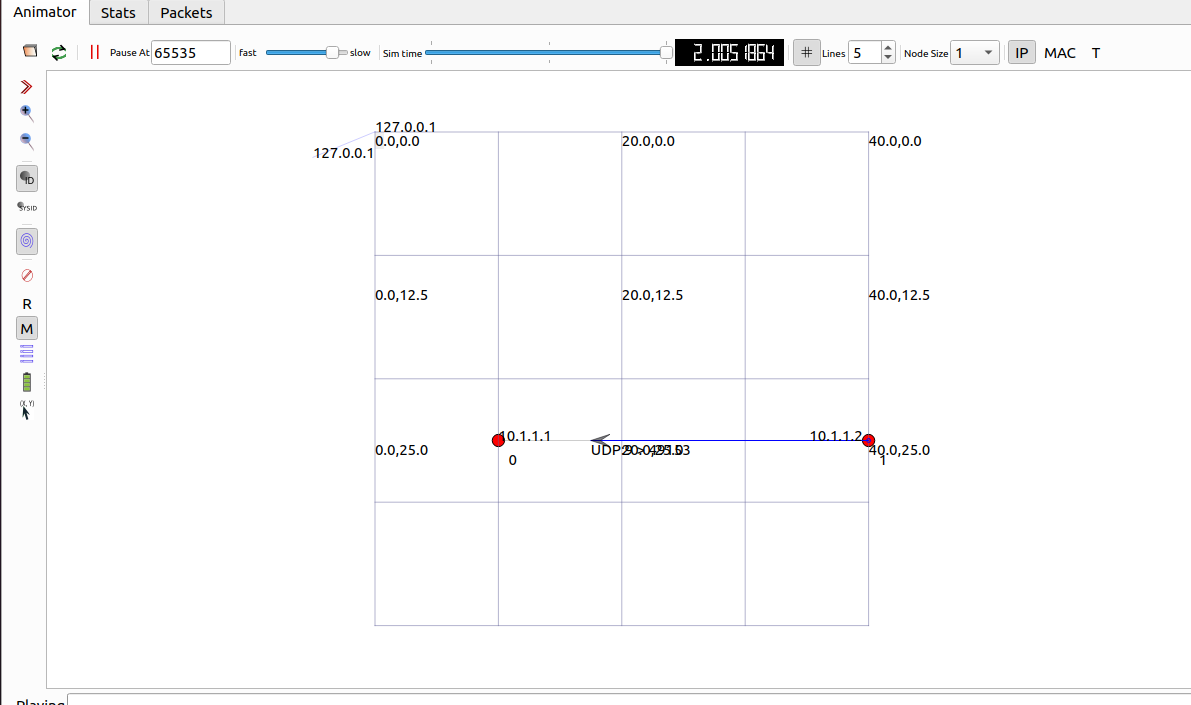
Click on folder icon



Choose first.xml file and open



Run



**Practical No. 5**

**Aim:** Program to simulate bus topology.

**Program:**

**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"

#include "ns3/mobility-module.h"

using namespace ns3;

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

int

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

{

bool verbose = true;

uint32\_t nCsma = 3;

CommandLine cmd (\_\_FILE\_\_);

cmd.AddValue ("nCsma", "Number of \"extra\" CSMA nodes/devices", nCsma);

cmd.AddValue ("verbose", "Tell echo applications to log if true", verbose);

cmd.Parse (argc,argv);

if (verbose)

{

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

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

}

nCsma = nCsma == 0 ? 1 : nCsma;

NodeContainer p2pNodes;

p2pNodes.Create (2);

NodeContainer csmaNodes;

csmaNodes.Add (p2pNodes.Get (1));

csmaNodes.Create (nCsma);

PointToPointHelper pointToPoint;

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

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

NetDeviceContainer p2pDevices;

p2pDevices = pointToPoint.Install (p2pNodes);

CsmaHelper csma;

csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));

csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));

NetDeviceContainer csmaDevices;

csmaDevices = csma.Install (csmaNodes);

InternetStackHelper stack;

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

stack.Install (csmaNodes);

Ipv4AddressHelper address;

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

Ipv4InterfaceContainer p2pInterfaces;

p2pInterfaces = address.Assign (p2pDevices);

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

Ipv4InterfaceContainer csmaInterfaces;

csmaInterfaces = address.Assign (csmaDevices);

UdpEchoServerHelper echoServer (9);

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

serverApps.Start (Seconds (1.0));

serverApps.Stop (Seconds (10.0));

UdpEchoClientHelper echoClient (csmaInterfaces.GetAddress (nCsma), 9);

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

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

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

ApplicationContainer clientApps = echoClient.Install (p2pNodes.Get (0));

clientApps.Start (Seconds (2.0));

clientApps.Stop (Seconds (10.0));

Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

pointToPoint.EnablePcapAll ("second");

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

// for netanim

MobilityHelper mobility;

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

mobility.Install(p2pNodes);

mobility.Install(csmaNodes);

AnimationInterface anim("second.xml");

AnimationInterface::SetConstantPosition (p2pNodes.Get(0), 10, 25);

// position of x and y

AnimationInterface::SetConstantPosition(p2pNodes.Get(1), 20,25);

// position of x an y

// csma node

AnimationInterface ::SetConstantPosition(csmaNodes.Get(1), 30,25);

AnimationInterface ::SetConstantPosition(csmaNodes.Get(2), 40,25);

AnimationInterface ::SetConstantPosition(csmaNodes.Get(3), 50,25);

anim.EnablePacketMetadata(true);

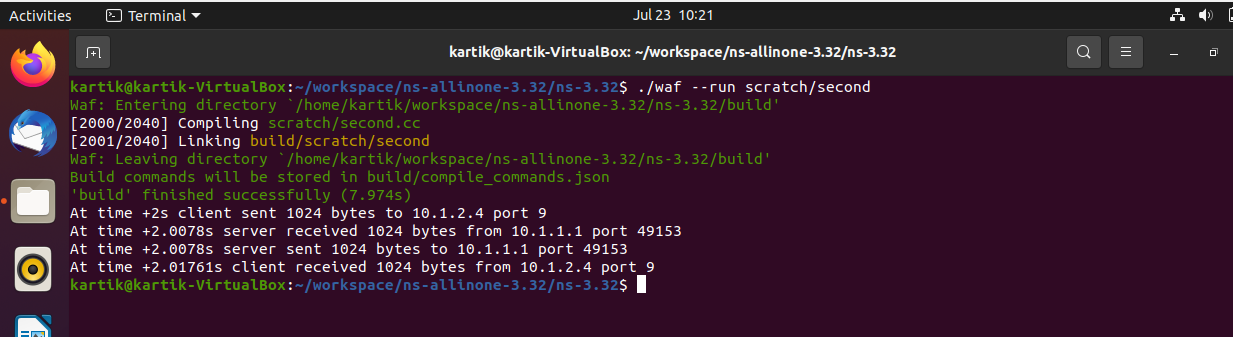
Simulator::Run ();

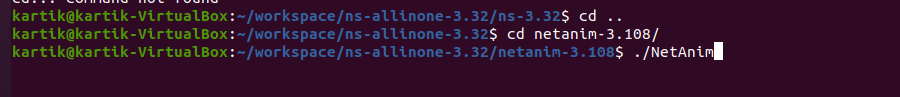
Simulator::Destroy ();

return 0;

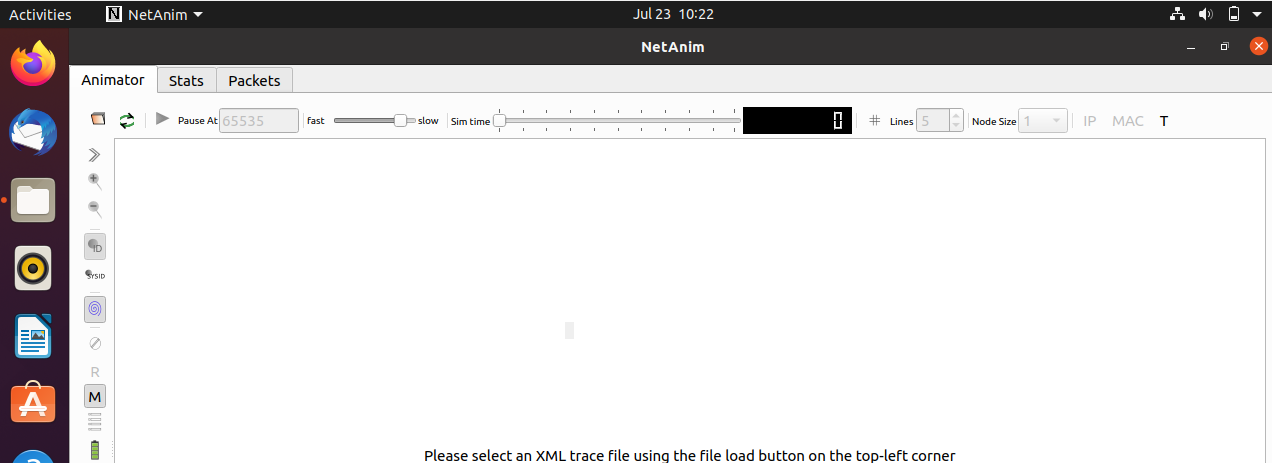
}

**Output:**

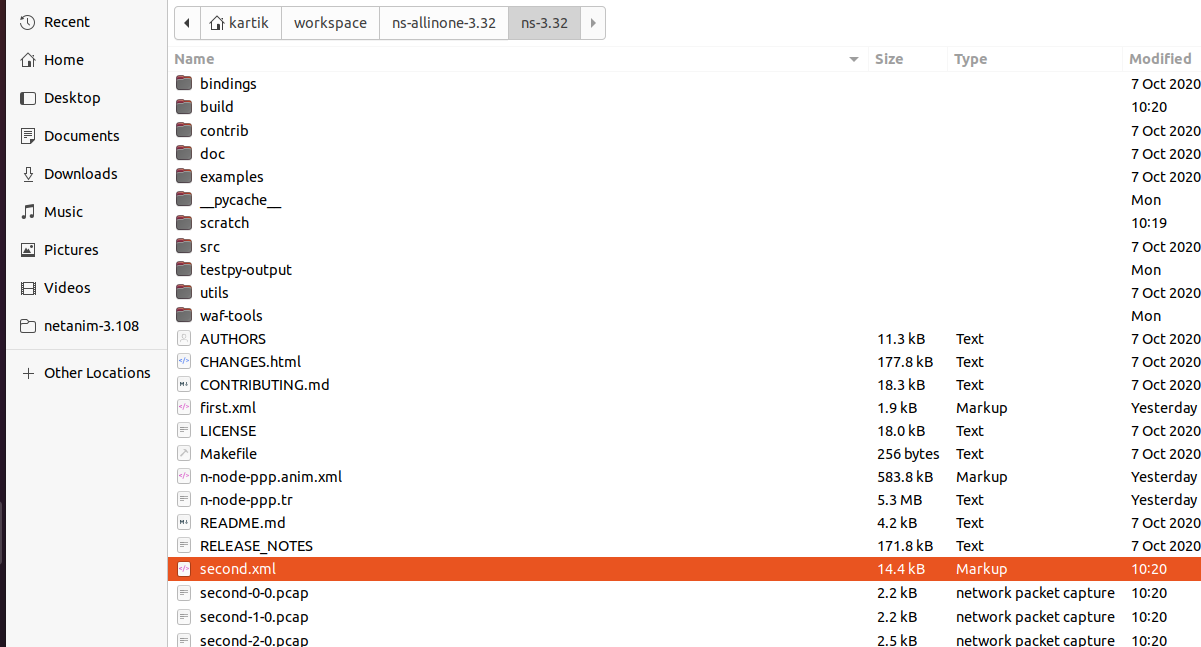




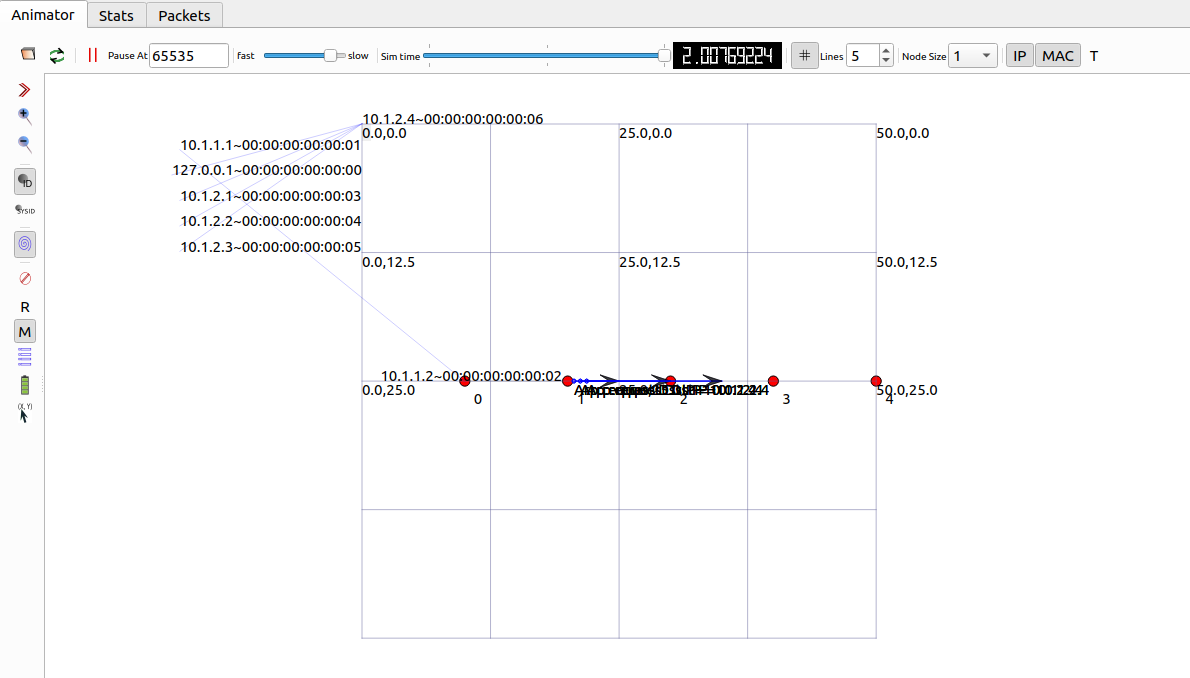
Click on folder icon



Choose second.xml file and open



Run



**Practical No. 6**

**Aim:** Program to simulate star topology.

**Program:**

**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;

CommandLine cmd (\_\_FILE\_\_);

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");

star.BoundingBox(1,1,100,100);

AnimationInterface anim("star.xml");

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

Simulator::Run ();

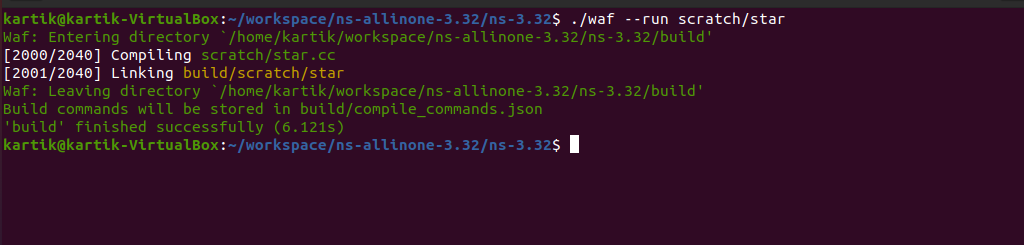
Simulator::Destroy ();

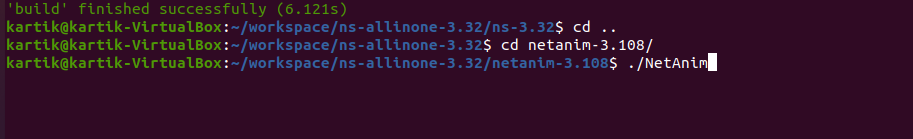
NS\_LOG\_INFO ("Done.");

return 0;

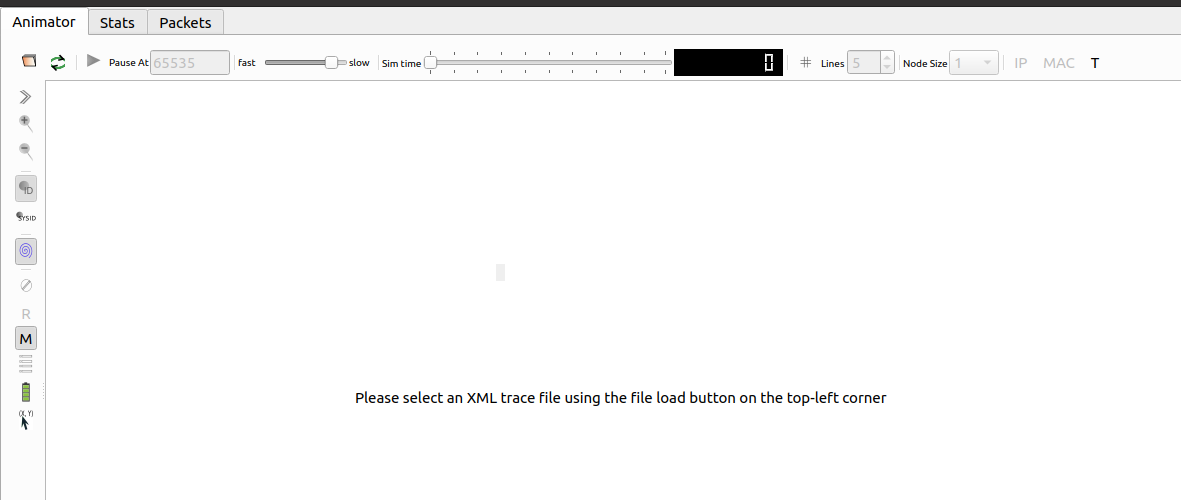
}

**Output:**

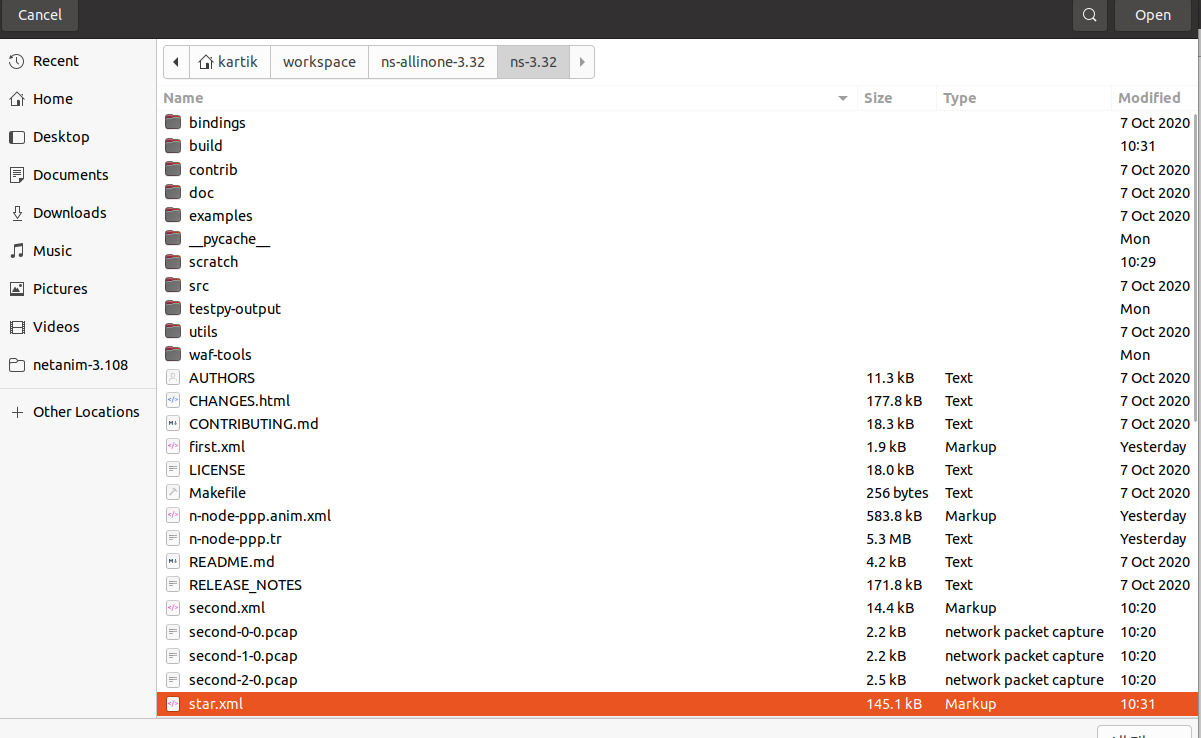




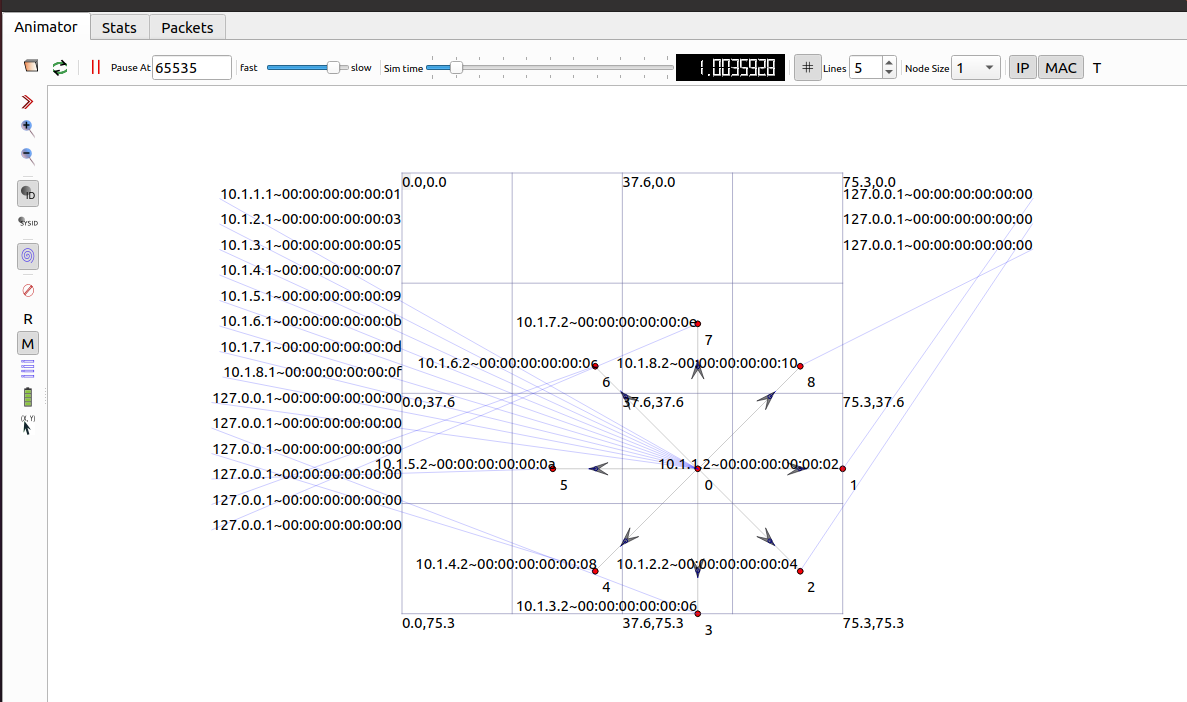
Click on folder icon



Choose star.xml file and open



Run



**Practical No. 7**

**Aim:** simple topology using Wireshark (first.cc).

**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"

// 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));

pointToPoint.EnablePcapAll("first");

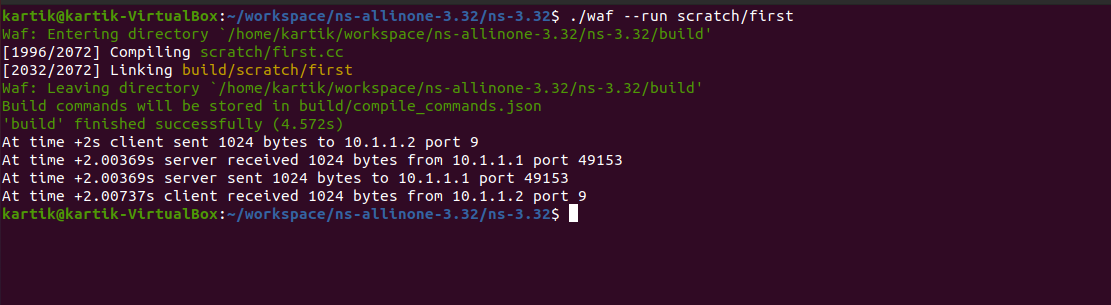
Simulator::Run ();

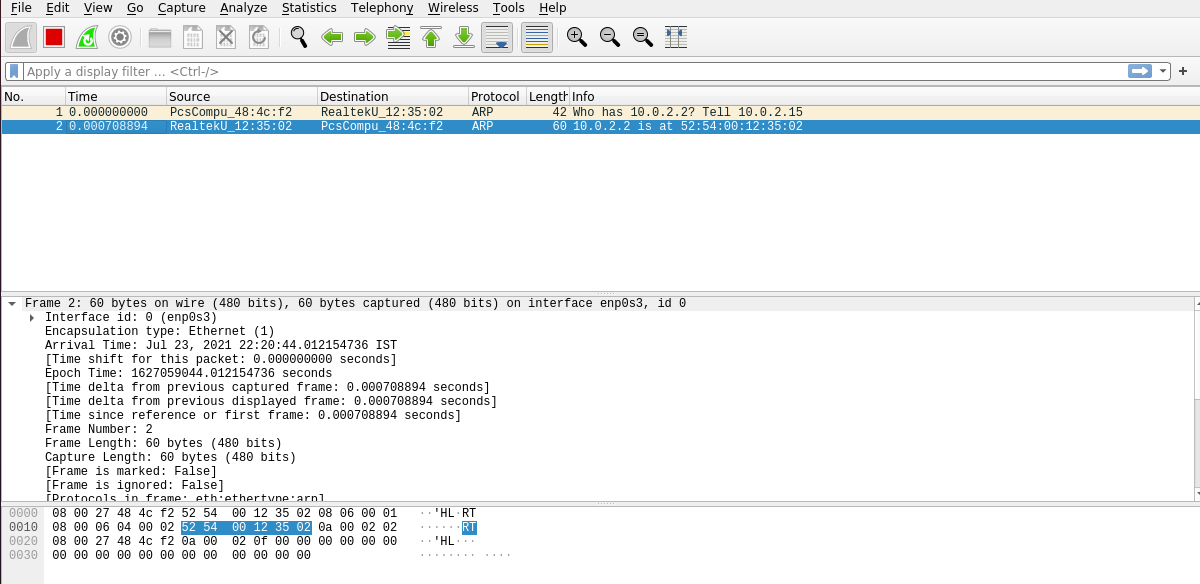
Simulator::Destroy ();

return 0;

}

**Output :**





**Practical No. 8**

**Aim :** simple topology using wifi yans (third.cc)

**Program:**

**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"

// 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 = true;

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 = YansWifiPhyHelper::Default ();

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));

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);

}

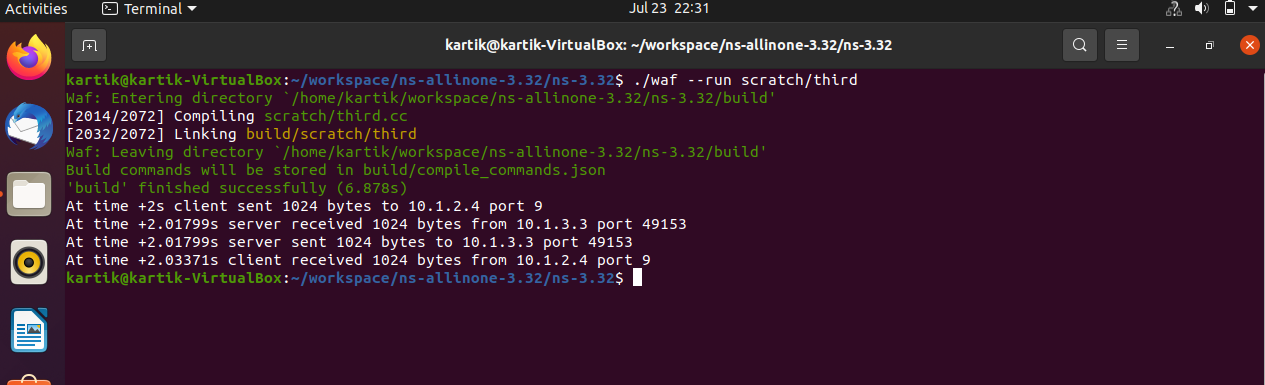
Simulator::Run ();

Simulator::Destroy ();

return 0;

}

**Output:**



**Practical No. 9**

**Aim:** simple topology using wifi yans --vis (third.cc)

**Program:**

**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"

// 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 = true;

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 = YansWifiPhyHelper::Default ();

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));

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);

}

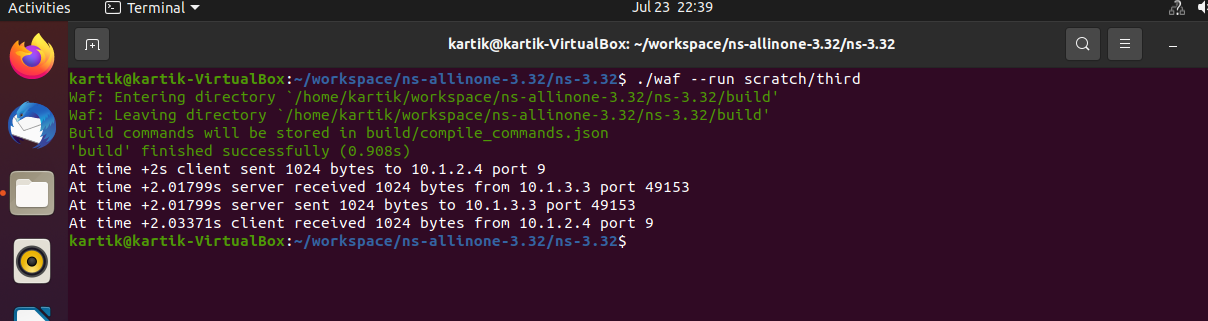
Simulator::Run ();

Simulator::Destroy ();

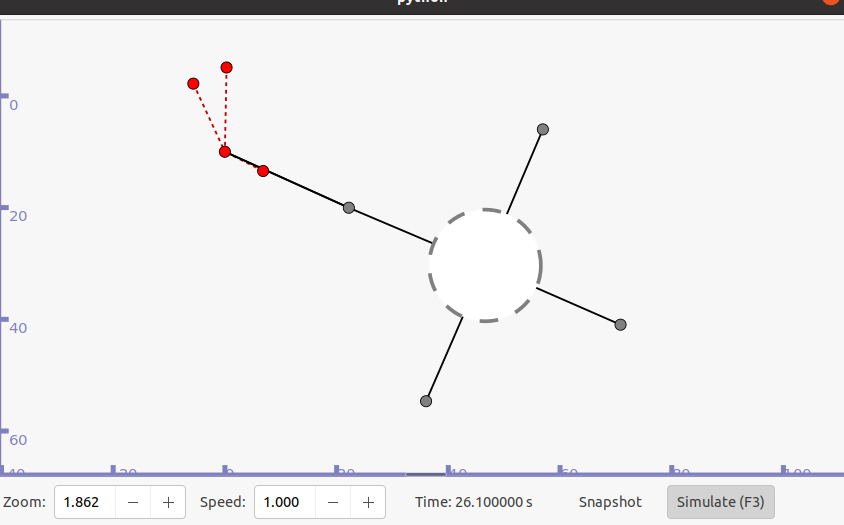
return 0;

}

**Output:**

****

****

****

**Practical No. 10**

**Aim:** simple topology using wifi yans wireshark(third.cc)

**Program:**

**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"

// 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 = true;

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 = YansWifiPhyHelper::Default ();

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));

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);

}

Simulator::Run ();

Simulator::Destroy ();

return 0;

}

**Output:**

