| **Name of Student:** Ajay Karthikesan | | | |
| --- | --- | --- | --- |
| **Roll Number:** 57 | | **Assignment Number:** 2 | |
| **Aim of Assignment:**  Write a Program to simulate traffic between two nodes( point to point). | | | |
| **DOP:** 10.4.23 | | **DOS:** 29.4.23 | |
| **CO Mapped:**  CO2 | **PO Mapped:**  PO1, PO2, PO3, PO5, PO7, PSO1 | **Faculty Signature:** | **Marks:** |

## 

## Practical No. 2

**Aim:** Write a Program to simulate traffic between two nodes( point to point)

**Theory:**

* CommandLine class
  + Parse command-line arguments
  + Instances of this class can be used to parse command-line arguments. Programs can register a general usage message with CommandLine::Usage, and arguments with CommandLine::AddValue.
* NodeContainer class
  + keep track of a set of node pointers.
  + Typically ns-3 helpers operate on more than one node at a time. For example a device helper may want to install devices on a large number of similar nodes. The helper Install methods usually take a NodeContainer as a parameter. NodeContainers hold the multiple Ptr<Node> which are used to refer to the nodes.
* PointToPointHelper
  + Build a set of PointToPointNetDevice objects
  + Normally we eschew multiple inheritance, however, the classes PcapUserHelperForDevice and AsciiTraceUserHelperForDevice are "mixins".
* NetDeviceContainer
  + holds a vector of ns3::NetDevice pointers
  + Typically ns-3 NetDevices are installed on nodes using a net device helper. The helper Install method takes a NodeContainer which holds some number of Ptr<Node>. For each of the Nodes in the NodeContainer the helper will instantiate a net device, add a MAC address and a queue to the device and install it to the node. For each of the devices, the helper also adds the device into a Container for later use by the caller.
  + This is that container used to hold the Ptr<NetDevice> which are instantiated by the device helper.
* InternetStackHelper
  + aggregate IP/TCP/UDP functionality to existing Nodes.
  + This helper enables pcap and ascii tracing of events in the internet stack associated with a node. This is substantially similar to the tracing that happens in device helpers, but the important difference is that, well,
  + there is no device. This means that the creation of output file names will change, and also the user-visible methods will not reference devices and therefore the number of trace enable methods is reduced.
  + Normally we avoid multiple inheritance in ns-3, however, the classes PcapUserHelperForIpv4 and AsciiTraceUserHelperForIpv4 are treated as "mixins". A mixin is a self-contained class that encapsulates a general attribute or a set of functionality that may be of interest to many other classes.
* Ipv4AddressHelper
  + A helper class to make life easier while doing simple IPv4 address assignment in scripts.
  + This class is a very simple IPv4 address generator. You can think of it as a simple local number incrementer. It has no notion that IP addresses are part of a global address space. If you have a complicated address assignment situation you may want to look at the Ipv4AddressGenerator which does recognize that IP address and network number generation is part of a global problem. Ipv4AddressHelper is a simple class to make simple problems easy to handle.
  + We do call into the global address generator to make sure that there are no duplicate addresses generated.
* Ipv4InterfaceContainer
  + holds a vector of std::pair of Ptr<Ipv4> and interface index.
  + Typically ns-3 Ipv4Interfaces are installed on devices using an Ipv4 address helper. The helper's Assign() method takes a NetDeviceContainer which holds some number of Ptr<NetDevice>. For each of the NetDevices in the NetDeviceContainer the helper will find the associated Ptr<Node> and Ptr<Ipv4>. It makes sure that an interface exists on the node for the device and then adds an Ipv4Address according to the address helper settings (incrementing the Ipv4Address somehow as it goes). The helper then converts the Ptr<Ipv4> and the interface index to a std::pair and adds them to a container -- a container of this type.
  + The point is then to be able to implicitly associate an index into the original NetDeviceContainer (that identifies a particular net device) with an identical index into the Ipv4InterfaceContainer that has a std::pair with the Ptr<Ipv4> and interface index you need to play with the interface.
* UdpEchoServerHelper
  + Create a server application which waits for input UDP packets and sends them back to the original sender.
* UdpEchoClientHelper
  + Create an application which sends a UDP packet and waits for an echo of this packet
* ApplicationContainer
  + holds a vector of ns3::Application pointers.
  + Typically ns-3 Applications are installed on nodes using an Application helper. The helper Install method takes a NodeContainer which holds some number of Ptr<Node>. For each of the Nodes in the NodeContainer the helper will instantiate an application, install it in a node and add a Ptr<Application> to that application into a Container for use by the caller. This is that container used to hold the Ptr<Application> which are instantiated by the Application helper.
* Simulator
  + Control the scheduling of simulation events.
  + The internal simulation clock is maintained as a 64-bit integer in a unit specified by the user through the Time::SetResolution function. This means that it is not possible to specify event expiration times with anything better than this user-specified accuracy. Events whose expiration time is the same modulo this accuracy are scheduled in FIFO order: the first event inserted in the scheduling queue is scheduled to expire first.
* Node
  + A network Node.
  + This class holds together:
    - a list of NetDevice objects which represent the network interfaces of this node which are connected to other Node instances through Channel instances.
    - a list of Application objects which represent the userspace traffic generation applications which interact with the Node through the Socket API.
    - a node Id: a unique per-node identifier.
    - a system Id: a unique Id used for parallel simulations.
* NetDevice
  + Network layer to device interface
  + This interface defines the API which the IP and ARP layers need to access to manage an instance of a network device layer. It currently does not support MAC-level multicast but this should not be too hard to add by adding extra methods to register MAC multicast addresses to filter out unwanted packets before handing them to the higher layers.
  + In Linux, this interface is analogous to the interface just above dev\_queue\_xmit() (i.e., IP packet is fully constructed with destination MAC address already selected).

**Code:**

#include "ns3/animation-interface.h"

#include "ns3/applications-module.h"

#include "ns3/core-module.h"

#include "ns3/internet-module.h"

#include "ns3/log.h"

#include "ns3/mobility-module.h"

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

#include <cstdint>

using namespace ns3;

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

// look for any command line arguments and act on it

CommandLine(\_\_FILE\_\_).Parse(argc, argv);

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

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

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

// create nodes(computers)

NodeContainer nodeContainer;

nodeContainer.Create(2);

// create net devices and channels internally

PointToPointHelper p2p;

p2p.SetDeviceAttribute("DataRate", StringValue("5Mbps"));

p2p.SetChannelAttribute("Delay", StringValue("2ms"));

// store net devices for later use

NetDeviceContainer netDevices = p2p.Install(nodeContainer);

// install Internet Stack like (UDP, TCP, IP, etc.)

InternetStackHelper{}.Install(nodeContainer);

Ipv4AddressHelper ipv4AddHelper;

ipv4AddHelper.SetBase("192.168.1.0", "255.255.255.0");

Ipv4InterfaceContainer ipv4InterfaceContainer =

ipv4AddHelper.Assign(netDevices);

// setup minimal udp echo client and udp echo server

constexpr std::uint32\_t UDP\_ECHO\_SERVER\_PORT\_NUMBER{57};

UdpEchoServerHelper udpEchoServerHelper(UDP\_ECHO\_SERVER\_PORT\_NUMBER);

UdpEchoClientHelper udpEchoClientHelper(ipv4InterfaceContainer.GetAddress(1),

UDP\_ECHO\_SERVER\_PORT\_NUMBER);

udpEchoClientHelper.SetAttribute("MaxPackets", UintegerValue(20));

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

udpEchoClientHelper.SetAttribute("Interval", TimeValue(Seconds(1)));

// setup application

ApplicationContainer udpServerApps =

udpEchoServerHelper.Install(nodeContainer.Get(1));

ApplicationContainer udpClientApps =

udpEchoClientHelper.Install(nodeContainer.Get(0));

// configure udp client and server

udpServerApps.Start(Seconds(1.0));

udpServerApps.Stop(Seconds(20.0));

udpClientApps.Start(Seconds(2.0));

udpClientApps.Stop(Seconds(20.0));

// generate trace file

MobilityHelper mobility;

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

mobility.Install(nodeContainer);

AnimationInterface{"test.xml"};

AnimationInterface::SetConstantPosition(nodeContainer.Get(0), 1, 1);

AnimationInterface::SetConstantPosition(nodeContainer.Get(1), 5, 1);

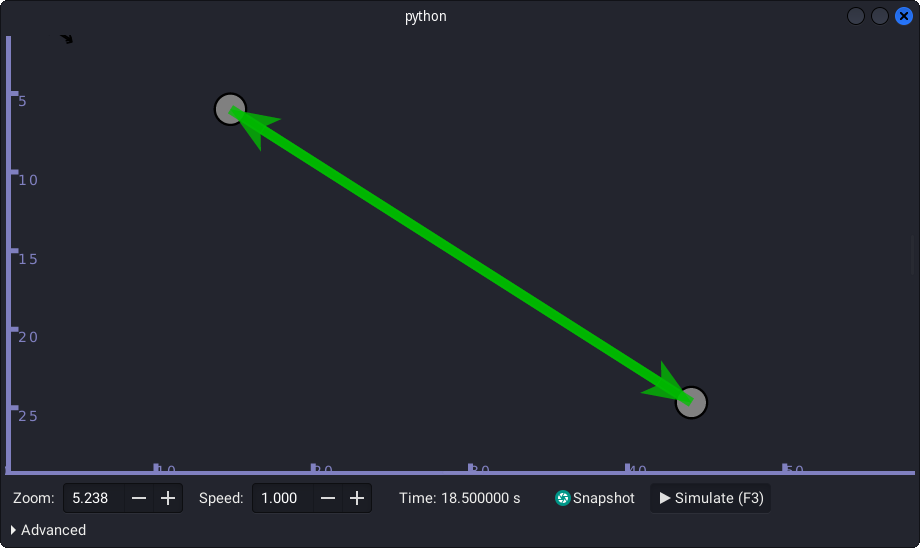
Simulator::Run();

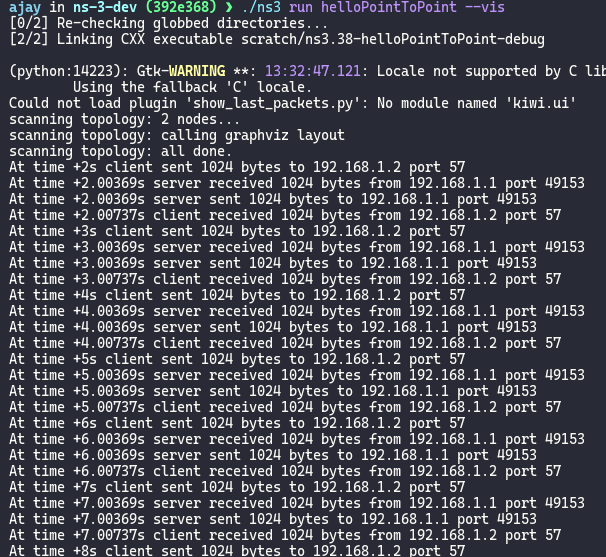
Simulator::Destroy();

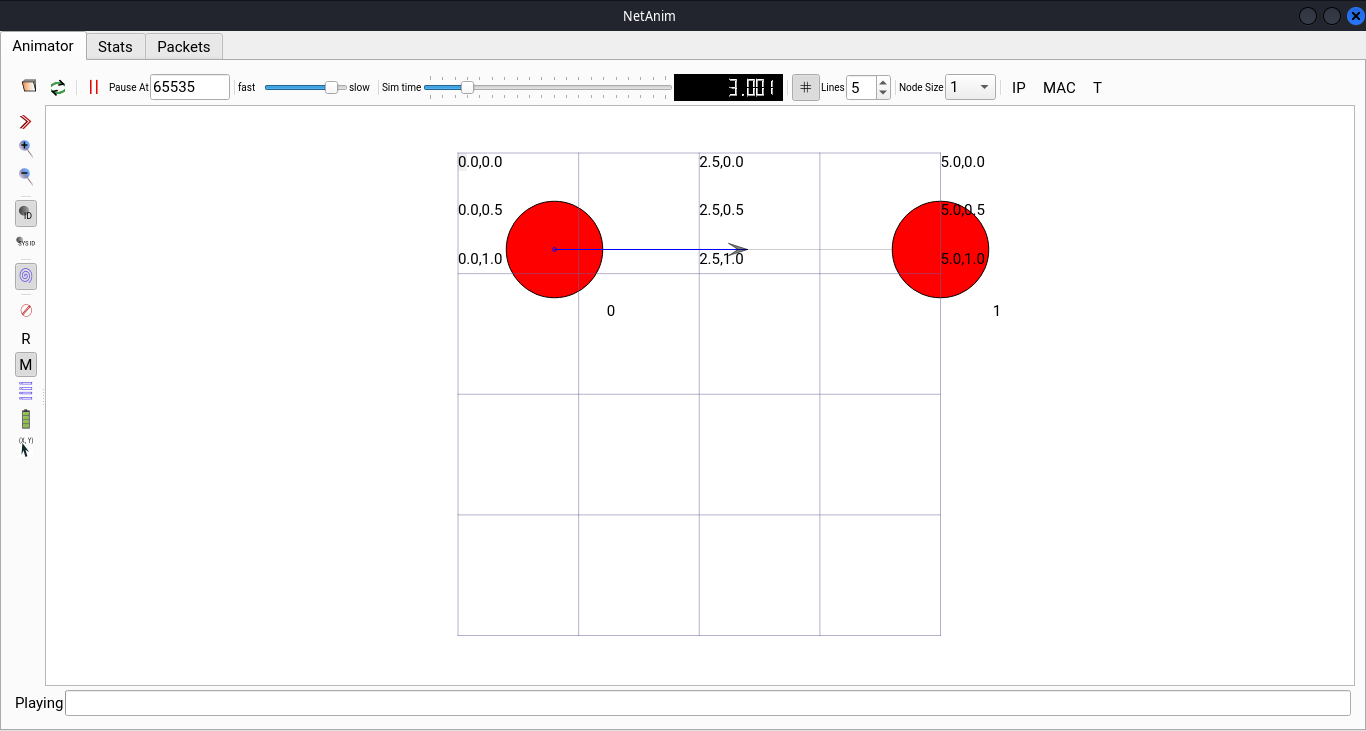
return 0;

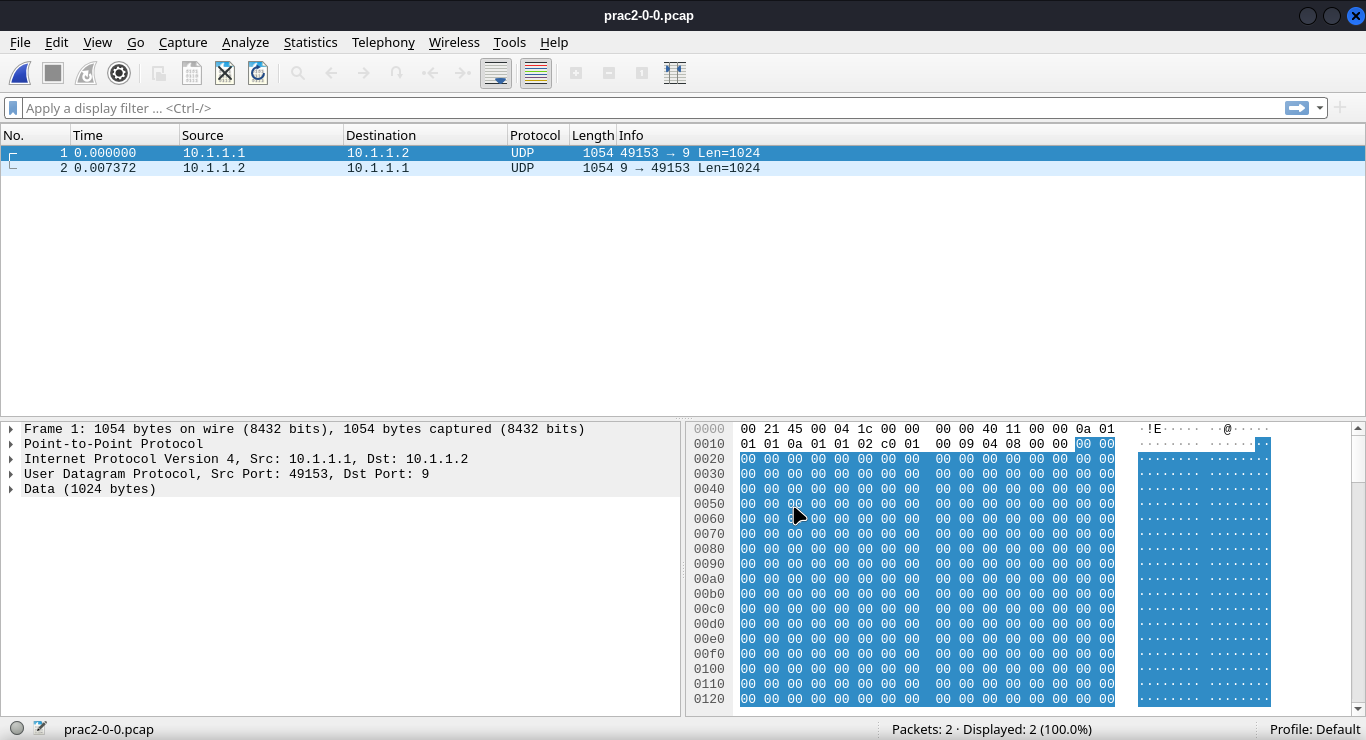
}

**Output:**









**Conclusion:**

I learnt how to write a program to simulate traffic between two nodes( point to point) using ns-3.