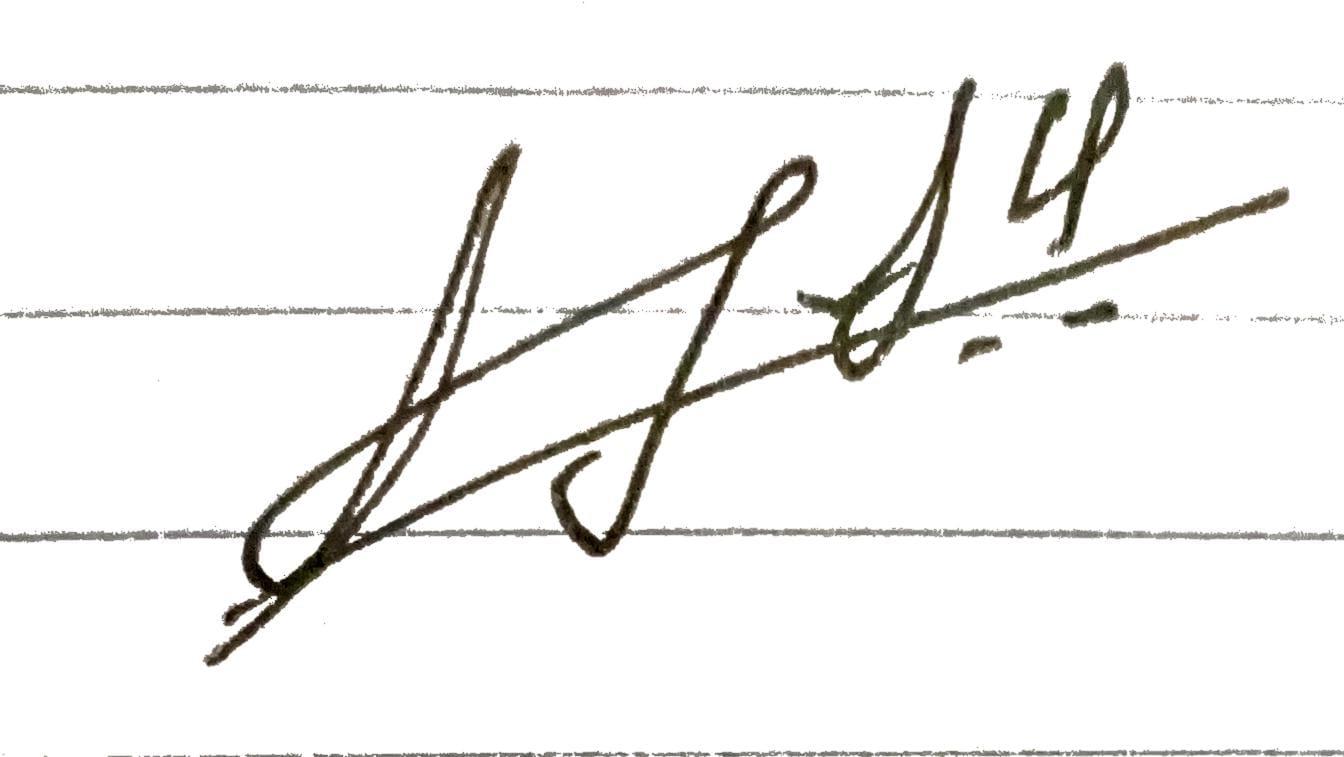
| **Name of Student : Sunny Satish Halkatti** | | | |
| --- | --- | --- | --- |
| **Roll Number : 17** | | **LAB Assignment Number:** **6** | |
| **Title of LAB Assignment :**  Program to simulate UDP server client.  . | | | |
| **DOP :** 27-04-2023 | | **DOS :** 04-05-2023 | |
| **CO Mapped :**  CO2 | **PO Mapped:**  PO1, PO2, PO3, PO5, PO7, PSO1 | **Faculty Signature :** | |

**Student’s Signature**

****

**NWL - Practical-6**

**Aim:** Program to simulate UDP server client.

**Theory:**

**User Datagram Protocol (UDP)** is a Transport Layer protocol. UDP is a part of the Internet Protocol suite, referred to as UDP/IP suite. Unlike TCP, it is an unreliable and connectionless protocol. So, there is no need to establish a connection prior to data transfer. The **UDP** helps to establish low-latency and loss-tolerating connections established over the network.The **UDP** enables process to process communication.

Though **Transmission Control Protocol (TCP)** is the dominant transport layer protocol used with most of the Internet services, it provides assured delivery, reliability, and much more but all these services cost us additional overhead and latency. Here, UDP comes into the picture. For real-time services like computer gaming, voice or video communication, live conferences; we need UDP. Since high performance is needed, UDP permits packets to be dropped instead of processing delayed packets. There is no error checking in UDP, so it also saves bandwidth. User Datagram Protocol (UDP) is more efficient in terms of both latency and bandwidth.

**Advantages of UDP:**

1. Transmission Delay

2. Multicast Applications

3. Connection Speed

4. Processing Time

5. Error Detection

**Disadvantages of UDP:**

1. Connectionless

2. Unreliable

3. Incorrect Order

4. No Congestion Control

5. Router Support

**Node:**

In Internet jargon, a computing device that connects to a network is called

a host or sometimes an end system. Because **ns-3** is a network simulator, not

specifically an Internet simulator, we intentionally do not use the term host

since it is closely associated with the Internet and its protocols. Instead, we use

a more generic term also used by other simulators that originates in Graph

Theory — the node.

In **ns-3** the basic computing device abstraction is called the node. This

abstraction is represented in C++ by the class Node. The Node class provides

methods for managing the representations of computing devices in simulations.

You should think of a Node as a computer to which you will add functionality.

One adds things like applications, protocol stacks and peripheral cards with

their associated drivers to enable the computer to do useful work. We use the

same basic model in ns-3.

The definition of a node depends on the network and protocol layer referred to.

A physical network node is an electronic device that is attached to a network,

and is capable of creating, receiving, or transmitting information over

a communication channel.

**NetDevice:**

It used to be the case that if you wanted to connect a computers to a network,

you had to buy a specific kind of network cable and a hardware device called (in

PC terminology) a peripheral card that needed to be installed in your computer.

If the peripheral card implemented some networking function, they were called

**Network Interface Cards, or NICs**. Today most computers come with the

network interface hardware built in and users don’t see these building blocks.

A NIC will not work without a software driver to control the hardware. In Unix

(or Linux), a piece of peripheral hardware is classified as a device. Devices are

controlled using device drivers, and network devices (NICs) are controlled

using network device drivers collectively known as net devices. In Unix and

Linux you refer to these net devices by names such as **eth0.**

In ns-3 the net device abstraction covers both the software driver and the

simulated hardware. A net device is “installed” in a Node in order to enable

the Node to communicate with other Nodes in the simulation via Channels. Just

as in a real computer, a Node may be connected to more than one Channel via

multiple NetDevices.

**Code:**

#include <fstream>

#include "ns3/core-module.h"

#include "ns3/csma-module.h"

#include "ns3/applications-module.h"

#include "ns3/internet-module.h"

#include "ns3/netanim-module.h"

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

using namespace ns3;

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

int

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

{

//

// Enable logging for UdpClient and

//

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

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

bool useV6 = false;

Address serverAddress;

CommandLine cmd (\_\_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 (2);

InternetStackHelper internet;

internet.Install (n);

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

//

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

//

CsmaHelper csma;

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

csma.SetChannelAttribute ("Delay", TimeValue (MilliSeconds (2)));

csma.SetDeviceAttribute ("Mtu", UintegerValue (1400));

NetDeviceContainer d = csma.Install (n);

//

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

//

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

if (useV6 == false)

{

Ipv4AddressHelper ipv4;

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

Ipv4InterfaceContainer i = ipv4.Assign (d);

serverAddress = Address (i.GetAddress (1));

}

else

{

Ipv6AddressHelper ipv6;

ipv6.SetBase ("2001:0000:f00d:cafe::", Ipv6Prefix (64));

Ipv6InterfaceContainer i6 = ipv6.Assign (d);

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

}

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

// Create one udpServer applications on node one.

//

uint16\_t port = 4000;

UdpServerHelper server (port);

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

apps.Start (Seconds (1.0));

apps.Stop (Seconds (10.0));

//

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

// node one.

//

uint32\_t MaxPacketSize = 1024;

Time interPacketInterval = Seconds (0.05);

uint32\_t maxPacketCount = 320;

UdpClientHelper client (serverAddress, port);

client.SetAttribute ("MaxPackets", UintegerValue (maxPacketCount));

client.SetAttribute ("Interval", TimeValue (interPacketInterval));

client.SetAttribute ("PacketSize", UintegerValue (MaxPacketSize));

apps = client.Install (n.Get (0));

apps.Start (Seconds (2.0));

apps.Stop (Seconds (10.0));

// Now, do the actual simulation.

AnimationInterface anim("udp.xml");

AnimationInterface::SetConstantPosition (n.Get(0), 10.0, 25.0);

AnimationInterface::SetConstantPosition (n.Get(1), 20.0, 25.0);

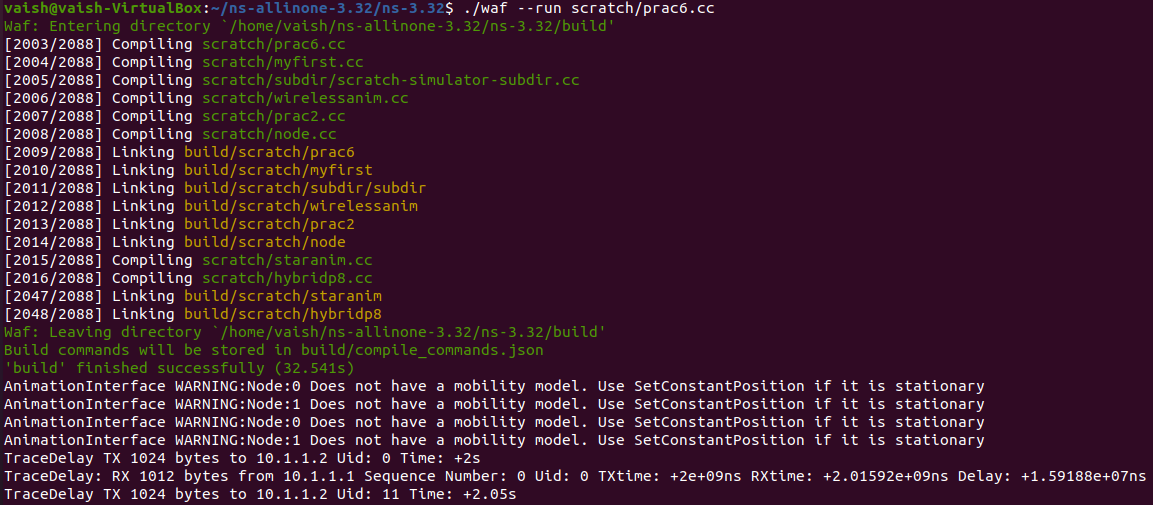
NS\_LOG\_INFO ("Run Simulation.");

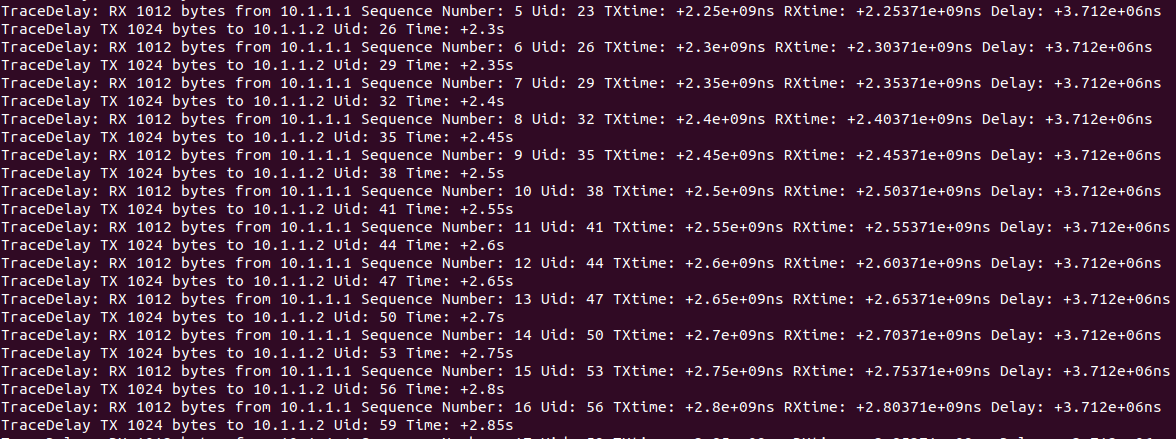
Simulator::Run ();

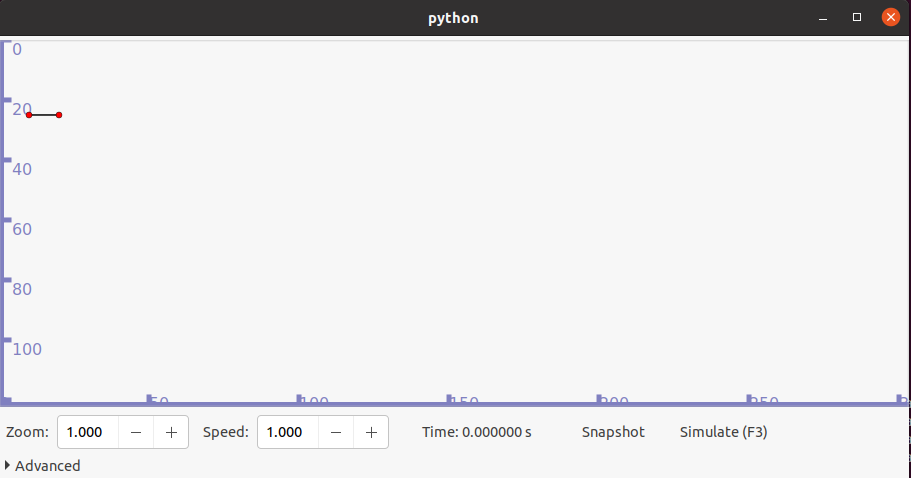
Simulator::Destroy ();

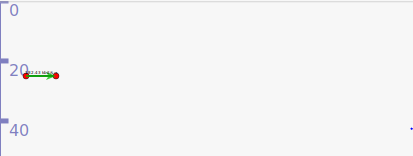
NS\_LOG\_INFO ("Done.");

}

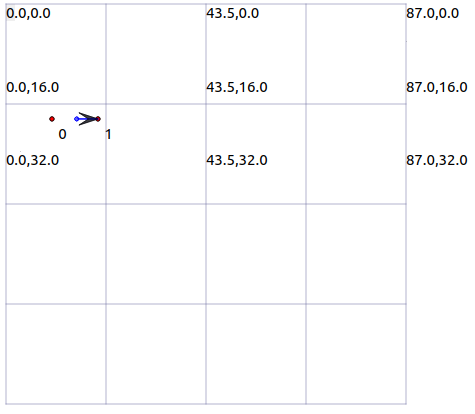








**Netanim**



**Conclusions:**

We have successfully implemented a program to simulate a UDP server client.