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

**THEORY:**

**Understanding UDP server client:**

In UDP, the client does not form a connection with the server like in TCP and instead just sends a datagram. Similarly, the server need not accept a connection and just waits for datagrams to arrive. Datagrams upon arrival contain the address of sender which the server uses to send data to the correct client.

In UDP, the client does not form a connection with the server like in TCP and instead, It just sends a datagram. Similarly, the server need not to accept a connection and just waits for datagrams to arrive. We can call a function called **connect()** in UDP but it does not result anything like it does in TCP. There is no 3-way handshake. It just checks for any immediate errors and store the peer’s IP address and port number. connect() is storing peers address so no need to pass **server address** and **server address length** arguments in **send to()**. The entire process can be broken down into following steps **UDP Server:**

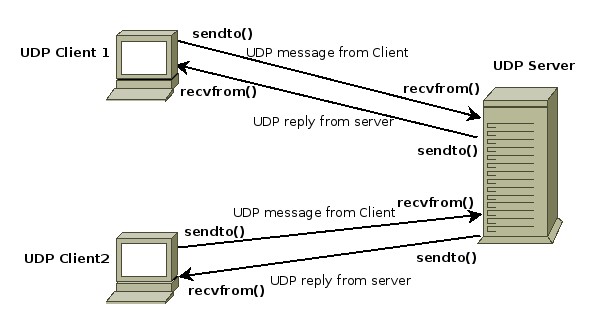
1. Create UDP socket.
2. Bind the socket to server address.
3. Wait until datagram packet arrives from client.
4. Process the datagram packet and send a reply to client.
5. Go back to Step 3.

**UDP Client:**

1. Create UDP socket.
2. Send message to server.
3. Wait until response from server is received.
4. Process reply and go back to step 2, if necessary.
5. Close socket descriptor and exit.

**UDP Overview:**

UDP is the abbreviation of User Datagram Protocol. UDP makes use of Internet Protocol of the TCP/IP suit. In communications using UDP, a client program sends a message packet to a destination server wherein the destination server also runs on UDP.



UDP the InterSystems IRIS User Datagram Protocol (UDP) binding. Provides two-way message transfer between a server and a large number of clients. UDP is not connection-based; each data packet transmission is an independent event. Provides fast and lightweight data transmission for local packet broadcasts and remote multicasting. Inherently less reliable than TCP. Does not provide message acknowledgement.

**Properties of UDP:**

* The UDP does not provide guaranteed delivery of message packets. If for some issue in a network if a packet is lost it could be lost forever.

Since there is no guarantee of assured delivery of messages, UDP is considered an unreliable protocol.

* The underlying mechanisms that implement UDP involve no connection-based communication. There is no streaming of data between a UDP server or and an UDP Client.
* An UDP client can send "n" number of distinct packets to an UDP server and it could also receive "n" number of distinct packets as replies from the UDP server.
* Since UDP is connectionless protocol the overhead involved in UDP is less compared to a connection based protocol like TCP.

**SOURCE CODE:**

// 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/netanim-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);

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));

csma.EnablePcapAll ("udp");

AnimationInterface anim("narenderUDP.xml");

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

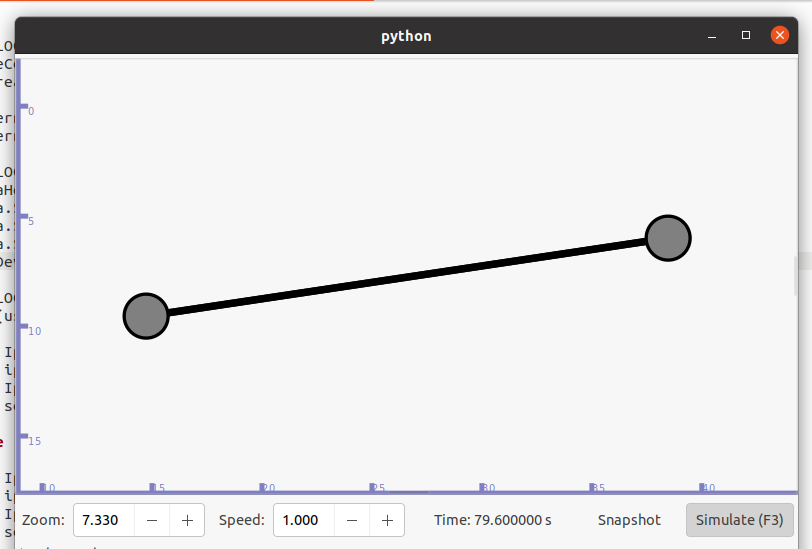
Simulator::Run ();

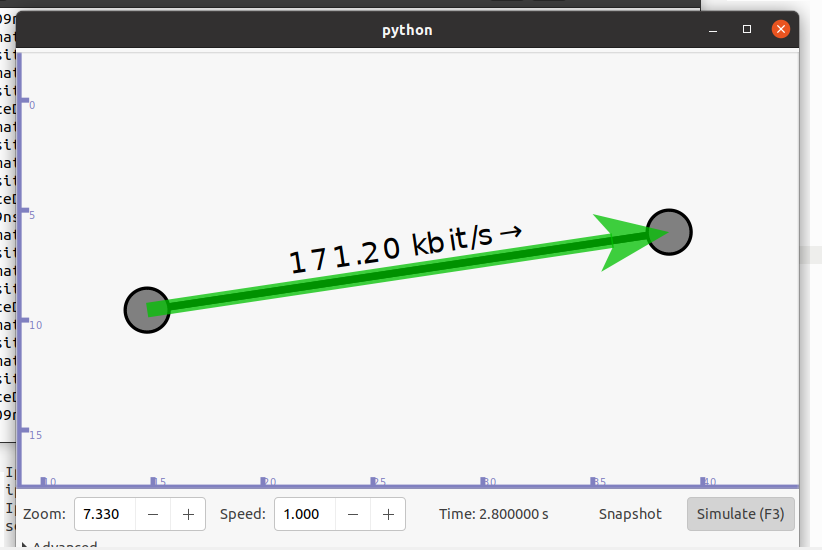
Simulator::Destroy ();

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

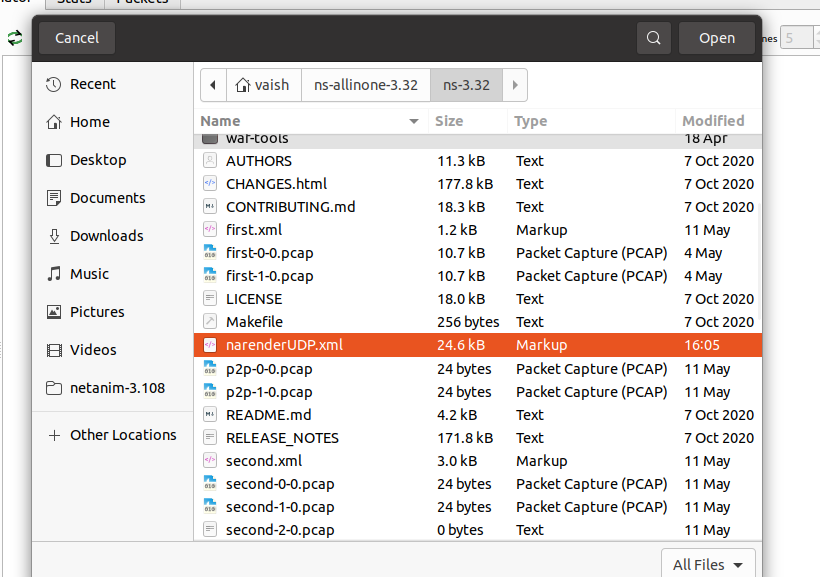
}

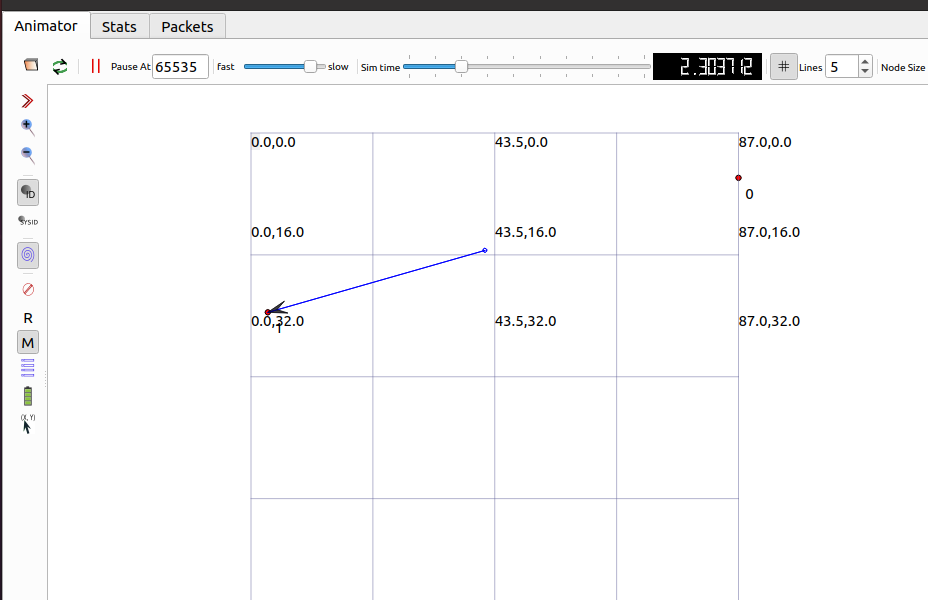
**OUTPUT:**



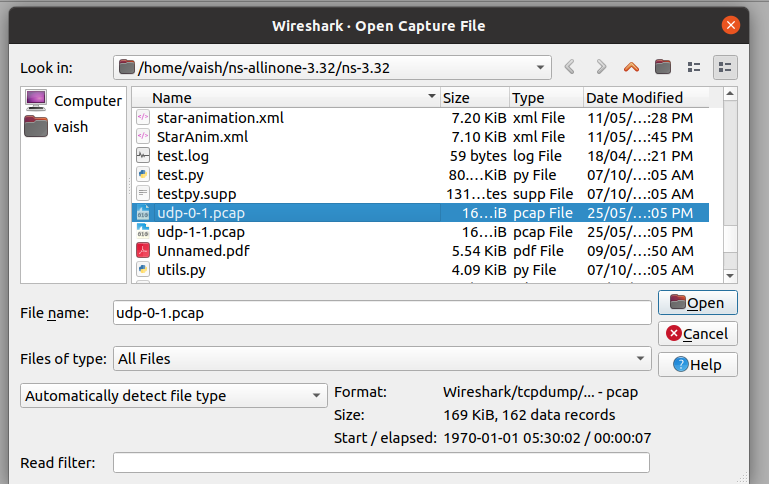


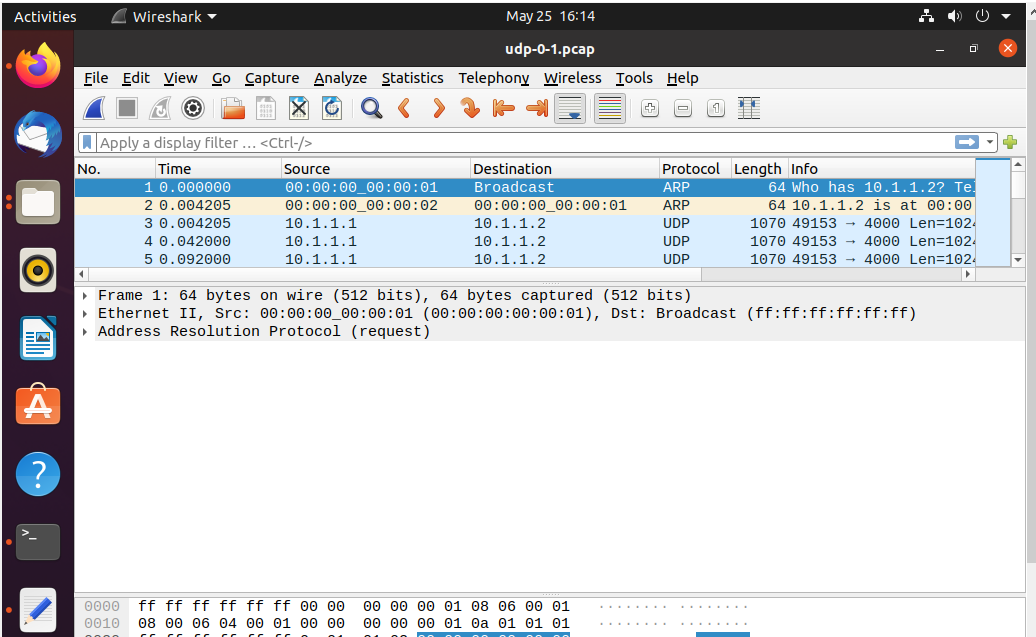
**NetAnim:**

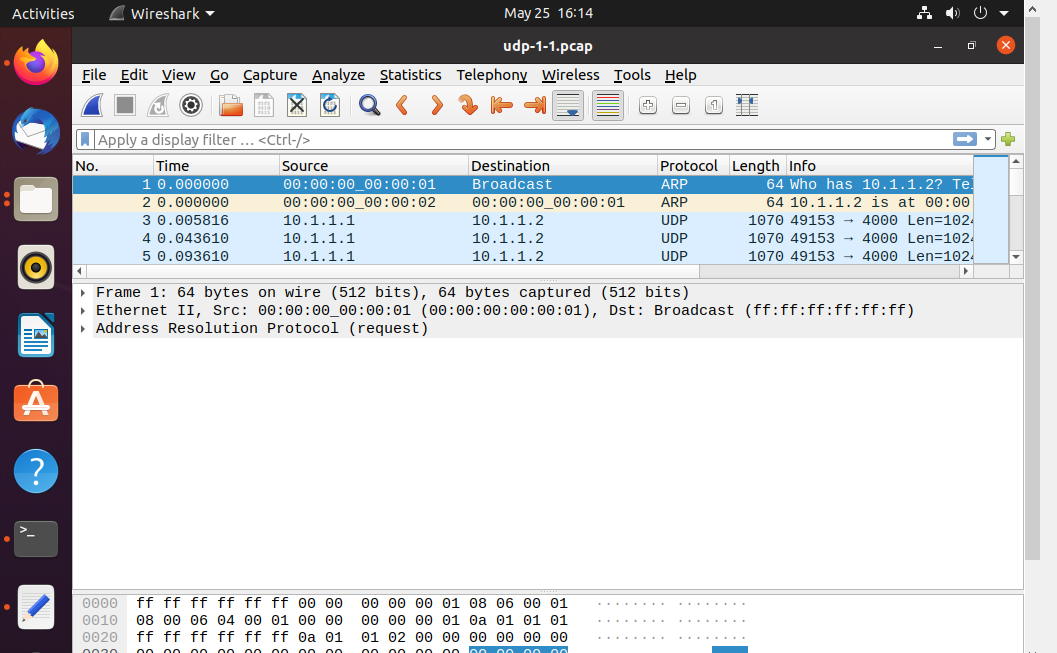




**WIRESHARK:**







**CONCLUSION:**

From this practical, I have successfully implemented the simulation of UDP server client.