**Practical No 12**

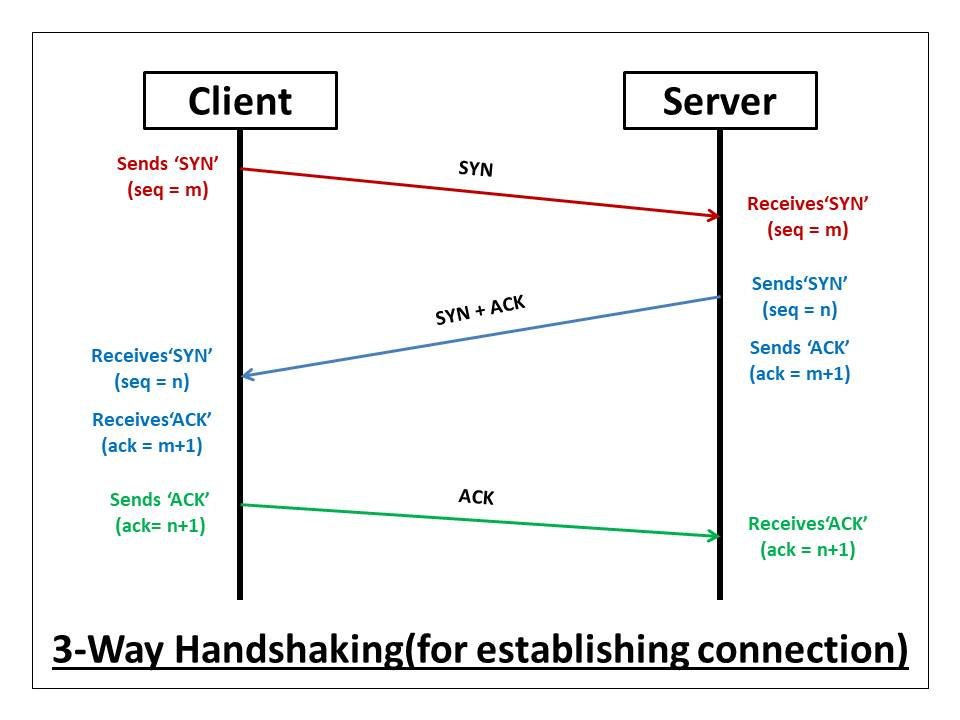
**Aim: -** Animate three-way handshake of TCP using NetAnim.

**Objective: -**

* Understanding of Three-way handshake of TCP
* Animation of Three-way handshake of TCP using NetAnim

**Theory: -**

TCP uses a three-way handshake to establish a reliable connection. The connection is full duplex, and both sides synchronize (SYN) and acknowledge (ACK) each other.



Steps **of** **Three-way handshake of TCP**:

1. SYN – The client establishes a connection with a server.
2. ACK+SYN – The server answers the client request.
3. Acknowledgement – Finally client confirms the server’s response and they two establish Secure connection.

**Advantages of Three-way handshake of TCP:**

* Reliable Connection Establishment: Ensures both parties agree on initial sequence numbers and connection parameters for reliable data transmission.
* SYN Flood Protection: Helps protect against SYN flood attacks by validating connection requests.
* Sequence Number Initialization: Initializes sequence numbers for tracking data segments and ensuring packet order.

**Disadvantages of Three-way handshake of TCP:**

* Latency: Introduces delay due to multiple round-trip exchanges before data transmission can begin.
* Resource Utilization: Requires network and system resources for each connection establishment, impacting scalability under high loads.
* Potential for Connection Failures: Adds complexity and potential points of failure in establishing connections, impacting overall reliability.

**Program: -**

#include <iostream>

#include <fstream>

#include <string>

#include <cassert>

#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/ipv4-global-routing-helper.h"

#include "ns3/netanim-module.h"

using namespace ns3;

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

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

//LogComponentEnable ("TcpServer", LOG\_LEVEL\_INFO);

//LogComponentEnable ("TcpL4Protocol", LOG\_LEVEL\_ALL);

//LogComponentEnable ("TcpSocketImpl", LOG\_LEVEL\_ALL);

//LogComponentEnable ("PacketSink", LOG\_LEVEL\_ALL);

// Set up some default values for the simulation.

Config::SetDefault ("ns3::OnOffApplication::PacketSize",

UintegerValue (250));

Config::SetDefault ("ns3::OnOffApplication::DataRate", StringValue

("5kb/s"));

uint32\_t N = 5; //number of nodes in the star

// Allow the user to override any of the defaults and the above

// Config::SetDefault()s at run-time, via command-line arguments

CommandLine cmd;

cmd.AddValue ("nNodes", "Number of nodes to place in the star", N);

cmd.Parse (argc, argv);

// Here, we will create N nodes in a star.

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

NodeContainer serverNode;

NodeContainer clientNodes;

serverNode.Create (1);

clientNodes.Create (N-1);

NodeContainer allNodes = NodeContainer (serverNode, clientNodes);

// Install network stacks on the nodes

InternetStackHelper internet;

internet.Install (allNodes);

//Collect an adjacency list of nodes for the p2p topology

std::vector<NodeContainer> nodeAdjacencyList (N-1);

for(uint32\_t i=0; i<nodeAdjacencyList.size (); ++i)

{

nodeAdjacencyList[i] = NodeContainer (serverNode, clientNodes.Get

(i));

}

// We create the channels first without any IP addressing information

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

PointToPointHelper p2p;

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

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

std::vector<NetDeviceContainer> deviceAdjacencyList (N-1);

for(uint32\_t i=0; i<deviceAdjacencyList.size (); ++i)

{

deviceAdjacencyList[i] = p2p.Install (nodeAdjacencyList[i]);

//Animating The Network

}

// Later, we add IP addresses.

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

Ipv4AddressHelper ipv4;

std::vector<Ipv4InterfaceContainer> interfaceAdjacencyList (N-1);

for(uint32\_t i=0; i<interfaceAdjacencyList.size (); ++i)

{

std::ostringstream subnet;

subnet<<"10.1."<<i+1<<".0";

ipv4.SetBase (subnet.str ().c\_str (), "255.255.255.0");

interfaceAdjacencyList[i] = ipv4.Assign (deviceAdjacencyList[i]);

}

//Turn on global static routing

Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

// Create a packet sink on the star "hub" to receive these packets

uint16\_t port = 50000;

Address sinkLocalAddress (InetSocketAddress (Ipv4Address::GetAny (), port));

PacketSinkHelper sinkHelper ("ns3::TcpSocketFactory",

sinkLocalAddress);

ApplicationContainer sinkApp = sinkHelper.Install (serverNode);

sinkApp.Start (Seconds (1.0));

sinkApp.Stop (Seconds (10.0));

// Create the OnOff applications to send TCP to the server

OnOffHelper clientHelper ("ns3::TcpSocketFactory", Address ());

clientHelper.SetAttribute ("OnTime", StringValue

("ns3::ConstantRandomVariable[Constant=1]"));

clientHelper.SetAttribute ("OffTime", StringValue

("ns3::ConstantRandomVariable[Constant=0]"));

//normally wouldn't need a loop here but the server IP address is different

//on each p2p subnet

ApplicationContainer clientApps;

for(uint32\_t i=0; i<clientNodes.GetN (); ++i)

{

AddressValue remoteAddress

(InetSocketAddress (interfaceAdjacencyList[i].GetAddress (0), port));

clientHelper.SetAttribute ("Remote", remoteAddress);

clientApps.Add (clientHelper.Install (clientNodes.Get (i)));

}

clientApps.Start (Seconds (1.0));

clientApps.Stop (Seconds (10.0));

//configure tracing

AsciiTraceHelper ascii;

p2p.EnableAsciiAll (ascii.CreateFileStream ("tcp-star-server.tr"));

p2p.EnablePcapAll ("tcp-star-server");

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

AnimationInterface anim ("handshake.xml");

anim.EnablePacketMetadata (true);

Simulator::Run ();

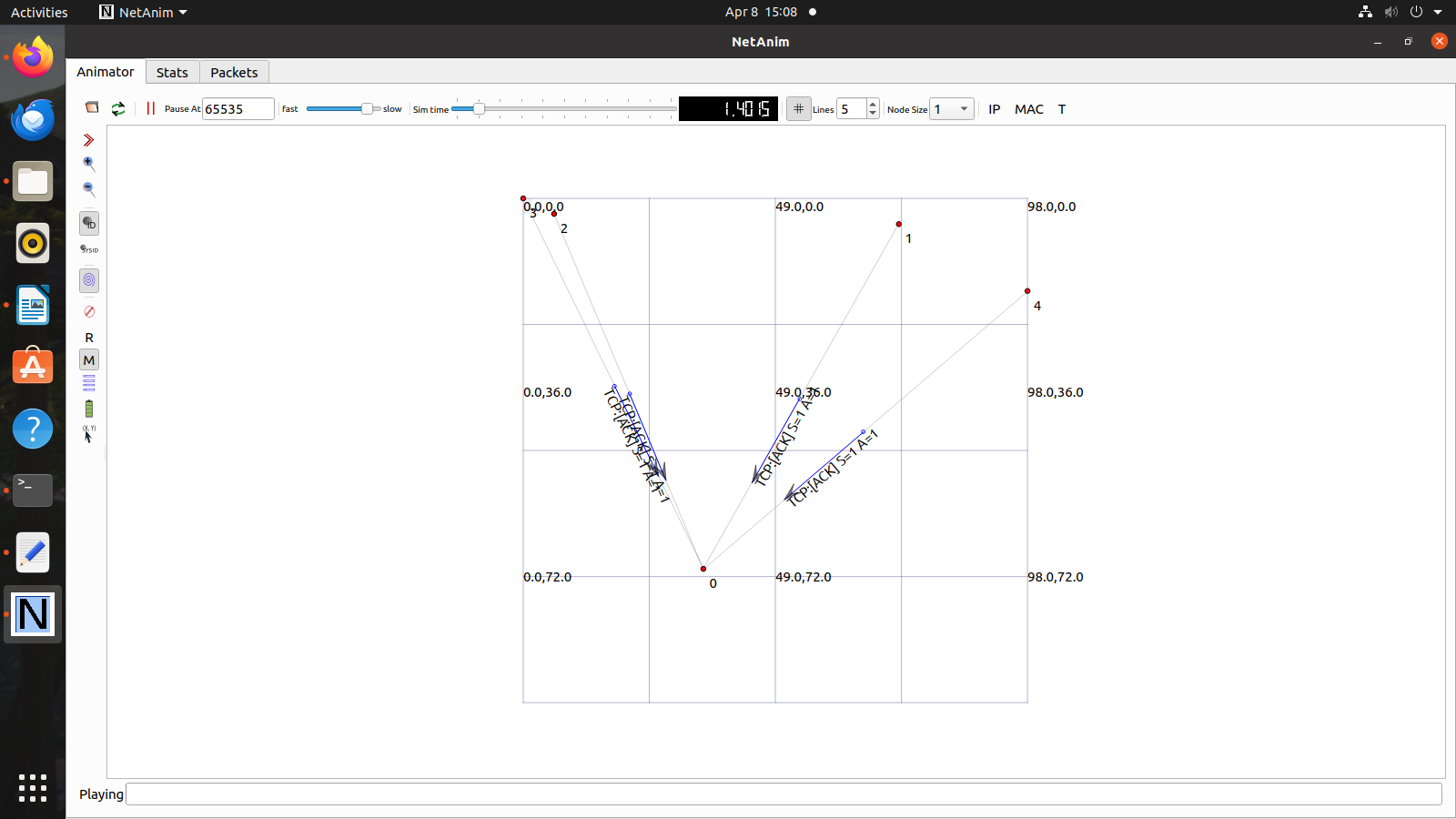
Simulator::Destroy ();

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

return 0;

}

**Output: -**



**Conclusion: -**

Successfully Animated Three-way handshake of TCP.