

COMPUTER NETWORKS LAB

ETCS 354

Submitted To:

Dr. Geetika Dhand

Submitted by:

Name: Aditya Gupta

Serial No: 37

Enrolment No: 35115002717

Class: CSE-2(B)



**Maharaja Surajmal Institute of Technology,
C-4 Janak Puri, New Delhi 110058**

INDEX

S.NO.	AIM	PAGE NUMBER
1	Introduction to Discrete Event Simulation tools NS2, NS3 and installation of Ns3.	2
2	Simulate point to point topology using NS3 (Using two nodes and three nodes).	10
3	Simulate the given architecture in NS3.	21
4	Simulate Bus Topology using NS3.	25
5	Create a Straight Cable and crossover cable using RS232C Connector.	29
6	Implementation and study of Stop and Wait Protocol.	32
7	Implementation and study of Go-back-n and Selective Repeat Protocol,	36
8	Implementation and study of Distance Vector Routing algorithm.	44
9	Implementation and study of Link state routing protocol.	50
10	Simulate Star Topology using TCP server in NS3.	56

PRACTICAL – 1

AIM: Introduction to Discrete Event Simulation tools NS2/NS3, OMNeT++ and installation of NS3.

1. OMNeT++?

OMNeT++ (Objective Modular Network Testbed in C++) is an extensible, modular, component based C++ simulation library and framework, primarily for building network simulators. “Network” is meant in a broader sense that includes wired and wireless communication networks, on-chip networks, queuing networks, and so on. Domain-specific functionality such as support for sensor networks, wireless ad-hoc networks, Internet protocols, performance modeling, photonic networks, etc., is provided by model frameworks, developed as independent projects. OMNeT++ offers an Eclipse-based IDE, a graphical runtime environment, and a host of other tools. There are extensions for real-time simulation, network emulation, database integration, SystemC integration, and several other functions. OMNeT++ is distributed under the Academic Public License.

Although OMNeT++ is not a network simulator itself, it has gained widespread popularity as a network simulation platform in the scientific community as well as in industrial settings, and building up a large user community.

OMNeT++ provides component architecture for models. Components (modules) are programmed in C++, and then assembled into larger components and models using a high-level language (NED). Reusability of models comes for free. OMNeT++ has extensive GUI support, and due to its modular architecture, the simulation kernel (and models) can be embedded easily into your applications.

1.1 Components

The main ingredients of OMNeT++ are:

- Simulation kernel library (C++)
- The NED topology description language
- Simulation IDE based on the Eclipse platform
- Interactive simulation runtime GUI (Qtenv)
- Command-line interface for simulation execution (Cmdenv)
- Utilities (make file creation tools, etc.) • Documentation, sample simulations, etc.

1.2 Models

During the years OMNeT++ has been available, countless simulation models and model frameworks have been written for it by researchers in diverse areas: queuing, resource modeling, internet protocols, wireless networks, switched LANs, peer-to-peer networks, media streaming, mobile ad-hoc networks, mesh networks, wireless sensor networks, vehicular networks, NoCs, optical networks, HPC systems, cloud computing, SANs, and more. Most of these model frameworks are open source, developed as independent projects, and follow their own release cycles.

The INET Framework can be considered the standard protocol model library of OMNeT++. INET contains models for the Internet stack and many other protocols and components. The INET Framework is maintained by the OMNeT++ team for the community, utilizing patches and new models contributed by members of the community. Several other simulation frameworks take INET as a base, and extend it into specific directions, such as vehicular networks (Veins, CoRE), overlay/peer-to-peer networks (OverSim), or LTE (SimuLTE).

1.3 Platforms

The OMNeT++ simulation kernel is standard C++, and runs basically on all platforms where a modern C++ compiler is available. The Simulation IDE requires Windows, Linux, or macOS.

2. Network Simulator -2(NS-2)

NS2 is an open-source simulation tool that runs on Linux. It is a discrete event simulator targeted at networking research and provides substantial support for simulation of routing, multicast protocols and IP protocols, such as UDP, TCP, RTP and SRM over wired and wireless (local and satellite) networks. It has many advantages that make it a useful tool, such as support for multiple protocols and the capability of graphically detailing network traffic. Additionally, NS2 supports several algorithms in routing and queuing. LAN routing and broadcasts are part of routing algorithms. Queuing algorithms include fair queuing, deficit round-robin and FIFO.

NS2 started as a variant of the REAL network simulator in 1989 (see Resources). REAL is a network simulator originally intended for studying the dynamic behavior of flow and congestion control schemes in packet-switched data networks.

NS2 is a tool that helps you better understand certain mechanisms in protocol definitions, such as congestion control, that are difficult to see in live testing. It provides good documentation and support for different add-ons. We recommend NS2 as a tool to help understand how protocols work and interact with different network topologies.

3. Network Simulator -3(NS-3)

NS-3 has been developed to provide an open, extensible network simulation platform, for networking research and education. In brief, ns-3 provides models of how packet data networks work and perform, and provides a simulation engine for users to conduct simulation experiments. Some of the reasons to use ns-3 include to perform studies that are more difficult or not possible to perform with real systems, to study system behavior in a highly controlled, reproducible environment, and to learn about how networks work. Users will note that the available model set in ns-3 focuses on modeling how Internet protocols and networks work, but ns-3 is not limited to Internet systems; several users are using ns-3 to model non-Internet-based systems.

Many simulation tools exist for network simulation studies. Below are a few distinguishing features of ns-3 in contrast to other tools.

1. NS-3 is designed as a set of libraries that can be combined together and also with other external software libraries. While some simulation platforms provide users with a single, integrated graphical user interface environment in which all tasks are carried out, ns-3 is more modular in this regard. Several external animators and data analysis and visualization tools can be used with ns-3. However, users should expect to work at the command line and with C++ and/or Python software development tools.
2. NS-3 is primarily used on Linux or macOS systems, although support exists for BSD systems and also for Windows frameworks that can build Linux code, such as Windows Subsystem for Linux, or Cygwin. Native Windows Visual Studio is not presently supported although a developer is working on future support. Windows users may also use a Linux virtual machine.
3. NS-3 is not an officially supported software product of any company. Support for ns-3 is done on a best-effort basis on the ns-3-users forum.

3.1 Installation of NS-3:-

(Operating System is Community Edition OS (CentOS))

- Download ns-allinone-3.2.4.tar.bz2
- Execute the following commands in the terminal

```
1. # tar -xvf<folder-name>
```

```
2. # cd ns allinone
```

```
3. # ./build.py
```

```
4. # ./waf configure --enable-tests
```

```
5. # ./waf --enable-examples
```

- · To run the program

Execute the command- # ./waf --run scratch/first

· To install the gcc compiler `sudo yum install gcc gcc-c++`

3.2 Commands used

- tar: The 'tar' stands for tape archive, is used to create Archive and extract the Archive files.
- o -xvf: This command extracts files from Archives.
- cd: cd command in linux known as change directory command. It is used to change current working directory.
- ./build.py: build ns-3 without any examples or tests.
- ./waf: You can use waf to enable/disable examples and tests once ns-3 has been built.
- yum: yum is an interactive, rpm based, package manager.
- o install: used to install the latest version of a package or group of packages while ensuring that all dependencies are satisfied.

Installation

Install following packages (command is given below):

➤ **For ns-3:**

1. gcc
2. g++
3. python
4. python-dev

➤ **For NetAnim:**

1. qt4-dev-tools

➤ **For PyViz:**

1. libgtk-3-dev
2. python-pygoocanvas
3. python-pygraphviz

➤ **For Wireshark and Gnuplot:**

1. wireshark
2. gnuplot

➤ **For TraceMetrics:**

1. openjdk-7-jdk

Command to install all these packages together:

```
yum install gcc g++ python python-dev qt4-dev-tools libgtk-3-dev python-pygoocanvas python-pygraphviz wireshark gnuplot openjdk-7-jdk
```

Steps to install Ns-3:

1. Download [ns-allinone-3.24.1.tar.bz2](#) and unzip it.
2. Go to ns-allinone-3.24.1 and give the following command:

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

This command will install ns-3, NetAnim and PyViz.

3. Once the installation completes, go to ns-allinone-3.24.1/ns-3.24.1 and give the following command:

```
./test.py -c core
```

DIFFERENCE BETWEEN NS2 AND NS3:

Application level Difference between NS2 and NS3:

NS3	NS2
<u>NS3</u> can be act as the emulator that it can connect to the real world	NS2 can not be act as the emulator.
Some of the NS2 models can be imported to NS3	NS3 scripts can not run in NS2 environment

Programming Language level Difference between Ns2 and NS3:

NS3	NS2
NS3 is written using C++	NS2 is written with the help of TCL and C++.
Compilation time is not a matter	C++ recompilation takes more time more than TCL , so most of the scripts are written using TCL
A Simulation script can be written in ns3	Simulation script is not possible with NS2
Python is available for the scripting language	Only TCL can be used as the scripting language

Packets difference in NS2 and NS3:

NS3	NS2
Information needed to send through the packet can be added at the header ,trailer, buffer ,etc	The header part of the NS2 includes all the information of header parts in the specified protocol
NS3 frees the memory that used to store the packets	NS2 never reuse or re allocate the memory until it get terminated.
The packet of ns3 consist of single buffer and	The packet of ns2 has headers and data for

small tags .	payload.
--------------	----------

File Format Difference between NS2 and NS3:

NS3	NS2
.tr-> files used for trace analysis	. tr-> files used for trace parameters
.XML->files are used for network Animation	.nam -> files used for Network Animation
.csv-> files used for gnu plot	.xg -> files used for graph

Visualization Difference between NS2 and NS3:

NS3	NS2
Python visualizer , Network Animator visualization is available	Nam animator is available for visualization.

Performance level difference between ns2 and ns3:

NS3	NS2
Memory allocation is good	Memory allocation is not good as NS3

CN LAB

System prevents unnecessary parameters to be stored.	Unnecessary parameters can not be prevented.
Total computation is less when compared to NS2	Total Computation time is high when compared to NS3

PRACTICAL - 2

AIM:

(a) To Simulate Point to Point topology using NS-3.

SOURCE CODE:

```
#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"


using namespace ns3;


NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");


int

main (int argc, char *argv[])

{

CommandLinecmd;

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));

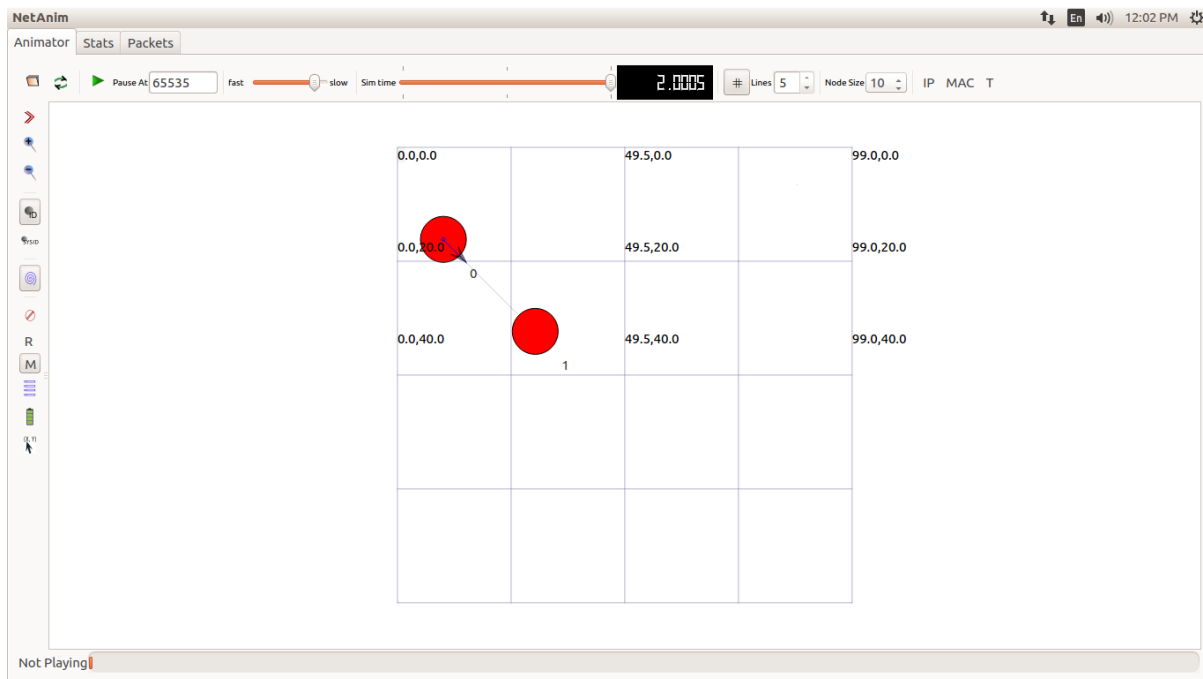
clientApps.Start (Seconds (2.0));

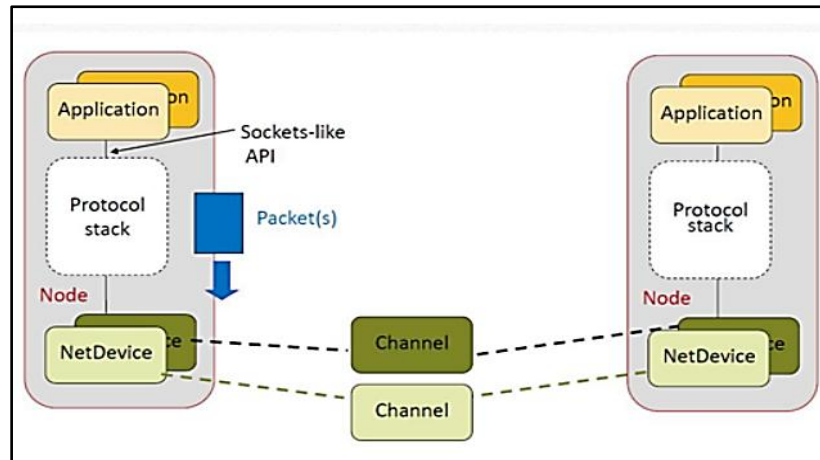
clientApps.Stop (Seconds (10.0));
```

CN LAB

```
AnimationInterface anim ("anim1.xml");  
  
anim.SetConstantPosition(nodes.Get (0), 1.0, 2.0);  
  
anim.SetConstantPosition(nodes.Get (1), 3.0, 4.0);  
  
Simulator::Run ();  
  
Simulator::Destroy ();  
  
return 0;  
  
}
```

Point to Point Network Simulation



Basic ns-3 model architecture**Classes used:**

- NodeContainer: The NodeContainer topology helper provides a convenient way to create, manage and access any Node objects that we create in order to run a simulation.
 - o The first line above just declares a NodeContainer which we call nodes.
 - o The second line calls the Create method on the nodes object and asks the container to create two nodes.
- PointToPointHelper: Build a set of PointToPointNetDevice objects.
 - o The first line instantiates a PointToPointHelper object on the stack.
 - o From a high-level perspective the next line, tells the PointToPointHelper object to use the value “5Mbps” (five megabits per second) as the “DataRate” when it creates a PointToPointNetDevice object
- NetDeviceContainer: Holds a vector of ns3::NetDevice pointers. After executing the pointToPoint.Install (nodes) call we will have two nodes, each with an installed point-to-point net device and a single point-to-point channel between them.
- InternetStackHelper: The InternetStackHelper is a topology helper that is to internet stacks what the PointToPointHelper is to point-to-point net devices. The Install method takes a NodeContainer as a parameter. When it is executed, it will install an Internet Stack (TCP, UDP, IP, etc.) on each of the nodes in the node container.
- Ipv4AddressHelper: Declares an address helper object and tell it that it should begin allocating IP addresses from the network 10.1.1.0 using the mask 255.255.255.0 to define

the allocatable bits. By default the addresses allocated will start at one and increase monotonically, so the first address allocated from this base will be 10.1.1.1, followed by 10.1.1.2, etc.

- Ipv4InterfaceContainer: Performs the actual address assignment.
- UdpEchoServerHelper: Similar to many other helper objects, the UdpEchoServerHelper object has an Install method. It is the execution of this method that actually causes the underlying echo server application to be instantiated and attached to a node. We now see that echoServer.Install is going to install a UdpEchoServerApplication on the node found at index number one of the NodeContainer we used to manage our nodes. Install will return a container that holds pointers to all of the applications (one in this case since we passed a NodeContainer containing one node) created by the helper. Applications require a time to “start” generating traffic and may take an optional time to “stop”.
- UdpEchoClientHelper: The echo client application is set up in a method substantially similar to that for the server. For the echo client, however, we need to set five different Attributes. The first two Attributes are set during construction of the UdpEchoClientHelper. We pass parameters that are used (internally to the helper) to set the “RemoteAddress” and “RemotePort” Attributes in accordance with our convention to make required Attributes parameters in the helper constructors.
- AnimationInterface: Interface to network animator. Provides functions that facilitate communications with an external or internal network animator.
- Simulator: When Simulator::Run is called, the system will begin looking through the list of scheduled events and executing them. First it will run the event at 1.0 seconds, which will enable the echo server application (this event may, in turn, schedule many other events). Then it will run the event scheduled for t=2.0 seconds which will start the echo client application. Eventually, since we only send one packet (recall the MaxPackets Attribute was set to one), the chain of events triggered by that single client echo request will taper off and the simulation will go idle. Once this happens, the remaining events will be the Stop events for the server and the client. When these events are executed, there are no further events to process and Simulator::Run returns. The simulation is then complete. All that remains is to clean up. This is done by calling the global function Simulator::Destroy.

(b) Simulate Point to Point network using three nodes.SOURCE CODE:

```
#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"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");

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

{

CommandLineCmd;

cmd.Parse (argc, argv);

Time::SetResolution (Time::NS);

LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);

LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);

NodeContainer nodes;

nodes.Create (3);

InternetStackHelper stack;

stack.Install (nodes);

PointToPointHelper pointToPoint;

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


CN LAB

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

PointToPointHelper pointToPoint1;

    pointToPoint1.SetDeviceAttribute ("DataRate",StringValue ("5Mbps"));

    pointToPoint1.SetChannelAttribute ("Delay",StringValue ("2ms"));

Ipv4AddressHelper address;

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

NetDeviceContainer devices;

    devices = pointToPoint.Install (nodes.Get(0),nodes.Get(1));

Ipv4InterfaceContainer interfaces = address.Assign (devices);

    devices = pointToPoint1.Install (nodes.Get(1),nodes.Get(2));

address.SetBase ("10.1.3.0", "255.255.255.0");

interfaces = address.Assign (devices);

Ipv4GlobalRoutingHelper::PopulateRoutingTables();

UdpEchoServerHelper echoServer (9);

ApplicationContainer serverApps = echoServer.Install (nodes.Get (2));

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)));

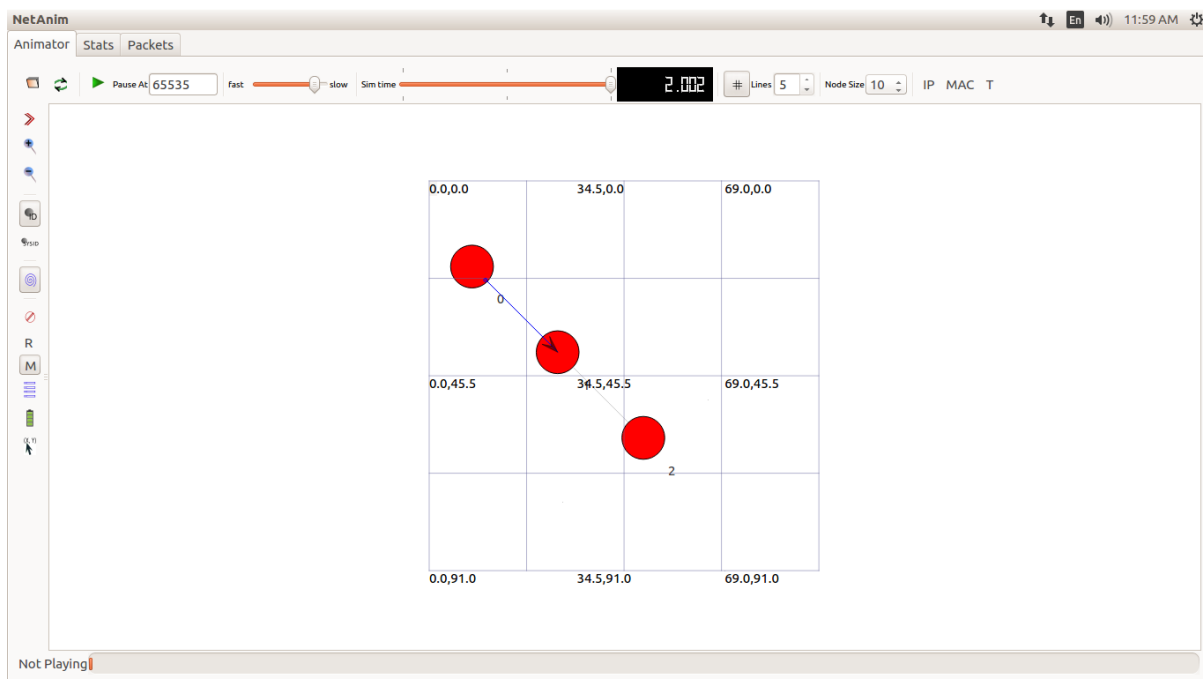
echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

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

clientApps.Start (Seconds (2.0));
```

```
clientApps.Stop (Seconds (10.0));  
  
AnimationInterface anim("anim1.xml");  
  
anim.SetConstantPosition (nodes.Get(0),10,20);  
  
anim.SetConstantPosition (nodes.Get(1),30,40);  
  
anim.SetConstantPosition (nodes.Get(2),50,60);  
  
Simulator::Run ();  
  
Simulator::Destroy ();  
  
return 0;  
  
}
```

Simulation Output



(c) Simulate Point to Point network using Four nodes.SOURCE CODE:

```

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

namespace ns3;

NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");

int main (int argc, char *argv[]) { CommandLine cmd; cmd.Parse (argc, argv);

Time::SetResolution (Time::NS);
LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO); LogComponentEnable
("UdpEchoServerApplication", LOG_LEVEL_INFO);

NodeContainer nodes;
nodes.Create (4);

PointToPointHelper pointToPoint1,pointToPoint2,pointToPoint3;

pointToPoint1.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
pointToPoint1.SetChannelAttribute ("Delay", StringValue ("2ms"));
pointToPoint2.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
pointToPoint2.SetChannelAttribute ("Delay", StringValue ("2ms"));
pointToPoint3.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
pointToPoint3.SetChannelAttribute ("Delay", StringValue ("2ms"));

InternetStackHelper stack;
stack.Install (nodes);

NetDeviceContainer devices;
devices = pointToPoint1.Install (nodes.Get(0),nodes.Get(1));
Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer interfaces = address.Assign (devices);
//interfaces = address.Assign (devices);

devices = pointToPoint2.Install (nodes.Get(1),nodes.Get(2));
//Ipv4AddressHelper address;
address.SetBase ("10.1.2.0", "255.255.255.0");
//Ipv4InterfaceContainer interfaces = address.Assign (devices);
interfaces = address.Assign (devices);

```

CN LAB

```
devices = pointToPoint3.Install (nodes.Get(2),nodes.Get(3));
//Ipv4AddressHelper address;
address.SetBase ("10.1.3.0", "255.255.255.0");
//Ipv4InterfaceContainer interfaces = address.Assign (devices);
interfaces = address.Assign (devices);
Ipv4GlobalRoutingHelper::PopulateRoutingTables();

UdpEchoServerHelper echoServer (9);

ApplicationContainer serverApps = echoServer.Install (nodes.Get (3));
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));

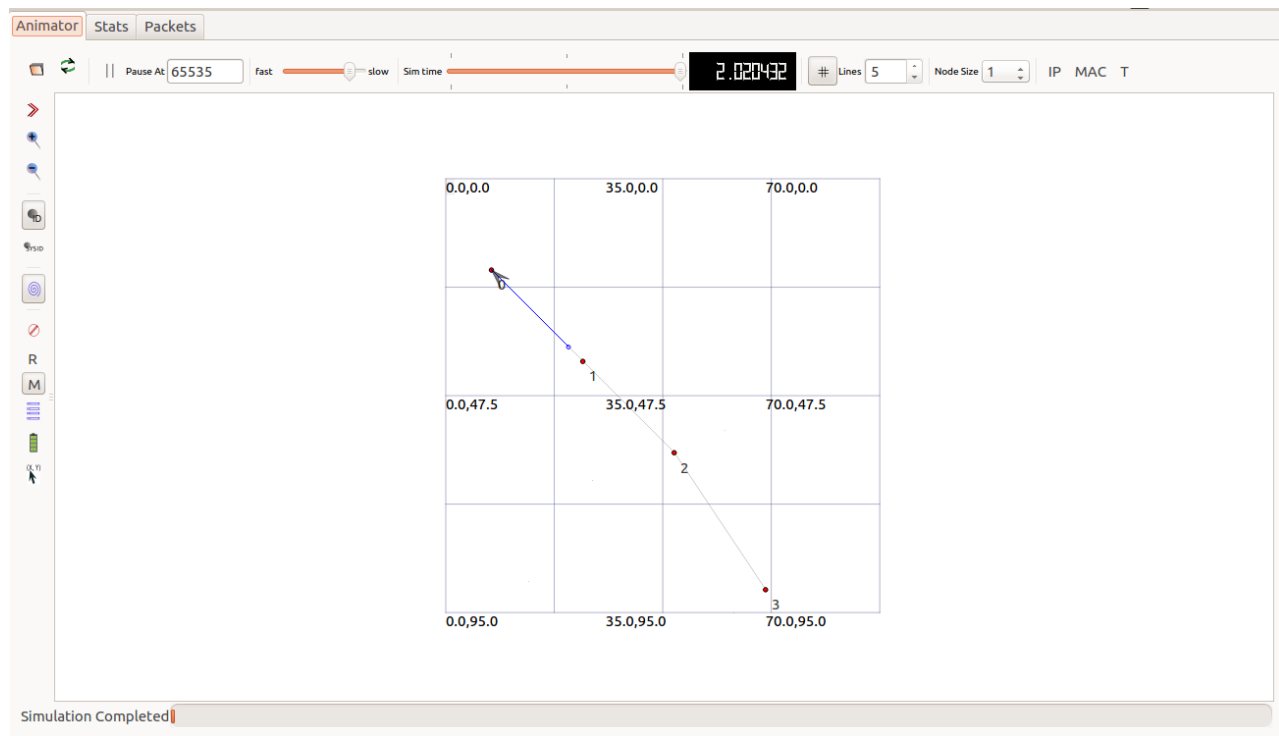
AnimationInterface anim("first.xml");
anim.SetConstantPosition(nodes.Get(0),10,20);
anim.SetConstantPosition(nodes.Get(1),30,40);
anim.SetConstantPosition(nodes.Get(2),50,60);
anim.SetConstantPosition(nodes.Get(3),70,90);

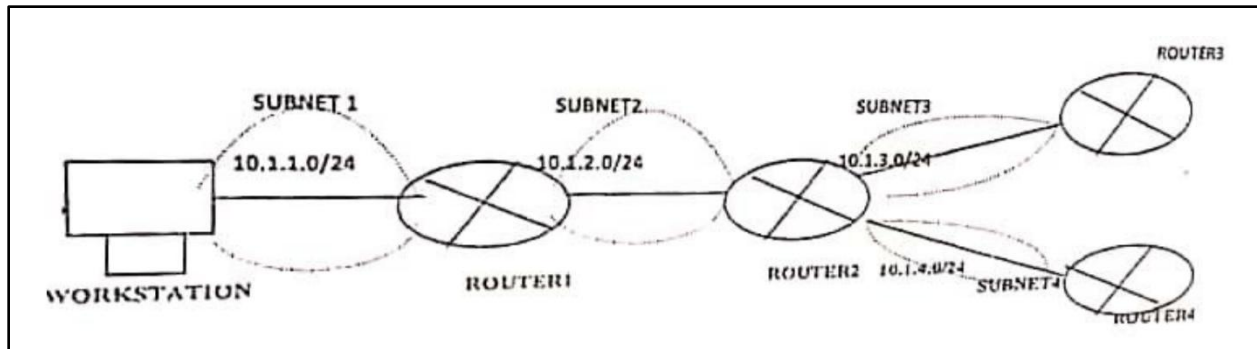
Simulator::Run ();
Simulator::Destroy ();
return 0;

}
```

CN LAB

Simulation Output



PRACTICAL - 3**AIM:** Simulate the given architecture in Ns3.**CODE:**

```

#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"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");

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

    cmd.Parse (argc, argv);

    Time::SetResolution (Time::NS);

    LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
  
```

CN LAB

```
LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
```

```
NodeContainer hosts;
```

```
NodeContainer routers;
```

```
hosts.Create(1);
```

```
routers.Create(4);
```

```
InternetStackHelper stack;
```

```
Ipv4AddressHelper address;
```

```
stack.Install(routers);
```

```
stack.Install(hosts);
```

```
PointToPointHelper p1, p2, p3, p4;
```

```
NodeContainer subnet1;
```

```
subnet1.Add(hosts.Get(0));
```

```
subnet1.Add(routers.Get(0));
```

```
NetDeviceContainer subnet1devices=p1.Install(subnet1);
```

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

```
Ipv4InterfaceContainer subnet1interfaces=address.Assign(subnet1devices);
```

```
NodeContainer subnet2;
```

```
subnet2.Add(routers.Get(0));
```

```
subnet2.Add(routers.Get(1));
```

```
NetDeviceContainer subnet2devices=p2.Install(subnet2);
```

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

```
Ipv4InterfaceContainer subnet2interfaces=address.Assign(subnet2devices);
```

```
NodeContainer subnet3;
```

CN LAB

```
subnet3.Add(routers.Get(1));

subnet3.Add(routers.Get(2));

NetDeviceContainer subnet3devices=p3.Install(subnet3);

address.SetBase("10.1.3.0", "255.255.255.0");


Ipv4InterfaceContainer subnet3interfaces=address.Assign(subnet3devices);

NodeContainer subnet4;

subnet4.Add(routers.Get(1));

subnet4.Add(routers.Get(3));

NetDeviceContainer subnet4devices=p4.Install(subnet4);

address.SetBase("10.1.4.0", "255.255.255.0");

Ipv4InterfaceContainer subnet4interfaces=address.Assign(subnet4devices);

Ipv4GlobalRoutingHelper::PopulateRoutingTables();

UdpEchoServerHelper echoServer (9);

ApplicationContainer serverApps = echoServer.Install (routers.Get (3));

serverApps.Start (Seconds (1.0));

serverApps.Stop (Seconds (10.0));


UdpEchoClientHelper echoClient (subnet4interfaces.GetAddress (1), 9);

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

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

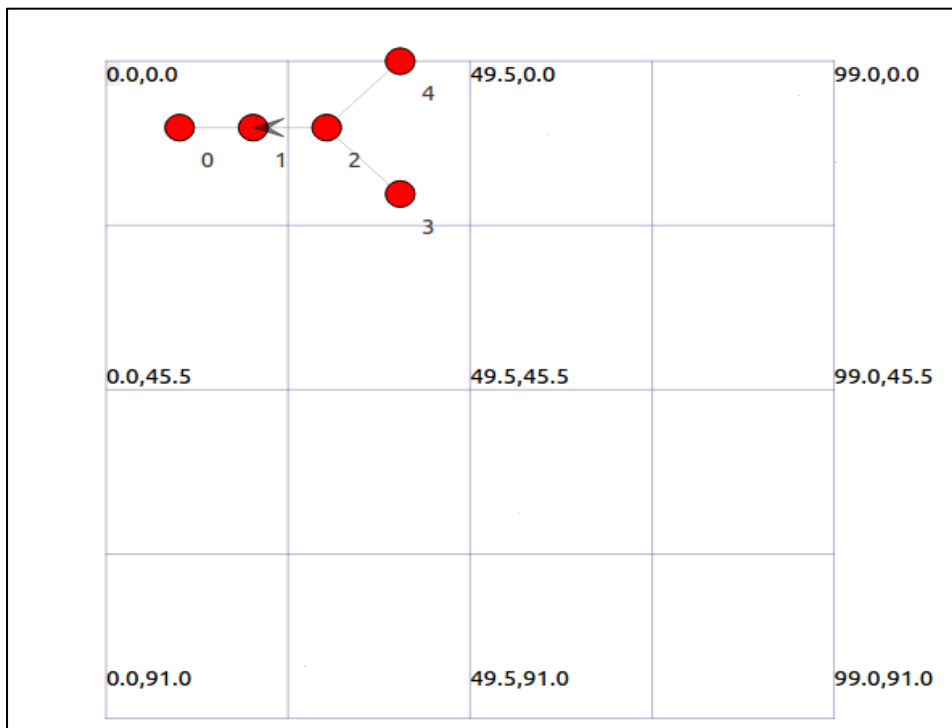
echoClient.SetAttribute ("PacketSize", UIntegerValue (1024));

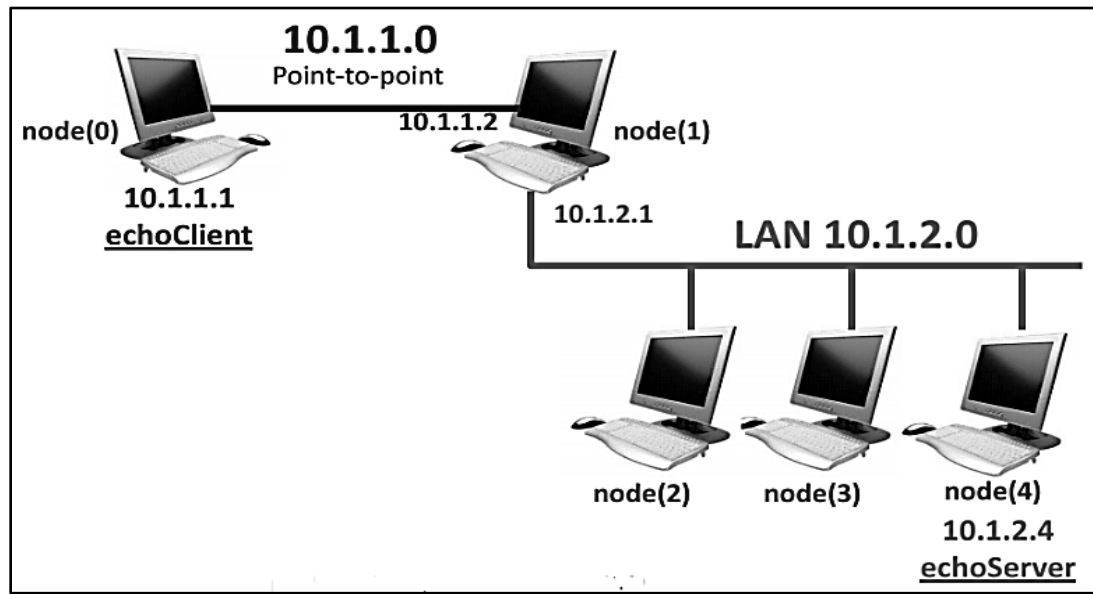

ApplicationContainer clientApps = echoClient.Install (hosts.Get (0));

clientApps.Start (Seconds (2.0));
```



```
clientApps.Stop (Seconds (10.0));  
  
AnimationInterfaceanim("r.xml");  
  
anim.SetConstantPosition(hosts.Get(0),10,10);  
  
anim.SetConstantPosition(routers.Get(0),20,10);  
  
anim.SetConstantPosition(routers.Get(1),30,10);  
  
anim.SetConstantPosition(routers.Get(2),40,20);  
  
anim.SetConstantPosition(routers.Get(3),40,0);  
  
  
Simulator::Run ();  
  
Simulator::Destroy ();  
  
return 0;  
  
}
```

OUTPUT:

PRACTICAL - 4**AIM: Simulate Bus topology between different Nodes using Ns3.****SOURCE CODE:**

```
#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"
```

```
// Default Network Topology
```

```
//
```

```
// 10.1.1.0
```

```
// n0 ----- n1  n2  n3  n4
```

```
// point-to-point |  |  |  |
```

```
//
```

```
// =====
```

```
// LAN 10.1.2.0
```

```

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("SecondScriptExample");

int
main (int argc, char *argv[])
{
    bool verbose = true;
    uint32_t nCsmas = 3;

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

    nCsmas = nCsmas == 0 ? 1 : nCsmas;

    NodeContainer p2pNodes;
    p2pNodes.Create (2);

    NodeContainer csmaNodes;
    csmaNodes.Add (p2pNodes.Get (1));
    csmaNodes.Create (nCsmas);

    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"));

```

CN LAB

```
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 (nCsmas));  
serverApps.Start (Seconds (1.0));  
serverApps.Stop (Seconds (10.0));
```

```
UdpEchoClientHelper echoClient (csmaInterfaces.GetAddress (nCsmas), 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);
```

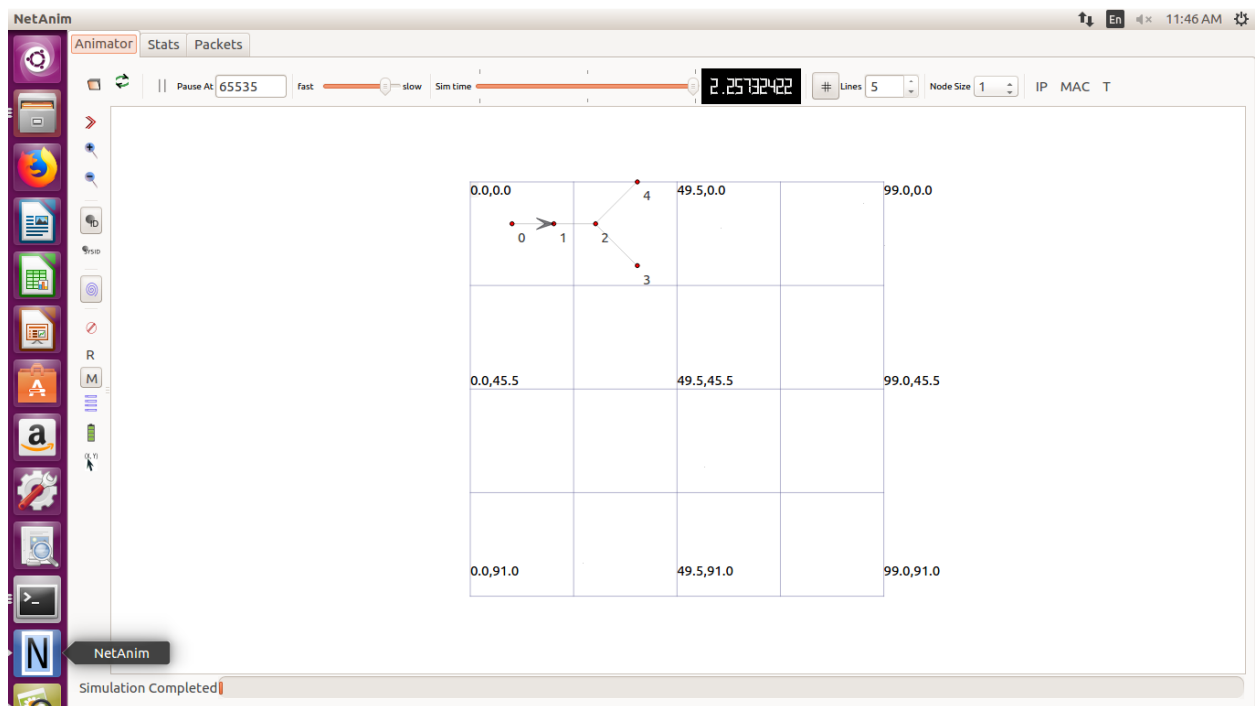
```
AnimationInterface anim ("animX.xml");  
anim.SetConstantPosition (p2pNodes.Get (0), 1.0, 2.0);
```

CN LAB

```
anim.SetConstantPosition(csmaNodes.Get(0),2.0,3.0);  
anim.SetConstantPosition(csmaNodes.Get(1),3.0,3.0);  
anim.SetConstantPosition(csmaNodes.Get(2),4.0,3.0);  
anim.SetConstantPosition(csmaNodes.Get(3),5.0,3.0);
```

```
Simulator::Run ();  
Simulator::Destroy ();  
return 0;  
}
```

OUTPUT:



PRACTICAL - 5**AIM: Create a Straight Cable and crossover cable using RS232C Connector.**

Serial communications with RS232 is one of the oldest and most widely spread communication methods in the computer world. The way this type of communication can be performed is pretty well defined in standards. I.e. with one exception. The standards show the use of DTE/DCE communication, the way a computer should communicate with a peripheral device like a modem. For your information, DTE means data terminal equipment (computers etc.) where DCE is the abbreviation of data communication equipment (modems). One of the main uses of serial communication today where no modem is involved—a serial null modem configuration with DTE/DTE communication—is not so well defined, especially when it comes to flow control. The terminology null modem for the situation where two computers communicate directly is so often used nowadays, that most people don't realize anymore the origin of the phrase and that a null modem connection is an exception, not the rule.

The standard 9 pins RS232 connector has 9 pins. Those are:

1. DCD -- Data Carrier Detected
2. RxD -- Receive Data
3. TxD -- Transmit Data
4. DTR -- Data Terminal Ready
5. GND -- Signal Ground
6. DSR -- Data Set Ready
7. RTS -- Request to Send Data
8. CTS -- Clear to Send Data
9. RI -- Ring Indicator

If you need only unidirectional connection then connect only RxD or TxD and If you need bi-directional communication it is obvious that I should connect RxD and TxD.

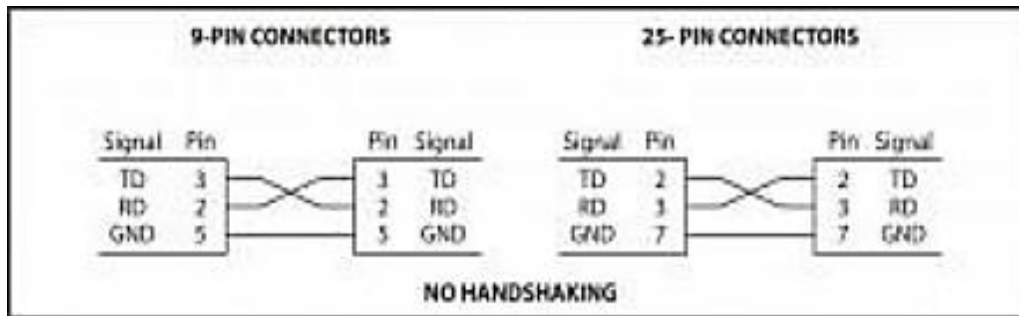
When two pieces of Data Terminal Equipment (terminals, or DTEs) need to be connected to each other, or two pieces of Data Circuit-terminating Equipment (modems, or DCEs) need to be connected to each other, a serial crossover or null modem cable is used. These cables align incoming signals with the corresponding outgoing signals on the other side of the connection. At a minimum, the Transmit Data (TD) line of a serial link needs to be paired with the corresponding Receive Data (RD) line on the other device.

Why would one need such a cable?

- It can be a way of connecting two PCs without networking interfaces such as Ethernet, making direct file transfers between systems possible.
- It can be a method for technically-minded users to debug systems with minimal software overhead (that is, as little code and as few drivers as possible).
- It can provide access to a serial console when problems make the local monitor and keyboard of a computer unavailable, or when a computer is being remotely operated or operated “headless” (that is, without a monitor, keyboard and mouse).
- Not all null modem cables are the same however, and some cables will not work in specific applications. The three standard types of null modem cables are described here, for DB-9 to DB-9 and DB-25 to DB-25 connections, making six variations. I haven't shown the three DB-9 to DB-25 versions, which in any case are just the DB-9 side at one end and the DB-25 side at the other

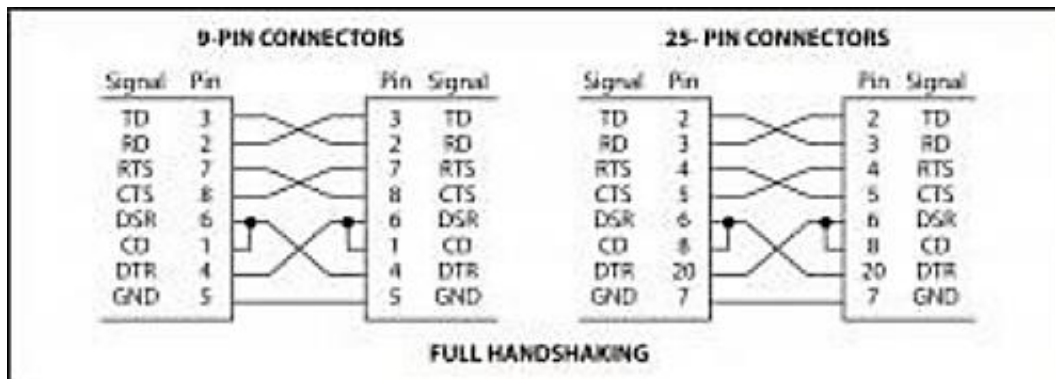
(see below, “9-Pin to 25-Pin Connection”). The bottom line is: there are nine variants of RS-232 crossover cables.

- The first, “No Handshaking,” swaps the complementary data transmit and receive lines. This is the minimum needed for a crossover connection. This cable will work as a crossover cable when control lines are not needed for the link. It cannot be used when hardware handshaking is required. It can be used when hardware flow control has been turned off on the serial ports involved, but doing so will simply bypass the control lines regardless of their state.



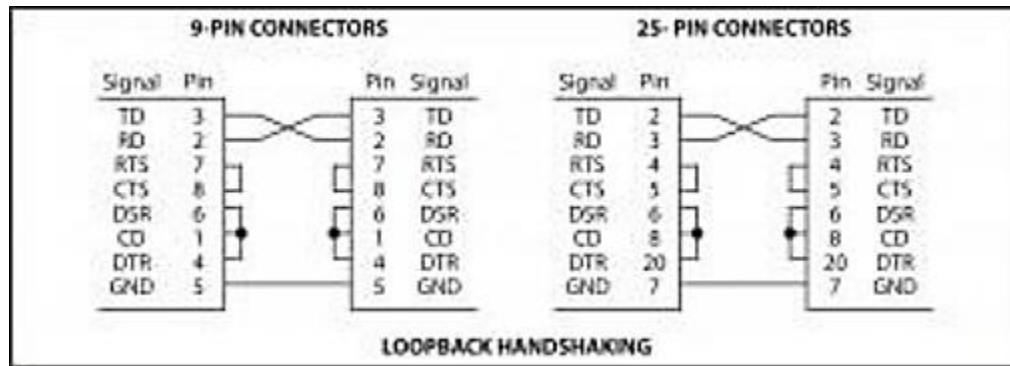
Crossover cable with no handshaking

- The second, “Full Handshaking,” swaps the data lines as well as the control lines needed for handshaking/flow control. The pairs of lines needed for handshaking are DTR/DSR and RTS/CTS. This cable crosses these pairs between the two ends of the cable. This cable will work as a crossover cable when control lines are not needed for the link, or when hardware handshaking is required. It can be used when hardware flow control has been turned off on the serial ports involved, but doing so will simply bypass the control lines regardless of their state.



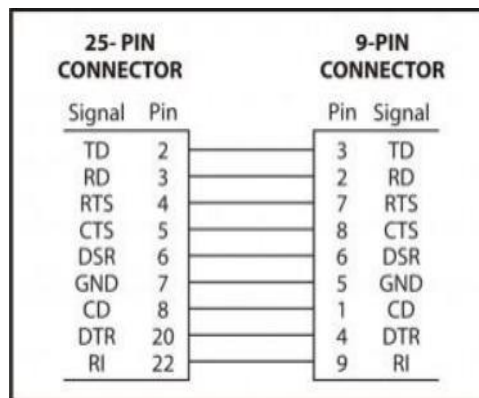
Crossover cable with full handshaking

- The third, “Loopback Handshaking,” swaps the data transmit and receive lines, but the DTR/DSR and RTS/CTS pairs are not swapped with their complements. Instead, the DTR line on one side of the link is looped back to the DSR line on the same side of the link, and the RTS line on one side of the link is looped back to the CTS on the same side of the link. This cable will work as a crossover cable when control lines are not needed for the link. It can be used regardless of the on/off state of hardware flow control on the serial ports involved, but will “fool” the link into accepting that hardware handshaking has been completed. It cannot be used when true hardware handshaking is required.



Crossover cable with loopback handshaking.

- If one side of a serial crossover cable uses a 9-pin connector and the other side uses a 25-pin connector, the lines map between these two connectors as follows:



9 pin to 25 pin connection

PRACTICAL - 6

AIM: Implementation and study of stop and wait protocol.

```
import java.util.*;

import java.io.*;

import java.net.Socket;

public class client{

    public static void main(String[] args) throws Exception{

        Scanner scn = new Scanner(System.in);

        Socket sc = new Socket("127.0.0.1",1234);

        Scanner p = new Scanner(sc.getInputStream());

        PrintStream ps = new PrintStream(sc.getOutputStream());

        System.out.println("Enter no of frames you want to send : ");

        int nf = scn.nextInt();

        String[] frames = new String[nf];

        for(int i = 0; i < nf;i++){

            System.out.println("Enter data of " + i + "th frame : ");

            frames[i] = scn.next();

        }

        System.out.println();

        for(int i = 0; i < nf;i++){

            ps.println(frames[i]);

            System.out.println(i + "th Frame send !!!" + " Time : " +

                System.currentTimeMillis());
```

```
String ack = p.next();

if(ack.equals("received")){

    System.out.println("Acknowledgement from reciever : " + ack + " Time : " +

                        System.currentTimeMillis() + "\n");

    Thread.sleep(2000);

}

}

ps.println("exit");

}

}
```

Receiver:

```
import java.util.*;

import java.net.Socket;

import java.io.*;

import java.net.ServerSocket;

public class server{

    public static void main(String[] args)throws Exception {

        ServerSocket ss = new ServerSocket(1234);

        System.out.println("Server Started");

        Socket s = ss.accept();

        Scanner reader = new Scanner(s.getInputStream());

        PrintStream printer = new PrintStream(s.getOutputStream());
```

```
String frame = "read";

String status = "exit";

while(frame.compareTo(status)!=0){

    frame = reader.next();

    if(frame.compareTo(status) == 0){

        System.out.println("All frames are received !!");

        break;

    }

    System.out.println("Message : " + frame + " Time : " +

                        System.currentTimeMillis());

    printer.println("received");

    System.out.println("Acknowledgment send !\n");

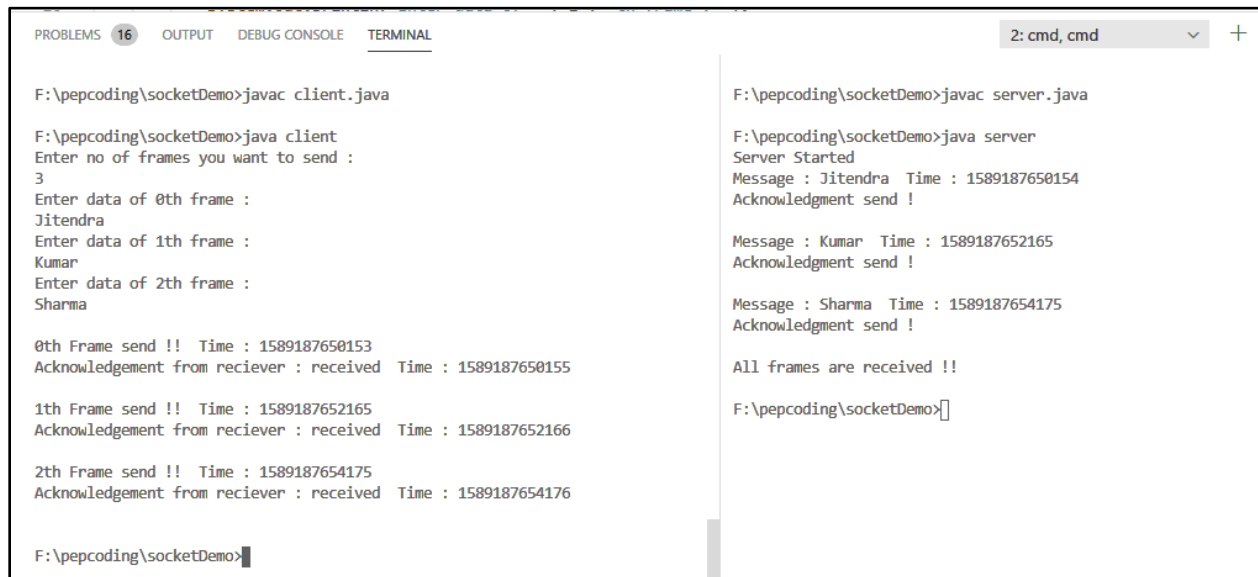
    Thread.sleep(1000);

}

}

}
```

OUTPUT:



```
PROBLEMS 16 OUTPUT DEBUG CONSOLE TERMINAL 2: cmd, cmd +

F:\pepcoding\socketDemo>javac client.java

F:\pepcoding\socketDemo>java client
Enter no of frames you want to send :
3
Enter data of 0th frame :
Jitendra
Enter data of 1th frame :
Kumar
Enter data of 2th frame :
Sharma

0th Frame send !! Time : 1589187650153
Acknowledgement from reciever : received Time : 1589187650155

1th Frame send !! Time : 1589187652165
Acknowledgement from reciever : received Time : 1589187652166

2th Frame send !! Time : 1589187654175
Acknowledgement from reciever : received Time : 1589187654176

F:\pepcoding\socketDemo>

F:\pepcoding\socketDemo>javac server.java

F:\pepcoding\socketDemo>java server
Server Started
Message : Jitendra Time : 1589187650154
Acknowledgment send !

Message : Kumar Time : 1589187652165
Acknowledgment send !

Message : Sharma Time : 1589187654175
Acknowledgment send !

All frames are received !!

F:\pepcoding\socketDemo>
```

PRACTICAL - 7

AIM: Implementation and study of Go-back-N and selective repeat protocol.

SLIDING WINDOW PROTOCOL (GO BACK N)

SOURCE CODE:

```
import java.io.*;

public class pract{

    public static void main(String args[]) throws IOException

    {

        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

        System.out.println("Please enter the Window Size: ");

        int window = Integer.parseInt(br.readLine());

        boolean loop = true;

        int sent = 0;

        while(loop)

        {

            for(int i = 0; i < window; i++)

            {

                System.out.println("Frame " + sent + " has been transmitted.");

                sent++;

                if(sent == window)

                    break;

            }

            System.out.println("Please enter the last Acknowledgement received.");
```

```
int ack = Integer.parseInt(br.readLine());

if(ack == window)

    loop = false;

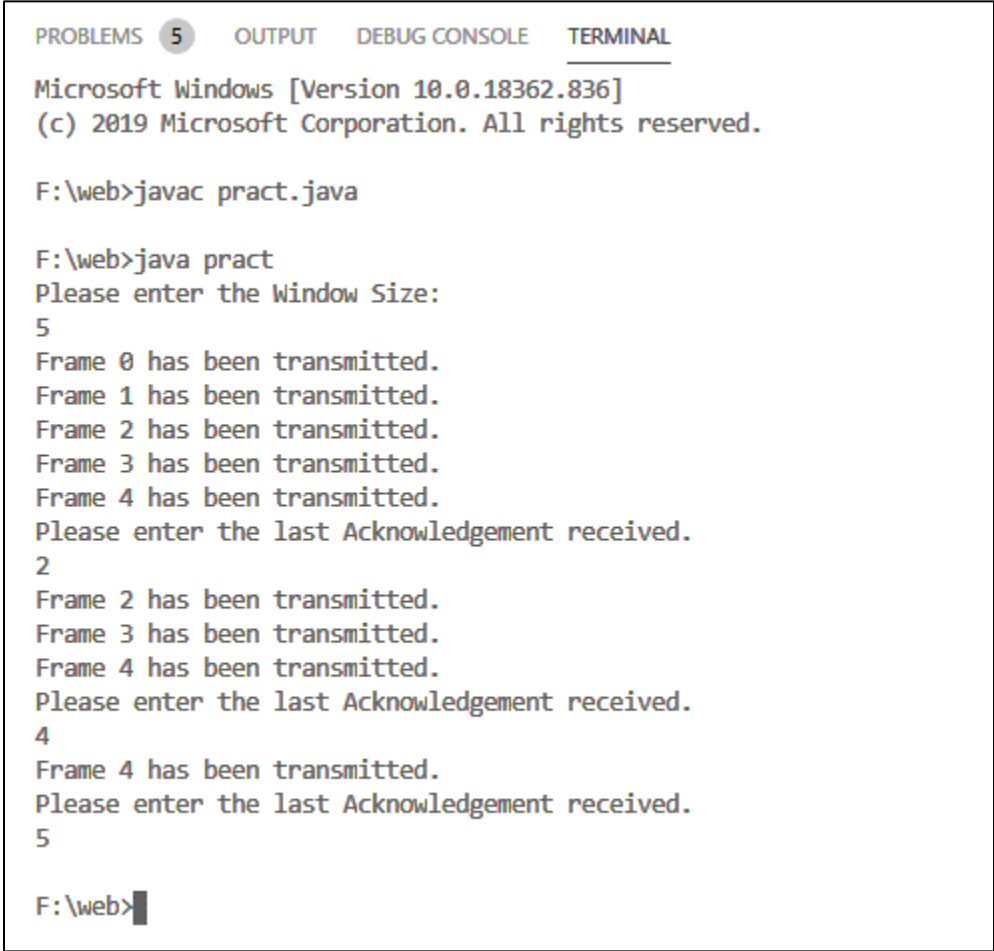
else

    sent = ack;

}

}

}
```

OUTPUT:

```
PROBLEMS 5 OUTPUT DEBUG CONSOLE TERMINAL
Microsoft Windows [Version 10.0.18362.836]
(c) 2019 Microsoft Corporation. All rights reserved.

F:\web>javac pract.java

F:\web>java pract
Please enter the Window Size:
5
Frame 0 has been transmitted.
Frame 1 has been transmitted.
Frame 2 has been transmitted.
Frame 3 has been transmitted.
Frame 4 has been transmitted.
Please enter the last Acknowledgement received.
2
Frame 2 has been transmitted.
Frame 3 has been transmitted.
Frame 4 has been transmitted.
Please enter the last Acknowledgement received.
4
Frame 4 has been transmitted.
Please enter the last Acknowledgement received.
5

F:\web>
```

SLIDING WINDOW PROTOCOL (SELECTIVE REPEAT)

ALGORITHM:

- Step 1: Start the program.
- Step 2: Generate a random that gives the total number of frames to be transmitted.
- Step 3: Set the size of the window.
- Step 4: Generate a random number less than or equal to the size of the current window and identify the number of frames to be transmitted at a given time.
- Step 5: Transmit the frames and receive the acknowledgement for the frames sent.
- Step 6: Find the remaining frames to be sent.
- Step 7: Find the current window size.
- Step 8: If an acknowledgement is not received for a particular frame retransmit that frame alone again.
- Step 9: Repeat the steps 4 to 8 till the number of remaining frames to be send becomes zero.
- Step 10: Stop the program.

SOURCE CODE:

Client:

```
import java.lang.System;

import java.net.*;

import java.io.*;

import java.text.*;

import java.util.Random;

import java.util.*;

public class client {

    static Socket connection;

    public static void main(String a[]) throws SocketException {

        try {

            int v[] = new int[10];

            int n = 0;
```

```

Random rand = new Random();

int rand = 0;

InetAddress addr = InetAddress.getByName("Localhost");

System.out.println(addr);

connection = new Socket(addr, 1234);

DataOutputStream out = new DataOutputStream(
                                connection.getOutputStream());

DataInputStream in = new DataInputStream(
                                connection.getInputStream());

int p = in.read();

System.out.println("No of frame is: " + p);

for (int i = 0; i < p; i++) {
    v[i] = in.read();

    System.out.println(v[i]);

    //g[i] = v[i];
}

rand = rand.nextInt(p); //FRAME NO. IS RANDOMLY GENERATED

v[rand] = -1;

for (int i = 0; i < p; i++)
{
    System.out.println("Received frame is: " + v[i]);
}

for (int i = 0; i < p; i++)

```



```
if (v[i] == -1) {  
    System.out.println("Request to retransmit from packet no "  
        + (i+1) + " again!!");  
    n = i;  
    out.write(n);  
    out.flush();  
}  
System.out.println();  
v[n] = in.read();  
System.out.println("Received frame is: " + v[n]);  
System.out.println("quiting");  
} catch (Exception e) {  
    System.out.println(e);  
}  
}  
}
```

Receiver:

```
import java.io.DataInputStream;  
import java.io.DataOutputStream;  
import java.io.IOException;  
import java.net.ServerSocket;  
import java.net.Socket;
```

```
import java.net.SocketException;

public class server
{
    static ServerSocket Serversocket;

    static DataInputStream dis;

    static DataOutputStream dos;

    public static void main(String[] args) throws SocketException
    {
        try
        {
            int a[] = { 30, 40, 50, 60, 70, 80, 90, 100 };

            Serversocket = new ServerSocket(1234);

            System.out.println("waiting for connection");

            Socket client = Serversocket.accept();

            dis = new DataInputStream(client.getInputStream());

            dos = new DataOutputStream(client.getOutputStream());

            System.out.println("The number of packets sent is:" + a.length);

            int y = a.length;

            dos.write(y);

            dos.flush();

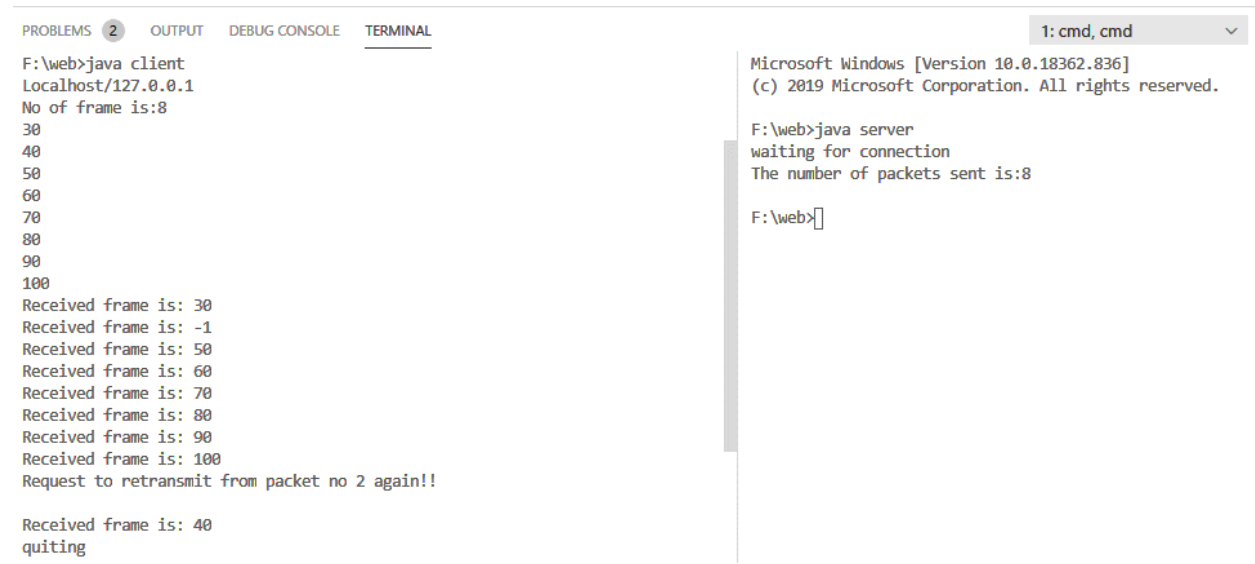
            for (int i = 0; i < a.length; i++)
            {
```

```
        dos.write(a[i]);  
        dos.flush();  
    }  
  
    int k = dis.read();  
  
    dos.write(a[k]);  
    dos.flush();  
}  
catch (IOException e)  
{  
    System.out.println(e);  
}  
finally  
{  
    try  
    {  
        dis.close();  
        dos.close();  
    }  
    catch (IOException e)  
    {  
        e.printStackTrace();  
    }  
}
```

CN LAB

```
}  
  
}  
  
}  
  
}
```

OUTPUT:



```
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL
F:\web>java client
Localhost/127.0.0.1
No of frame is:8
30
40
50
60
70
80
90
100
Received frame is: 30
Received frame is: -1
Received frame is: 50
Received frame is: 60
Received frame is: 70
Received frame is: 80
Received frame is: 90
Received frame is: 100
Request to retransmit from packet no 2 again!!

Received frame is: 40
quiting
```

```
1: cmd, cmd
Microsoft Windows [Version 10.0.18362.836]
(c) 2019 Microsoft Corporation. All rights reserved.

F:\web>java server
waiting for connection
The number of packets sent is:8

F:\web>
```

PRACTICAL - 8

AIM: Implementation and study of distance vector routing algorithm.

DESCRIPTION:

- In distance vector routing, each router periodically shares its knowledge about the entire internet with its neighbors.
- The three keys to understanding how this algorithm works are as follows:
- Sharing knowledge about the entire autonomous system.
- Sharing only with neighbors.
- Sharing at regular intervals.
- Every router keeps a routing table that has one entry for each destination network which the router aware

BACKGROUND:

- Routing algorithm is a part of network layer software which is responsible for deciding which output line an incoming packet should be transmitted on. If the subnet uses datagram internally, this decision must be made anew for every arriving data packet since the best route may have changed since last time. If the subnet uses virtual circuits internally, routing decisions are made only when a new established route is being set up. The latter case is sometimes called session routing, because a route remains in force for an entire user session (e.g., login session at a terminal or a file).
- Routing algorithms can be grouped into two major classes: adaptive and non-adaptive. Non Adaptive algorithms do not base their routing decisions on measurement or estimates of current traffic and topology. Instead, the choice of route to use to get from I to J (for all I and J) is computed in advance, offline, and downloaded to the routers when the network is booted. This procedure is sometimes called static routing.
- Adaptive algorithms, in contrast, change their routing decisions to reflect changes in the topology, and usually the traffic as well. Adaptive algorithms differ in where they get information (e.g., locally, from adjacent routers, or from all routers), when they change the routes, and what metric is used for optimization (e.g., distance, number of hops, or estimated transit time).
- Two algorithms in particular, distance vector routing and link state routing are the most popular. Distance vector routing algorithms operate by having each router maintain a table (i.e., vector) giving the best known distance to each destination and which line to get there. These tables are updated by exchanging information with the neighbour's. The distance vector routing algorithm is sometimes called by other names, including the distributed Bellman-Ford routing algorithm and the Ford-Fulkerson algorithm, after the researchers who developed it (Bellman, 1957; and Ford and Fulkerson, 1962). It was the original ARPANET routing algorithm and was also used in the Internet under the RIP and in early versions of DEC net and Novell's IPX. AppleTalk and Cisco routers use improved distance vector protocols.
- In distance vector routing, each router maintains a routing table indexed by, and containing one entry for, each router in subnet. This entry contains two parts: the preferred outgoing line to use for that destination, and an estimate of the time or distance to that destination. The metric used might be number of hops, time delay in milliseconds, total number of packets queued along the path, or something similar.
- The router is assumed to know the "distance" to each of its neighbours. If the metric is hops, the distance is just one hop. If the metric is queue length, the router simply examines each queue. If

the metric is delayed, the router can measure it directly with special ECHO packets that the receiver just timestamps and sends back as fast as possible.

ALGORITHM:

1. Add one hop to the hop count for each advertised destination.
2. Repeat the following steps for each advertised destination
3. If(destination not in the routing table)
Add the advertised information to the table.
4. Else If(next-hop field is the same)
Replace entry in the table with the advertised one.
5. Else If(advertized hop count smaller than one in table)
Replace entry in Routing table.

SOURCE CODE:

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <limits.h>

struct Edge

{
    int source, destination, weight;
};

struct Graph

{
    int V, E;

    struct Edge* edge;
};

struct Graph* createGraph(int V, int E)

{
    struct Graph* graph = (struct Graph*) malloc( sizeof(struct Graph));

    graph->V = V;
```

```

graph->E = E;

graph->edge = (struct Edge*) malloc( graph->E * sizeof( struct Edge ) );

return graph;
}

void FinalSolution(int dist[], int n)
{
printf("\nVertex\tDistance from Source Vertex\n");

    int i;

    for (i = 0; i < n; ++i){
        printf("%d \t\t %d\n", i, dist[i]);
    }
}

void BellmanFord(struct Graph* graph, int source)
{
    int V = graph->V;

    int E = graph->E;

    int StoreDistance[V];

    int i,j;

    for (i = 0; i < V; i++)
StoreDistance[i] = INT_MAX;

StoreDistance[source] = 0;

    for (i = 1; i <= V-1; i++)
    {
        for (j = 0; j < E; j++)
        {

```

```

    int u = graph->edge[j].source;

    int v = graph->edge[j].destination;

    int weight = graph->edge[j].weight;

    if (StoreDistance[u] + weight < StoreDistance[v])
StoreDistance[v] = StoreDistance[u] + weight;

    }

}

for (i = 0; i < E; i++)

{

    int u = graph->edge[i].source;

    int v = graph->edge[i].destination;

    int weight = graph->edge[i].weight;

    if (StoreDistance[u] + weight < StoreDistance[v])

printf("This graph contains negative edge cycle\n");

}

```

```

FinalSolution(StoreDistance, V);

return;

}

int main()

{

    int V,E,S;

printf("Enter number of vertices in graph\n");

scanf("%d",&V);

printf("Enter number of edges in graph\n");

```



```
scanf("%d",&E);

printf("Enter your source vertex number\n");

scanf("%d",&S);

struct Graph* graph = createGraph(V, E);

int i;

for(i=0;i<E;i++){

printf("\nEnter edge %d properties Source, destination, weight respectively\n",i+1);

scanf("%d",&graph->edge[i].source);

scanf("%d",&graph->edge[i].destination);

scanf("%d",&graph->edge[i].weight);

}

BellmanFord(graph, S);

return 0;

}
```

OUTPUT:

```
Enter number of vertices in graph
5
Enter number of edges in graph
10
Enter your source vertex number
0

Enter edge 1 properties Source, destination, weight respectively
0
1
6

Enter edge 2 properties Source, destination, weight respectively
0 2 7

Enter edge 3 properties Source, destination, weight respectively
1 2 8

Enter edge 4 properties Source, destination, weight respectively
1 4 -4

Enter edge 5 properties Source, destination, weight respectively
1 3 5

Enter edge 6 properties Source, destination, weight respectively
3 1 -2

Enter edge 7 properties Source, destination, weight respectively
2 3 -3

Enter edge 8 properties Source, destination, weight respectively
2 4 9

Enter edge 9 properties Source, destination, weight respectively
4 0 2

Enter edge 10 properties Source, destination, weight respectively
4 3 7

Vertex   Distance from Source Vertex
0         0
1         2
2         7
```

PRACTICAL - 9**AIM: Implementation and study of link state routing algorithm.****EXPLANATION:**

Link-state routing protocols are one of the two main classes of routing protocols used in packet switching networks for computer communications, the other being distance-vector routing protocols. Examples of link-state routing protocols include open shortest path first (OSPF) and intermediate system to intermediate system (IS-IS).

The link-state protocol is performed by every *switching node* in the network (i.e., nodes that are prepared to forward packets; in the Internet, these are called routers). The basic concept of link-state routing is that every node constructs a *map* of the connectivity to the network, in the form of a graph, showing which nodes are connected to which other nodes. Each node then independently calculates the next best logical *path* from it to every possible destination in the network. The collection of best paths will then form the node's routing table.

First, each node needs to determine what other ports it is connected to, over fully working links; it does this using a reachability protocol which it runs periodically and separately with each of its directly connected neighbors.

Next, each node periodically (and in case of connectivity changes) sends a short message, the link-state advertisement, which:

- Identifies the node which is producing it.
- Identifies all the other nodes (either routers or networks) to which it is directly connected.
- Includes a sequence number, which increases every time the source node makes up a new version of the message.

This message is sent to all that node's neighbors. As a necessary precursor, each node in the network remembers, for every one of its neighbors, the sequence number of the last link-state message which it received from that node. When a link-state advertisement is received at a node, the node looks up the sequence number it has stored for the source of that link-state message: if this message is newer (i.e., has a higher sequence number), it is saved, and a copy is sent in turn to each of that node's neighbors. This procedure rapidly gets a copy of the latest version of each node's link-state advertisement to every node in the network.

Networks running link state algorithms can also be segmented into hierarchies which limit the scope of route changes. These features mean that link state algorithms scale better to larger networks.

Finally, with the complete set of link-state advertisements (one from each node in the network) in hand, each node produces the graph for the map of the network. The algorithm iterates over the collection of link-state advertisements; for each one, it makes links on the map of the network, from the node which sent that message, to all the nodes which that message indicates are neighbors of the sending node.

No link is considered to have been correctly reported unless the two ends agree; i.e., if one node reports that it is connected to another, but the other node does not report that it is connected to the first, there is a problem, and the link is not included on the map.

The link-state message giving information about the neighbors is recomputed, and then flooded throughout the network, whenever there is a change in the connectivity between the node and its neighbors; e.g., when a link fails. Any such change will be detected by the reachability protocol which each node runs with its neighbors.

The second main stage in the link-state algorithm is to produce routing tables, by inspecting the maps. Each node independently runs an algorithm over the map to determine the shortest path from itself to every other node in the network. For this, Dijkstra's algorithm is used.

ALGORITHM:

Step 1: Use a routing protocol to collect the whole network topology

Step 2: Obtain destination reachability information as well as link weights/states

Step 3: Compute shortest paths using Dijkstra's algorithm from a node to all other nodes

Step 4: Construct routing tables that show the destination addresses and the next hop addresses

Note that while Dijkstra's algorithm gives you end-to-end routes, the routing tables may only store the next hop address.

DIJKSTRA'S ALGORITHM:

Let the node at which we are starting be called the initial node. Let the distance of node Y be the distance from the initial node to Y. Dijkstra's algorithm will assign some initial distance values and will try to improve them step by step.

1. Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes.
2. Mark all nodes unvisited. Set the initial node as current. Create a set of the unvisited nodes called the unvisited set consisting of all the nodes except the initial node.
3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances. For example, if the current node A is marked with a distance of 6, and the edge connecting it with a neighbor B has length 2, then the distance to B (through A) will be $6+2=8$. If this distance is less than the previously recorded tentative distance of B, then overwrite that distance. Even though a neighbor has been examined, it is not marked as "visited" at this time, and it remains in the unvisited set.
4. When we are done considering all of the neighbors of the current node, mark the current node as visited and remove it from the unvisited set. A visited node will never be checked again.
5. If the destination node has been marked visited (when planning a route between two specific nodes) or if the smallest tentative distance among the nodes in the unvisited set is infinity (when planning a complete traversal), then stop. The algorithm has finished.
6. Select the unvisited node that is marked with the smallest tentative distance, and set it as the new "current node" then go back to step 3.

SOURCE CODE:

```
#include<stdio.h>

#include<conio.h>

#define INFINITY 9999

#define MAX 10

void dijkstra(int G[MAX][MAX],int n,intstartnode);

int main()
{
    int G[MAX][MAX],i,j,n,u;

    printf("Enter no. of vertices:");

    scanf("%d",&n);

    printf("\nEnter the adjacency matrix:\n");

    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            scanf("%d",&G[i][j]);

    printf("\nEnter the starting node:");

    scanf("%d",&u);

    dijkstra(G,n,u);

    return 0;
}

void dijkstra(int G[MAX][MAX],int n,intstartnode)
{
    int cost[MAX][MAX],distance[MAX],pred[MAX];

    int visited[MAX],count,mindistance,nextnode,i,j;
```

```
for(i=0;i<n;i++)
    for(j=0;j<n;j++)
        if(G[i][j]==0)
            cost[i][j]=INFINITY;
        else
            cost[i][j]=G[i][j];

for(i=0;i<n;i++)
{
    distance[i]=cost[startnode][i];
    pred[i]=startnode;
    visited[i]=0;
}
distance[startnode]=0;
visited[startnode]=1;
count=1;
while(count<n-1)
{
    mindistance=INFINITY;
    for(i=0;i<n;i++)
        if(distance[i]<mindistance&&!visited[i])
        {
            mindistance=distance[i];
            nextnode=i;
        }
}
```

CN LAB

```
visited[nextnode]=1;
for(i=0;i<n;i++)
    if(!visited[i])
        if(mindistance+cost[nextnode][i]<distance[i])
        {
            distance[i]=mindistance+cost[nextnode][i];
            pred[i]=nextnode;
        }
count++;
}

for(i=0;i<n;i++)
    if(i!=startnode)
    {
        printf("\nDistance of node%d=%d",i,distance[i]);
        printf("\nPath=%d",i);

        j=i;
        do
        {
            j=pred[j];
            printf("<-%d",j);
        }while(j!=startnode);
    }
}
```

OUTPUT:

```
Enter no. of vertices:5

Enter the adjacency matrix:
0 10 0 30 100
10 0 50 0 0
0 50 0 20 10
30 0 20 0 60
100 0 10 60 0

Enter the starting node:0

Distance of node1=10
Path=1<-0
Distance of node2=50
Path=2<-3<-0
Distance of node3=30
Path=3<-0
Distance of node4=60
Path=4<-2<-3<-0_
```



1:59 AM
5/2019

PRACTICAL - 10**AIM: Simulate Star Topology using TCP server using NS3.**

```

#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 = 10; //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
    CommandLineCmd;
    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.

```

CN LAB

```
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]);  
}  
  
// 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 ();
```

CN LAB

```
// Create a packet sink on the star "hub" to receive these packets  
uint16_t port = 50000;  
Address sinkLocalAddress (InetSocketAddress (Ipv4Address::GetAny (), port));  
PacketSinkHelpersinkHelper ("ns3::TcpSocketFactory", sinkLocalAddress);  
ApplicationContainersinkApp = sinkHelper.Install (serverNode);  
sinkApp.Start (Seconds (1.0));  
sinkApp.Stop (Seconds (10.0));  
  
// Create the OnOff applications to send TCP to the server  
OnOffHelperclientHelper ("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  
ApplicationContainerclientApps;  
for(uint32_t i=0; i<clientNodes.GetN (); ++i)  
{  
AddressValueremoteAddress  
(InetSocketAddress (interfaceAdjacencyList[i].GetAddress (0), port));  
clientHelper.SetAttribute ("Remote", remoteAddress);  
clientApps.Add (clientHelper.Install (clientNodes.Get (i)));  
}  
clientApps.Start (Seconds (2.0));  
clientApps.Stop (Seconds (10.0));  
  
//configure tracing  
AsciiTraceHelper ascii;  
p2p.EnableAsciiAll (ascii.CreateFileStream ("tcp-star-server.tr"));  
p2p.EnablePcapAll ("tcp-star-server");  
  
// ApplicationContainerclientApps=echoClient.Install(nodes.Get(0));  
//clientApps.Start(Seconds(2.0));  
//clientApps.Stop(Seconds(10.0));  
AnimationInterfaceanim("appu3.xml");  
anim.SetConstantPosition(serverNode.Get(0),25,25);  
anim.SetConstantPosition(clientNodes.Get(1), 00,00);  
anim.SetConstantPosition(clientNodes.Get(2), 25,00);  
anim.SetConstantPosition(clientNodes.Get(3), 50,00);  
anim.SetConstantPosition(clientNodes.Get(4), 00,25);  
anim.SetConstantPosition(clientNodes.Get(5), 50,25);  
anim.SetConstantPosition(clientNodes.Get(6), 00,50);  
anim.SetConstantPosition(clientNodes.Get(7), 25,50);  
anim.SetConstantPosition(clientNodes.Get(8), 50,50);
```

```

NS_LOG_INFO ("Run Simulation.");
Simulator::Run ();
Simulator::Destroy ();
NS_LOG_INFO ("Done.");

return 0;
}

```

OUTPUT:

