Ex No.	2	Implementation of Client-Server Communication Using TCP
Date		

AIM:
To implement a chat server and client in java using TCP sockets.

DESCRIPTION:
TCP Clients sends request to server and server will receives the request and response with acknowledgement. Every time client communicates with server and receives response from it.

ALGORITHM:

Server
1. Create a server socket and bind it to port.
2. Listen for new connection and when a connection arrives, accept it.
3. Read Client's message and display it
4. Get a message from user and send it to client
5. Repeat steps 3-4 until the client sends "end"
6. Close all streams
7. Close the server and client socket
8. Stop

Client
1. Create a client socket and connect it to the server"s port number
2. Get a message from user and send it to server
3. Read server's response and display it
4. Repeat steps 2-3 until chat is terminated with "end" message
5. Close all input/output streams
6. Close the client socket
7. Stop

PROGRAM:

//Server.java

```
import java.io.*;
import java.net.*;
class Server {
 public static void main(String args[]) {
 String data = "Networks Lab";
 try {
 ServerSocket srvr = new ServerSocket(1234);
 Socket skt = srvr.accept();
 System.out.print("Server has connected!\n");
 PrintWriter out = new PrintWriter(skt.getOutputStream(), true);
 System.out.print("Sending string: '" + data + "'\n");
```

```
out.print(data);
out.close();
skt.close();
srvr.close();
}
catch(Exception e) {
System.out.print("Whoops! It didn't work!\n");
}
}
}
```

//Client.java

```
import java.io.*;
import java.net.*;
class Client {
public static void main(String args[]) {
try {
Socket skt = new Socket("localhost", 1234);
BufferedReader in = new BufferedReader(new
InputStreamReader(skt.getInputStream()));
System.out.print("Received string: '");
while (!in.ready()) {}
System.out.println(in.readLine());
System.out.print("'\n");
in.close();
}
catch(Exception e) {
System.out.print("Whoops! It didn't work!\n");
}}}
```

OUTPUT

Server:

$ javac Server.java
$ java Server
Server started
Client connected

Cilent

$ javac Client.java
$ java Client

Ex No.	3	IMPLEMENTATION OF TCP/IP ECHO
Date		

AIM:
To implementation of echo client server using TCP/IP

DESCRIPTION:
TCP Server gets the message and opens the server socket to read the client details. Client send its address to the server. Then client receives the message from server to display.

ALGORITHM

Server

1. Create a server socket and bind it to port.
2. Listen for new connection and when a connection arrives, accept it.
3. Read the data from client.
4. Echo the data back to the client.
5. Repeat steps 4-5 until „bye" or „null" is read.
6. Close all streams.
7. Close the server socket.
8. Stop.

Client

1. Create a client socket and connect it to the server"s port number.
2. Get input from user.
3. If equal to bye or null, then go to step 7.
4. Send user data to the server.
5. Display the data echoed by the server.
6. Repeat steps 2-4.
7. Close the input and output streams.
8. Close the client socket.
9. Stop.

Program:

// TCP Echo Server--tcpechoserver.java

```
import java.net.*;
import java.io.*;
public class tcpechoserver
{
public static void main(String[] arg) throws IOException
{
ServerSocket sock = null;
BufferedReader fromClient = null;
OutputStreamWriter toClient = null;
Socket client = null;
```

```
try
{
sock = new ServerSocket(4000); System.out.println("Server Ready");
client = sock.accept(); System.out.println("Client Connected");
fromClient = new BufferedReader(new
InputStreamReader(client.getInputStream()));
toClient = new OutputStreamWriter(client.getOutputStream());
String line;
while (true)
{
line = fromClient.readLine();
if ( (line == null) || line.equals("bye"))
break;
System.out.println ("Client [ " + line + " ]");
toClient.write("Server [ "+ line +" ]\n");
toClient.flush();
}
fromClient.close();
toClient.close();
client.close();
sock.close();
System.out.println("Client Disconnected");
}
catch (IOException ioe)
{
System.err.println(ioe);
}
}
}
```

//TCP Echo Client--tcpechoclient.java

```
import java.net.*;
import java.io.*;
public class tcpechoclient
{
public static void main(String[] args) throws IOException
{
BufferedReader fromServer = null, fromUser = null;
PrintWriter toServer = null;
Socket sock = null;
try
{
if (args.length == 0)
sock = new Socket(InetAddress.getLocalHost(),4000);
```

```
else
sock = new Socket(InetAddress.getByName(args[0]),4000);
fromServer = new BufferedReader(new
InputStreamReader(sock.getInputStream()));
fromUser = new BufferedReader(new InputStreamReader(System.in));
toServer = new PrintWriter(sock.getOutputStream(),true);
String Usrmsg, Srvmsg;
System.out.println("Type \"bye\" to quit");
while (true)
{
System.out.print("Enter msg to server : ");
Usrmsg = fromUser.readLine();
if (Usrmsg==null || Usrmsg.equals("bye"))
{
toServer.println("bye"); break;
}
else
toServer.println(Usrmsg);
Srvmsg = fromServer.readLine();
System.out.println(Srvmsg);
}
fromUser.close();
fromServer.close();
toServer.close();
sock.close();
}
catch (IOException ioe)
{
System.err.println(ioe);
}
```

OUTPUT :

Server:

```
$ javac tcpechoserver.java
$ java tcpechoserver
Server Ready Client Connected Client [ hello ]
Client [ how are you ] Client [ i am fine ] Client [ ok ]
Client Disconnected
```

Client :

```
$ javac tcpechoclient.java
$ java tcpechoclient
Type "bye" to quit
Enter msg to server : hello
Server [ hello ]
Enter msg to server : how are you
Server [ how are you ]
Enter msg to server : i am fine
Server [ i am fine ]
Enter msg to server : ok
Server [ ok ]
Enter msg to server : bye
```

3

Ex No.	4	PROGRAM USING UDP SOCKET UDP CHAT SERVER/CLIENT
Date		

AIM:

To implement a chat server and client in java using UDP sockets.

DESCRIPTION:

UDP is a connectionless protocol and the socket is created for client and server to transfer the data. Socket connection is achieved using the port number. Domain Name System is the naming convention that divides the Internet into logical domains identified in Internet Protocol version 4 (IPv4) as a 32-bit portion of the total address.

ALGORITHM:

Server

1. Create two ports, server port and client port.
2. Create a datagram socket and bind it to client port.
3. Create a datagram packet to receive client message.
4. Wait for client's data and accept it.
5. Read Client's message.
6. Get data from user.
7. Create a datagram packet and send message through server port.
8. Repeat steps 3-7 until the client has something to send.
9. Close the server socket.
10. Stop.

Client

1. Create two ports, server port and client port.
2. Create a datagram socket and bind it to server port.
3. Get data from user.
4. Create a datagram packet and send data with server ip address and client port.
5. Create a datagram packet to receive server message.
6. Read server's response and display it.
7. Repeat steps 3-6 until there is some text to send.
8. Close the client socket.
9. Stop.

Program:

// UDP Chat Server--udpchatserver.java

```
import java.io.*;
import java.net.*;
class udpchatserver
{
public static int clientport = 8040,serverport = 8050;
```

```
public static void main(String args[]) throws Exception
{
DatagramSocket SrvSoc = new DatagramSocket(clientport);
byte[] SData = new byte[1024];
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
System.out.println("Server Ready");
while (true)
{
byte[] RData = new byte[1024];
DatagramPacket RPack = new DatagramPacket(RData,RData.length);
SrvSoc.receive(RPack);
String Text = new String(RPack.getData());
if (Text.trim().length() == 0)
break;
System.out.println("\nFrom Client <<< " + Text );
System.out.print("Msg to Cleint : " );
String srvmsg = br.readLine();
InetAddress IPAddr = RPack.getAddress();
SData = srvmsg.getBytes();
DatagramPacket SPack = new DatagramPacket(SData,SData.length,IPAddr,
serverport);
SrvSoc.send(SPack);
}
System.out.println("\nClient Quits\n");
SrvSoc.close();
}
}
```

// UDP Chat Client--udpchatclient.java

```
import java.io.*;
import java.net.*;
class udpchatclient
{
public static int clientport = 8040,serverport = 8050;
public static void main(String args[]) throws Exception
{
BufferedReader br = new BufferedReader(new InputStreamReader (System.in));
DatagramSocket CliSoc = new DatagramSocket(serverport);
InetAddress IPAddr;
String Text;
if (args.length == 0)
IPAddr = InetAddress.getLocalHost();
else
IPAddr = InetAddress.getByName(args[0]);
```

```
byte[] SData = new byte[1024];
System.out.println("Press Enter without text to quit");
while (true)
{
System.out.print("\nEnter text for server : ");
Text = br.readLine();
SData = Text.getBytes();
DatagramPacket SPack = new DatagramPacket(SData,SData.length, IPAddr,
clientport );
CliSoc.send(SPack);
if (Text.trim().length() == 0)
break;
byte[] RData = new byte[1024];
DatagramPacket RPack = new DatagramPacket(RData,RData.length);
CliSoc.receive(RPack);
String Echo = new String(RPack.getData()) ;
Echo = Echo.trim();
System.out.println("From Server <<< " + Echo);
}
CliSoc.close();
}
}
```

OUTPUT :

Server

```
$ javac udpchatserver.java
$ java udpchatserver
Server Ready
From Client <<< are u the SERVER
Msg to Cleint : yes
From Client <<< what do u have to serve
Msg to Cleint : no eatables
Client Quits
```

Client

```
$ javac udpchatclient.java
$ java udpchatclient
Press Enter without text to quit
Enter text for server : are u the SERVER From Server <<< yes
Enter text for server : what do u have to serve
From Server <<< no eatables
Enter text for server : Ok
```

Ex No.	8	NETWORK TOPOLOGY : BUS TOPOLOGY
Date		

AIM:
To create scenario and study the performance of token bus protocol through simulation.

HARDWARE / SOFTWARE REQUIREMENTS:
NS-2

THEORY:
Token bus is a LAN protocol operating in the MAC layer. Token bus is standardized as per IEEE 802.4. Token bus can operate at speeds of 5Mbps, 10 Mbps and 20 Mbps. The operation of token bus is as follows: Unlike token ring in token bus the ring topology is virtually created and maintained by the protocol. A node can receive data even if it is not part of the virtual ring, a node joins the virtual ring only if it has data to transmit. In token bus data is transmitted to the destination node only where as other control frames is hop to hop. After each data transmission there is a solicit_successsor control frame transmitted which reduces the performance of the protocol.

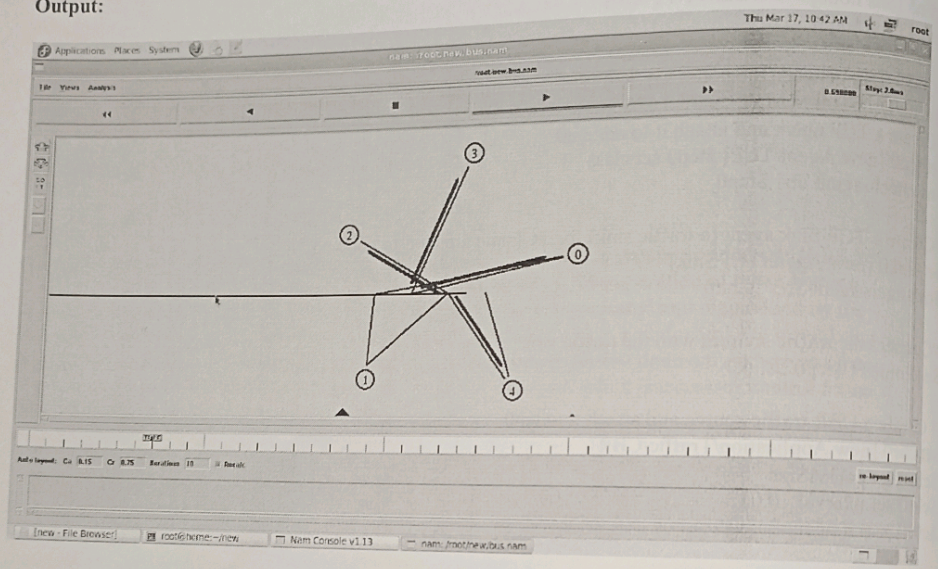
ALGORITHM:
1. Create a simulator object
2. Define different colors for different data flows
3. Open a nam trace file and define finish procedure then close the trace file, and execute nam on trace file.
4. Create five nodes that forms a network numbered from 0 to 4
5. Create duplex links between the nodes and add Orientation to the nodes for setting a LAN topology
6. Setup TCP Connection between n(1) and n(3)
7. Apply CBR Traffic over TCP.
8. Schedule events and run the program.

PROGRAM:

```
#Create a simulator object
set ns [new Simulator]

#Open the nam trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#Define a 'finish' procedure
proc finish {} {
        global ns nf
        $ns flush-trace
        #Close the trace file
        close $nf
        #Executenam on the trace file
        exec nam out.nam &
        exit 0
}
#Create five nodes
set n0 [$ns node]
set n1 [$ns node]
```

```
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]

#Create Lan between the nodes set lan0 [$ns newLan "$n0 $n1 $n2 $n3 $n4" 0.5Mb 40ms LL
Queue/DropTail MAC/Csma/Cd Channel]
#Create a TCP agent and attach it to node n0
set tcp0 [new Agent/TCP] $tcp0 set class_ 1
$ns attach-agent $n1 $tcp0

#Create a TCP Sink agent (a traffic sink) for TCP and attach it to node n3
set sink0 [new Agent/TCPSink]
$ns attach-agent $n3 $sink0

#Connect the traffic sources with the traffic sink
$ns connect $tcp0 $sink0

# Create a CBR traffic source and attach it to tcp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.01
$cbr0 attach-agent $tcp0

#Schedule events for the CBR agents
$ns at 0.5 "$cbr0 start"
$ns at 4.5 "$cbr0 stop"

#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"

#Run the simulation
$ns run
```

Output:


Thu Mar 17, 10:42 AM
root
Applications Places System
File Views Analysis
0
1
2
3
4
Nam Console v1.13

Ex No.	10	NETWORK TOPOLOGY : STAR TOPOLOGY
Date		

AIM:
To create scenario and study the performance of token ring protocols through simulation.

HARDWARE / SOFTWARE REQUIREMENTS:
NS-2

THEORY:
Star networks are one of the most common computer network topologies. In its simplest form, a star network consists of one central switch, hub or computer, which acts as a conduit to transmit messages. This consists of a central node, to which all other nodes are connected; this central node provides a common connection point for all nodes through a hub. In star topology, every node (computer workstation or any other peripheral) is connected to a central node called a hub or switch. The switch is the server and the peripherals are the clients. Thus, the hub and leaf nodes, and the transmission lines between them, form a graph with the topology of a star. If the central node is passive, the originating node must be able to tolerate the reception of an echo of its own transmission, delayed by the two-way transmission time (i.e. to and from the central node) plus any delay generated in the central node. An active star network has an active central node that usually has the means to prevent echo-related problems.
The star topology reduces the damage caused by line failure by connecting all of the systems to a central node. When applied to a bus-based network, this central hub rebroadcasts all transmissions received from any peripheral node to all peripheral nodes on the network, sometimes including the originating node. All peripheral nodes may thus communicate with all others by transmitting to, and receiving from, the central node only. The failure of a transmission line linking any peripheral node to the central node will result in the isolation of that peripheral node from all others, but the rest of the systems will be unaffected.

ALGORITHM:
1. Create a simulator object
2. Define different colors for different data flows
3. Open a nam trace file and define finish procedure then close the trace file, and execute nam on trace file.
4. Create six nodes that forms a network numbered from 0 to 5
5. Create duplex links between the nodes to form a STAR Topology
6. Setup TCP Connection between n(1) and n(3)
7. Apply CBR Traffic over TCP
8. Schedule events and run the program.

PROGRAM:

```
#Create a simulator object set
ns [new Simulator]

#Open the nam trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#Define a 'finish' procedure
```

```
proc finish {} {
        global ns nf
        $ns flush-trace
        #Close the trace file
        close $nf
        #Executenam on the trace file
        exec nam out.nam &
        exit0

}

#Create six nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]

#Change the shape of center node in a star topology
$n0 shape square

#Create links between the nodes
$ns duplex-link $n0 $n1 1Mb 10ms DropTail
$ns duplex-link $n0 $n2 1Mb 10ms DropTail
$ns duplex-link $n0 $n3 1Mb 10ms DropTail
$ns duplex-link $n0 $n4 1Mb 10ms DropTail
$ns duplex-link $n0 $n5 1Mb 10ms DropTail

#Create a TCP agent and attach it to node n0
set tcp0 [new Agent/TCP]
$tcp0 set class_ 1
$ns attach-agent $n1 $tcp0

#Create a TCP Sink agent (a traffic sink) for TCP and attach it to node n3
set sink0 [new Agent/TCPSink]
$ns attach-agent $n3 $sink0

#Connect the traffic sources with the traffic sink
$ns connect $tcp0 $sink0

# Create a CBR traffic source and attach it to tcp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.01
$cbr0 attach-agent $tcp0
#Schedule events for the CBR agents
$ns at 0.5 "$cbr0 start"
$ns at 4.5 "$cbr0 stop"

#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"
```

#Run the simulation
$ns run

Output: