SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 of the UGC Act, 1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

Computer Networks

Laboratory Manual

Course Code: CSE303

Semester: V

Lab Manual

2024

SHANMUGHA ARTS, SCIENCE, TECHNOLOGY AND RESEARCH ACADEMY

(SASTRA Deemed to be) University

Tirumalaisamudram, Thanjavur-613 401

School of Computing

Course Objective:

This course will help the learner to understand socket programming using TCP and UDP, simulate and analyse various network models. It makes the learner to understand various protocols and standards in networking

Course Learning Outcomes:

Upon successful completion of this course, the learner will be able to

- Demonstrate socket programming and RMI
- Evaluate remote method invocation protocol
- Analyze various TCP flow control and congestion control algorithms
- Compare and criticize various routing protocols
- Demonstrate wired LAN using simulator
- Evaluate wireless LAN using various metrics

List of Experiments:

1. Development of a chat application based on socket programming. Development of a secure file transfer application.
2. Implementation of Remote Method Invocation.
3. Simulation and analysis of wired network.
4. Simulation and analysis of wireless network.
5. Simulation and analysis of TCP flow control and TCP congestion control. Simulation and comparative analysis of distance vector routing and link state routing.
6. Simulation of Broadcast routing and multicast routing.
7. Simulation and analysis of DHCP protocol.
8. Simulation and analysis of Ethernet LAN IEEE 802.3.
9. Simulation and analysis of Wireless LAN IEEE 802.11.
10. Study of SASTRA network infrastructure.

Exercise No. 1 Development of a Chat Application Based on Socket Programming

To develop a chat application using socket programming that allows multiple users to communicate in real-time over a network. This involves implementing client-server architecture, ensuring efficient data transmission, handling multiple connections, and providing a user-friendly interface for seamless communication.

Objectives

1. Implement a basic client-server architecture using socket programming.
2. Enable real-time communication between multiple users over a network.
3. Handle multiple client connections simultaneously.

Prerequisite / Tools Required

Tools: Java Programming

Interface: Graphical User Interface (GUI) using (Java Swing/AWT)

Theory or Concept

- **Socket class**

Object of the Socket class provides features to communicate between client and server.

- **Server Socket class**

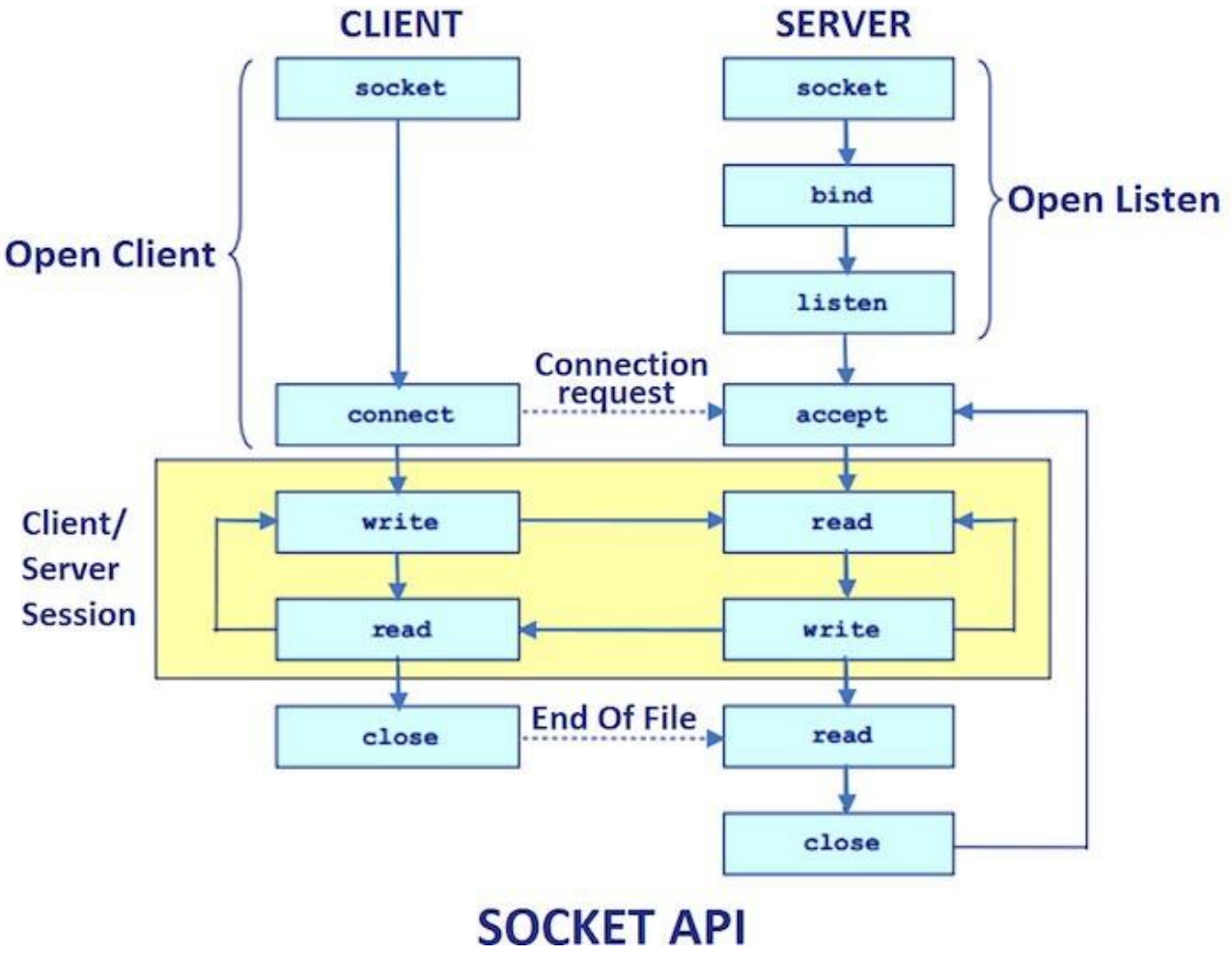
Object of the Server Socket to establish communication with the clients.

Use the Java Swing to create GUI with desired components. Some important components are like

- Use JFrame class to create a GUI.
- JTextfield ---------- for text entry
- JTextarea ---------- text area for display texts
- JButton ---------- button to do some action
- JPanel to attach other GUI components

Procedure / Algorithm

- Set Up the Development Environment
- Design the Chat Application Architecture
- Implement the Server
    - Create the server socket
    - Accept and handle client connections
- Implement the Client
    - Create the client socket
    - Send and receive messages



- Enhance and Debug the Application
- Deploy the Application

Sample Code

Important Methods:

	Method Description
public InputStream getInputStream()	returns the InputStream attached with this socket.
public OutputStream getOutputStream()	returns the OutputStream attached with this socket.
public synchronized void close()	closes this socket

Output / Performance Measures

Important Methods:	**Method Description**
public Socket accept()	returns the socket and establish a connection between server and client.
public synchronized void close()	closes the server socket.

Creating Server:

ServerSocket ss=new ServerSocket(4567);	// 4567 - user defined port number
Socket s=ss.accept();	//establishes connection and waits for the client

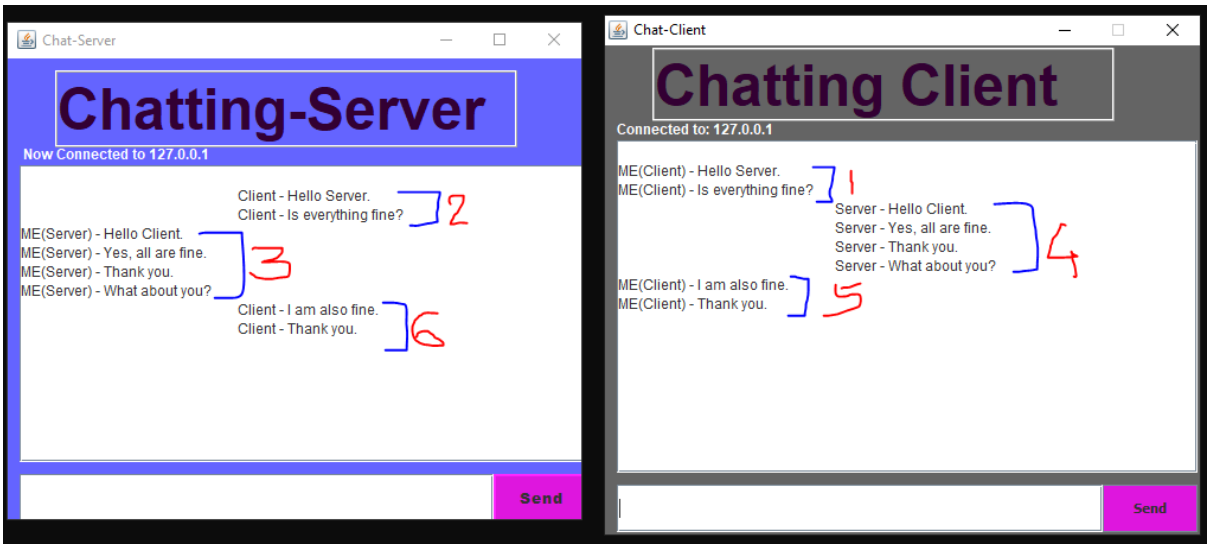
Creating Client:

We need to pass the IP address or hostname of the Server and a port number.

For the same system, we can either use "localhost" or 127.0.0.1.

Socket s=new Socket ("localhost",4567);

Output:



Exercise No. 2 Development of a Secure File Transfer Application

To develop a secure file transfer application that ensures the safe and efficient transmission of files between users over a network.

Objectives

- Develop a secure file transfer application using appropriate programming languages and tools.
- Implement encryption techniques to ensure data confidentiality during file transmission.
- Incorporate authentication mechanisms to verify user identities and prevent unauthorized access.
- Design a user-friendly interface for easy file selection, upload, and download.

Prerequisite / Tools Required:

Tools: Java Programming

Interface: Graphical User Interface (GUI) using (Java Swing/AWT)

Theory or Concept

Important Classes connection-oriented socket programming:

Socket class -- to communicate client and server

ServerSocket class -- to listen clients

- **Socket class**

Object of the Socket class provides features to communicate between client and server.

- **Server Socket class**

Object of the Server Socket to establish communication with the clients.

Use the Java Swing to create GUI with desired components. Some important components are like

- Use JFrame class to create a GUI.
- JTextfield ---------- for text entry
- JTextarea ---------- text area for display texts
- JButton ---------- button to do some action
- JPanel to attach other GUI components

To run the client-server programs, following points you need to follow:

- Keep the Server program (TCPServer.java) in separate directory Compile it, and run it.
- To run it, atleat one argument is required i.e. the path of the Server program such “c:\server program\”.
- Keep the Client program (TCPClient.java) in separate directory Compile it, and run it.
- To run it, atleat one argument is required i.e. the path of the Client program such “c:\client program\”.
- There two options in Client user interface i.e. Upload and Download. Upload
- User writes filename such \CData.txt, and then press Upload button, which encrypts the client's file using CAESAR cipher, and sends to the server.

Download

User chooses one of the available files from server to download i.e. Sdata.txt, and then puts **“Backslash”** before chosen file like \Sdata.txt, upon pressing **Download** button, the data of the chosen file gets encrypted and sent to the client, and the decryption of the data gets done using **CAESAR cipher**.

CAESAR cipher: it is one of earliest and easist symmetric cipher. It uses 3 as the default key.

Procedure / Algorithm

- Set Up the Development Environment
- Design the Chat Application Architecture
- Implement the Server
    - Create the server socket
    - Accept and handle client connections
- Implement the Client
    - Create the client socket
    - Implement file upload and download
- Enhance and Debug the Application

Deploy the Application

Sample Code

Important Methods:

	Method Description
public InputStream getInputStream()	returns the InputStream attached with this socket.

public OutputStream getOutputStream()	returns the OutputStream attached with this socket.
public synchronized void close()	closes this socket

Output / Performance Measures

Important Methods:	**Method Description**
public Socket accept()	returns the socket and establish a connection between server and client.
public synchronized void close()	closes the server socket.

Creating Server:

ServerSocket ss=new ServerSocket(4567);	// 4567 - user defined port number
Socket s=ss.accept();	//establishes connection and waits for the client

Creating Client:

We need to pass the IP address or hostname of the Server and a port number.

For the same system, we can either use "localhost" or 127.0.0.1.

Socket s=new Socket ("localhost",4567);

Encryption:

CipherText=(PlainText + 3) mod 256 // 256 for entire ASCII data

Decryption:

PlainText=(CipherText – 3) mod 256

if PlainText is <0, then add PlainText with 256

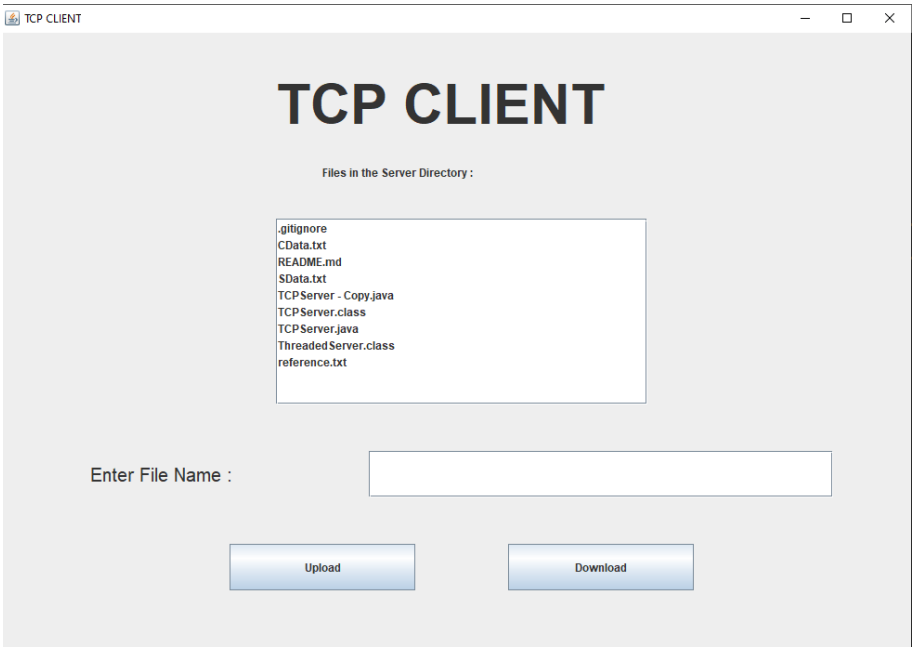
Output / Performance Measures

Server runs and waits for client

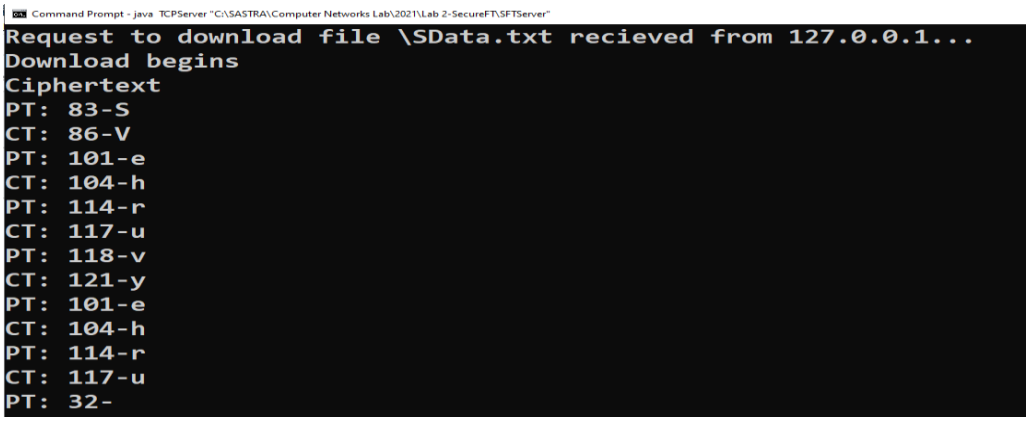
```
C:\SASTRA\Computer Networks Lab\2021\Lab 2-SecureFT\SFTServer>java TCPServer "
C:\SASTRA\Computer Networks Lab\2021\Lab 2-SecureFT\SFTServer"
Server started...
Waiting for connections...
```

Client runs, a window gets appeared and waits for user interaction

```
C:\SASTRA\Computer Networks Lab\2021\Lab 2-SecureFT\SFTClient>java TCPClient "
C:\SASTRA\Computer Networks Lab\2021\Lab 2-SecureFT\SFTClient"
Server says Hi!
```

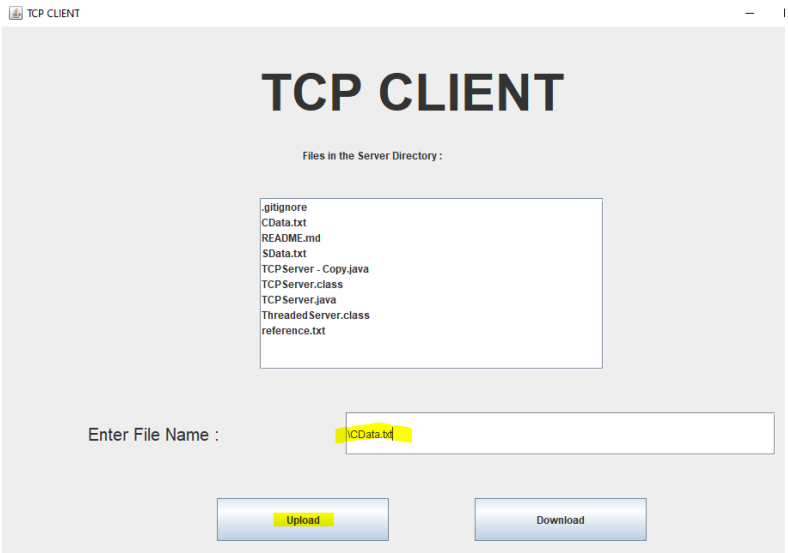


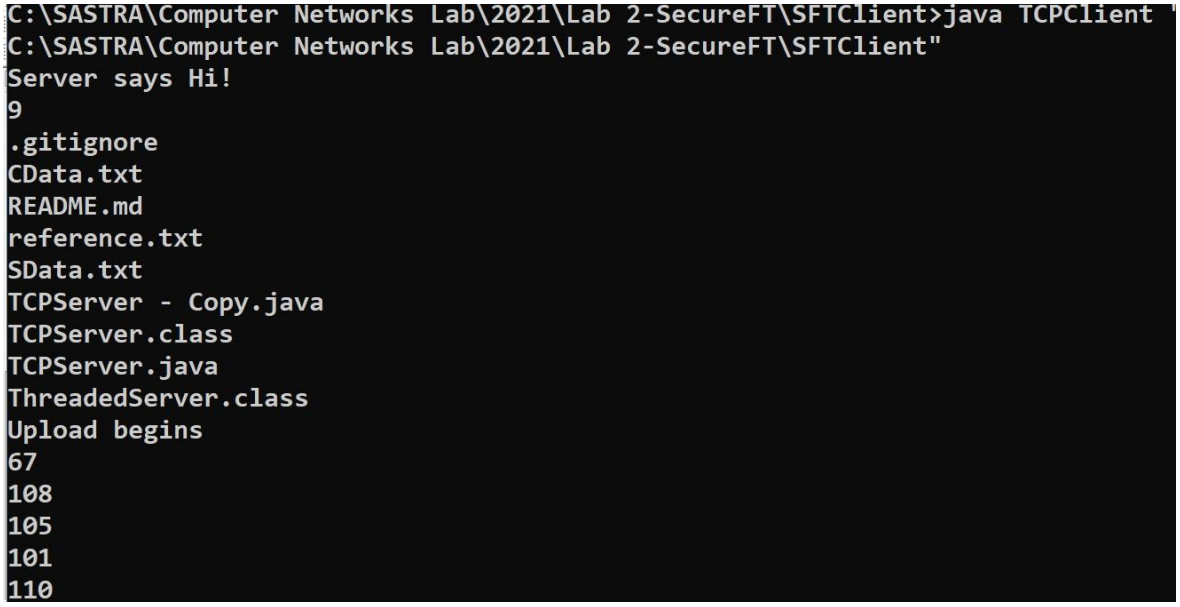
Suppose, you want **download** a file such as SData.txt, and press Download button, then the following type of screen appears.



Note: Verify the **SData.txt** at client's directory, it contains plaintext, because, while downloading at client system, decryption process gets executed.

Suppose, you want **Upload** a file such as CData.txt, and press Upload button, then the following type of screen appears.





Here, the cipher text are shown as numbers.

Note: Verify the **Cdata.txt** at server directory, it contains ciphertext.

Exercise No. 3 Implementation of Remote Method Invocation (RMI)

To implement Remote Method Invocation (RMI) in a Java application, allowing methods to be called from a remote Java Virtual Machine (JVM).

Objectives

- Learn the concepts of Remote Procedure Call and Remote Method Invocation
- Learn to develop applications which uses Distributed Objects
- Learn the working and significance of stub object (Client side) and Skeleton objects (Server side).

Prerequisite / Tools Required:

Tools: Java Programming

Interface: Character User Interface (CUI)

Procedure / Algorithm

- Creating an interface Both client and server have to agree Implementing the interface Should contain the operational code
- Creating the RMI server Contains both class and interface Binding with RMI registry (namespace, keeps server's object)
- Creating the RMI client Call Registry to obtain reference to remote object Compiling and starting the server Refer

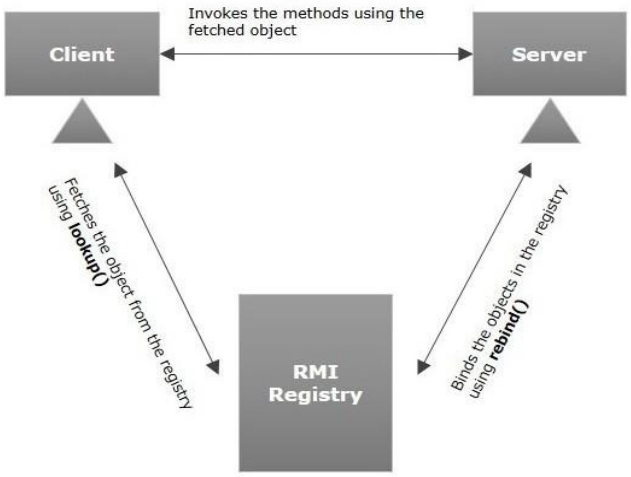


Figure1: RMI Registry Process

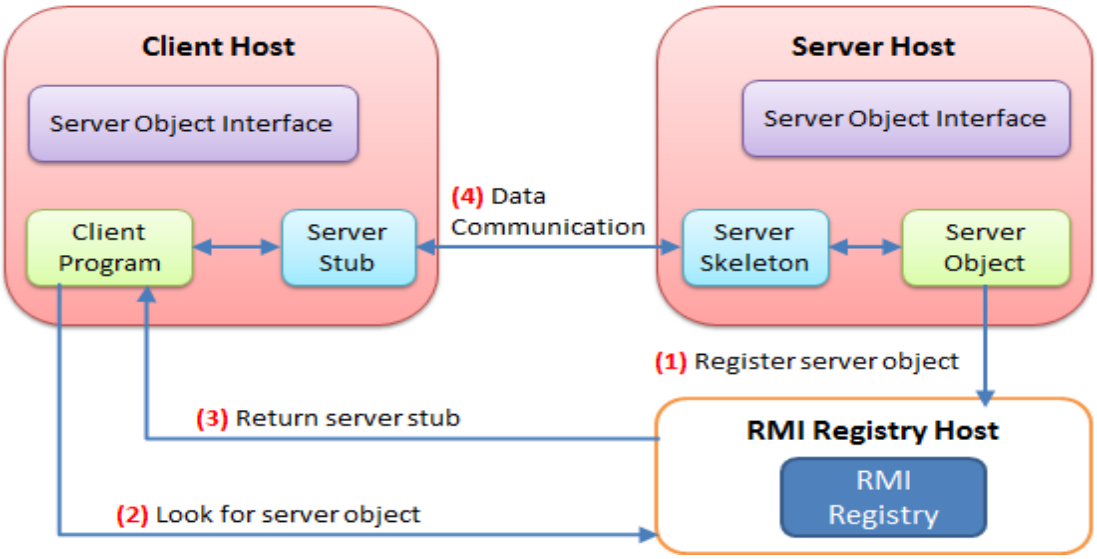


Figure2: Implementation of RMI

Sample Code

Input	Output
Add function, Parameters(operands)	Return value
200,300	500

Output / Performance Measures:

To RUN

- Open two command prompt and set the java path & class path (if not set).
- Compile all java prog- java file (Client & Server), using javac *.java, in any one prompt.
- At Server-side, run RMIRegistry using: start rmiregistry, an additional prompt will be opened, and rmiregistry gets started. Do not close it until the demonstration is not finished.

Start server program using: **java MyServer**

```
C:\SASTRA\Computer Networks Lab\2021\Lab 3 - Impl_of_RMI>javac *.java

C:\SASTRA\Computer Networks Lab\2021\Lab 3 - Impl_of_RMI>start rmiregistry

C:\SASTRA\Computer Networks Lab\2021\Lab 3 - Impl_of_RMI>java MyServer
Server Started...
```

Run client program from another system or prompt using server ip: j**ava MyClient <IP or localhost>**

```
C:\SASTRA\Computer Networks Lab\2021\Lab 3 - Impl_of_RMI>java MyClient localho
st
Addition: 500.0
```

Exercise No. 4 Simulation and Analysis of Wired Network using NS2 Simulator

To Simulate and analysis of wired network using NS2 simulator.

Objectives

- Model network topology.
- Implement network protocols (e.g., TCP, UDP).
- Analyze traffic patterns.
- Evaluate performance metrics (e.g., throughput, delay, packet loss).
- Optimize network parameters and configurations.

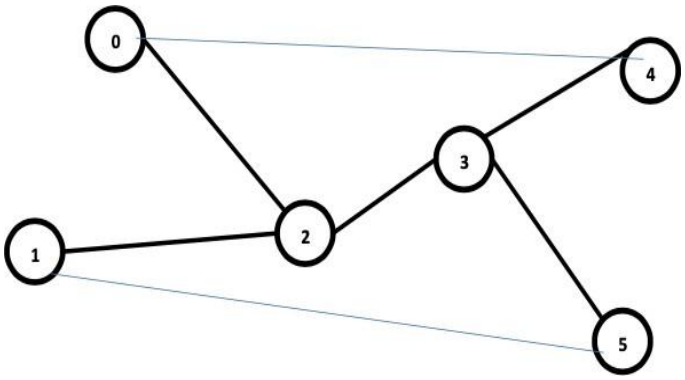
Prerequisite / Tools Required:

Tools: NS2 Simulator

Procedure / Algorithm

- Node 0 – Act as an agent and can send the UDP packet at CBR mode.
- Node 1 – Act as an agent and can send TCP packet at FTP mode.
- Node 4 – Receives the packet from Node 0. It is going to receive UDP packet
- hence it is NULL agent.
- Node 5 – Receives the packet from Node 1. It is going to receive TCP packet
- hence it acts as Sink Agent.
- Link capacity between Node 0 to Node 2 is 5Mbps bandwidth and 2ms speed.
- Link capacity between Node 1 to Node 2 is 10Mbps bandwidth and 5ms speed.
- Link capacity between Node 2 to Node 3 is 4Mbps bandwidth and 3ms speed.
- Link capacity between Node 3 to Node 4 is 100Mbps bandwidth and 2ms speed.
- Link capacity between Node 3 to Node 5 is 15Mbps bandwidth and 4ms speed.

Network Design:



Software Requirements:

- Oracle VM VirtualBox for Windows systems
- Install Linux on the above VM
- Install ns2 in the above Linux System
    - (at Linux command prompt: sudo apt-get install ns2)
- Install nam in the above Linux system
    - (at Linux command prompt: sudo apt-get install nam)

How to run a simulation

Write code using any editor in the Linux system and save the file with .tcl extension, ie. Tool Command Language. For example, use the gedit editor

ns <file_name>.tcl ——> to run

nam <file_name>.nam ——-> to see simulation

gedit <file_name>.tr ——-> to see trace details

Sample Code

Sample Code for above design using NS2 (<file_name>.tcl)

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

#Create a trace file, this file is for logging purpose
set tracefile [open wired.tr w]
$ns trace-all $tracefile

#Create a animation information or NAM file creation set
namfile [open wired.nam w]
$ns namtrace-all $namfile

#Create 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]

#Creation of link between nodes with Drop Tail Queue
#Droptail means Dropping the tail.
$nsduplex-link$n0$n25Mb2ms DropTail
$nsduplex-link$n1$n210Mb5msDropTail
$nsduplex-link$n2$n34Mb3ms DropTail
$nsduplex-link$n3$n4100Mb2msDropTail
$nsduplex-link$n3$n515Mb4msDropTail

#CreationofAgents #Node 0 to Node 4
setudp[newAgent/UDP] setnull[new Agent/Null]
$nsattach-agent$n0$udp
$nsattach-agent$n4$null
$nsconnect$udp$null

#CreationofTCPAgent settcp[newAgent/TCP]
setsink[newAgent/TCPSink]
$nsattach-agent$n1 $tcp
$nsattach-agent$n5 $sink
$nsconnect$tcp $sink

#CreationofApplicationCBR,FTP
#CBR-ConstantBitRate(Examplemp3filesthathaveaCBRor192kbps,320kbps,etc.)
#FTP - File Transfer Protocol (Ex: To download a file from a network)
```

```
setcbr[newApplication/Traffic/CBR]
$cbrattach-agent$udp
setftp[newApplication/FTP]
$ftpattach-agent$tcp

#Startthetraffic
$nsat1.0"$cbr start"
$nsat2.0"$ftpstart"

$nsat10.0"finish"
#The following procedure will be called at 10.0 seconds proc finish {} {
globalnstracefilenamfile
$ns flush-trace close $tracefile close $namfile exit 0
}
puts"Simulationisstarting..."
$nsrun
```

Exercise No. 5 Simulation and Analysis of Wireless Network using NS2 Simulator

Simulate and analysis of wireless network using NS2 simulator.

Objectives

- Model wireless network topologies and scenarios.
- Implement and simulate wireless communication protocols (e.g., IEEE 802.11).
- Analyze the impact of mobility patterns on network performance.
- Evaluate performance metrics such as throughput, packet loss, and delay.
- Optimize network parameters including transmission power and routing protocols.

Prerequisite / Tools Required

Tools: NS2 simulator

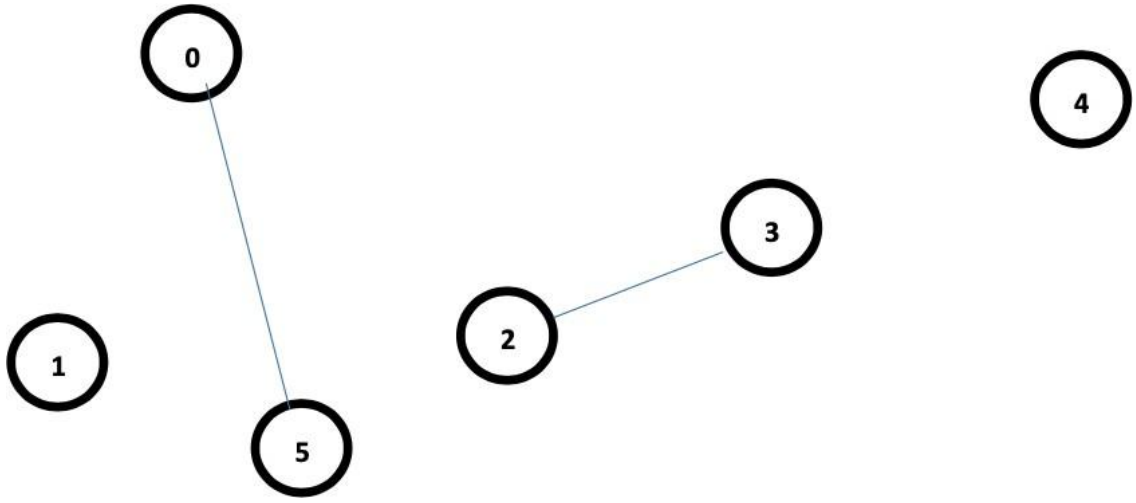
Procedure / Algorithm

Node 2 – Act as an agent and can send the UDP packet at CBR mode.

Node 0 – Act as an agent and can send TCP packet at FTP mode.

Node 3 – Receives the packet from Node 2. It is going to receive UDP packet hence it is NULL agent.

Node 5 – Receives the packet from Node 0. It is going to receive TCP packet hence it acts as Sink Agent.



Node 0, 2, 3 - Stationary Node

Node 1, 4, 5 - Mobile Node

Sample Code
Sample Code for above design using NS2 (<file_name>.tcl)

```
#Example of Wireless networks
#Step 1 initialize variables
#Step 2 - Create a Simulator object
#step 3 - Create Tracing and animation file
#step 4 - topography
#step 5 - GOD - General Operations Director
#step 6 - Create nodes
#Step 7 - Create Channel (Communication PATH)
#step 8 - Position of the nodes (Wireless nodes needs a location)
#step 9 - Any mobility codes (if the nodes are moving)
#step 10 - TCP, UDP Traffic
#run the simulation

#Initialize the variables
set val(chan) Channel/WirelessChannel ;#Channel Type
set val(prop) Propagation/TwoRayGround ;# radio-propagation model
set val(netif) Phy/WirelessPhy ;# network interface type WAVELAN DSSS
2.4GHz
set val(mac) Mac/802_11 ;# MAC type
set val(ifq) Queue/DropTail/PriQueue ;# interface queue type
set val(ll) LL ;# link layer type
set val(ant) Antenna/OmniAntenna ;# antenna model
set val(ifqlen) 50 ;# max packet in ifq
set val(nn) 6 ;# number of mobilenodes
set val(rp) AODV ;# routing protocol
set val(x) 500 ;# in metres set
val(y) 500 ;# in metres
#Adhoc OnDemand Distance Vector

#Creation of Simulator
set ns [new Simulator]

#Creation of Trace and namefile
set tracefile [open wireless.tr w]
$ns trace-all $tracefile

#Creation of Network Animation file
set namfile [open wireless.nam w]
$ns namtrace-all-wireless $namfile $val(x) $val(y)
```

```
#Create Topography
set topo [new Topography]
$topo load_flatgrid $val(x) $val(y)

#GOD Creation - General Operations Director
create-god $val(nn)
set channel1 [new $val(chan)]
set channel2 [new $val(chan)]
set channel3 [new $val(chan)]

#Configure the Node
$ns node-config -adhocRouting $val(rp) \
-llType $val(ll) \
-macType $val(mac) \
-ifqType $val(ifq) \
-ifqLen $val(ifqlen) \
-antType $val(ant) \
-propType $val(prop) \
-phyType $val(netif) \
-topoInstance $topo \
-agentTrace ON \
-macTrace ON \
-routerTrace ON \
-movementTrace ON \
-channel $channel1
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
$n0 random-motion 0
$n1 random-motion 0
$n2 random-motion 0
$n3 random-motion 0
$n4 random-motion 0
$n5 random-motion 0
$ns initial_node_pos $n0 20
$ns initial_node_pos $n1 20
$ns initial_node_pos $n2 20
$ns initial_node_pos $n3 20
$ns initial_node_pos $n4 20
$ns initial_node_pos $n5 50
```

```
#Initial coordinates of the nodes
$n0 set X_ 10.0
$n0 set Y_ 20.0
$n0 set Z_ 0.0
$n1 set X_ 210.0
$n1 set Y_ 230.0
$n1 set Z_ 0.0
$n2 set X_ 100.0
$n2 set Y_ 200.0
$n2 set Z_ 0.0
$n3 set X_ 150.0
$n3 set Y_ 230.0
$n3 set Z_ 0.0
$n4 set X_ 430.0
$n4 set Y_ 320.0
$n4 set Z_ 0.0
$n5 set X_ 270.0
$n5 set Y_ 120.0
$n5 set Z_ 0.0

#Dont mention any values above than 500 because in this example, we use X and Y as
500,500

#Mobility of the nodes
#At what Time? Which node? Where to? at What Speed?
$ns at 1.0 "$n1 setdest 490.0 340.0 25.0"
$ns at 1.0 "$n4 setdest 300.0 130.0 5.0"
$ns at 1.0 "$n5 setdest 190.0 440.0 15.0"

#The nodes can move any number of times at any location during the simulation
(runtime)
$ns at 20.0 "$n5 setdest 100.0 200.0 30.0"

#Creation of agents
set tcp [new Agent/TCP]
set sink [new Agent/TCPSink]
$ns attach-agent $n0 $tcp
$ns attach-agent $n5 $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 1.0 "$ftp start"
set udp [new Agent/UDP]
```

```
set null [new Agent/Null]
$ns attach-agent $n2 $udp
$ns attach-agent $n3 $null
$ns connect $udp $null
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$ns at 1.0 "$cbr start"
$ns at 30.0 "finish"
proc finish {} {
global ns tracefile namfile
$ns flush-trace
close $tracefile
close $namfile
exit 0
}
puts "Starting Simulation"
$ns run
```

Note:

AODV (Ad-hoc On-demand Distance Vector) is a loop-free routing protocol for ad hoc networks.
It is designed to be self-starting in an environment of mobile nodes, withstanding a variety of network behaviors such as node mobility, link failures and packet losses. At each node, AODV maintains a routing table.

TwoRayGround Model: Predicts losses, and act accordingly. In telecommunications, *Direct-Sequence Spread Spectrum* (**DSSS**) is a spread-spectrum modulation technique primarily used to reduce overall signal interference.

Exercise No. 6 Simulation and analysis of TCP flow control and TCP Congestion Control

To simulate and analyze TCP flow control and TCP congestion control mechanisms. This involves modelling and evaluating how TCP manages data transmission rates, controls congestion, and ensures reliable communication in a network environment.

Objectives

- Comparison of various TCP versions
- Wired NW Phases of congestion control:
- Slow start
- Congestion avoidance
- Congestion detection

Prerequisite / Tools Required:

Tools: ns-2, nsg, gnuplot

Theory or Concept

SlowStart -> exponential start (1,2,4,8, may reduce also) ->|ssthreshold-> +1-

--> |congestion detection (heavy cogestion ->slow start, light congestion ->

congestion avoidance).

There are 4 variables:

- **rwnd - receiver window**
- **swnd - sender window**
- **cwnd - congestion window**
- **ssthreshold**

x-axis: number of the timing
y-axis: cwnd value

Procedure / Algorithm

- Set Up the Development Environment
- Model TCP Flow Control and Congestion Control
- Create Simulation Scenarios
- Implement the Simulation
- Evaluate TCP Mechanisms

Sample Code

==== Software requirements========== TCP Congestion Control - ***ns2***

Additional Software:

NSG (NS-2 Scenarios Generator2 (NSG-2)) for scenario generation

$] java -jar NSG2.1.jar // run

ns2

$] ns filename.tcl

gnuplot

$] gnuplot

gnuplot] plot "file.plt"

To install gnuplot:

$] sudo apt install gnuplot

To install Java:

$] sudo apt install default-jdk //if Java not available, then install

Exercise No. 7 Simulation and comparative analysis of distance vector Routing and link state routing

To simulate and conduct a comparative analysis of Distance Vector Routing and Link State Routing protocols. This involves evaluating their performance in various network scenarios, assessing key metrics such as convergence time, routing overhead, and scalability.

Objectives

- Implement and simulate Distance Vector Routing protocol using NS2.
- Implement and simulate Link State Routing protocol using NS2.
- Model various network topologies to test the performance of both routing protocols.
- Analyze the convergence time of Distance Vector and Link State routing.
- Evaluate routing overhead for both protocols.

Prerequisite / Tools Required:

Tools: NS2 Simulator

Theory or Concept

- In **link state routing**, each router shares its knowledge of its neighborhood with every other router in the internet work.
- Knowledge about Neighborhood: Instead of sending its entire routing table a router sends info about its neighborhood only.
- To all Routers: each router sends this information to every other router on the internet work not just to its neighbor .It does so by a process called flooding.
- Information sharing when there is a change: Each router sends out information about the neighbors when there is change.

Procedure / Algorithm

The Dijkstra algorithm follows four steps to discover what is called the shortest path tree(routing table) for each router: The algorithm begins to build the tree by identifying its roots. The root router's trees the router itself. The algorithm then attaches all nodes that can be reached from the root. The algorithm compares the tree's temporary arcs and identifies the arc with the lowest cumulative cost. This arc and the node to which it connects are now a permanent part of the shortest path tree. The algorithm examines the database and identifies every node that can be reached from its chosen node. These nodes and their arcs are added temporarily to the tree. The last

two steps are repeated until every node in the network has become a permanent part of the tree.

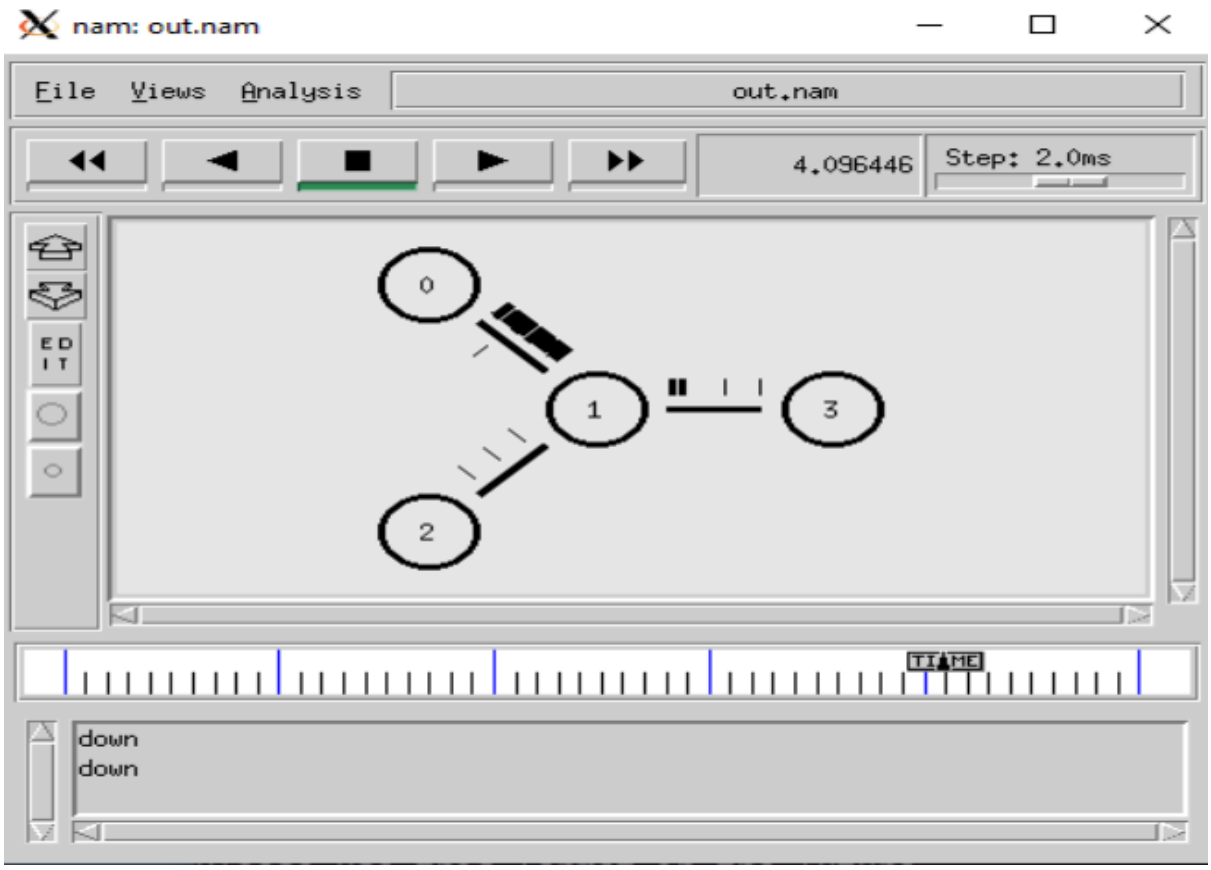
- Create a simulator object.
- Define different colors for different data flows.
- Open nam and trace files and define finish procedure then close the nam & trace files, and execute nam on trace file.
- Create n number of nodes using for loop.
- Create duplex links between the nodes.
- Setup UDP Connection between n(0) and n(5).
- Setup another UDP connection between n(1) and n(5).
- Apply CBR Traffic over both UDP connections.
- Choose Link state routing protocol to transmit data from sender to receiver.
- Schedule events and run the program.

Sample Code

```
set ns [new Simulator]
set nr [open thro.tr w]
$ns trace-all $nr
set nf [open thro.nam w]
$ns namtrace-all $nf
proc finish { }
{ global ns nr
nf
$ns flushtrace
close
$nf close $nr
exec nam
thro.nam & exit 0
}
for { set i 0 } { $i < 12} { incr i
1 } { set n($i) [$ns node]}
for {set i 0} {$i < 8} {incr i} {
$ns duplex-link $n($i) $n([expr $i+1]) 1Mb 10ms DropTail }
$ns duplex-link $n(0) $n(8) 1Mb 10ms DropTail
$ns duplex-link $n(1) $n(10) 1Mb 10ms DropTail
$ns duplex-link $n(0) $n(9) 1Mb 10ms DropTail
$ns duplex-link $n(9) $n(11) 1Mb 10ms DropTail
$ns duplex-link $n(10) $n(11) 1Mb 10ms DropTail
```

```
$ns duplex-link $n(11) $n(5) 1Mb 10ms
DropTail set udp0 [new Agent/UDP]
$ns attach-agent $n(0) $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent
$udp0 set null0 [new
Agent/Null]
$ns attach-agent $n(5) $null0
$ns connect $udp0
$null0 set udp1 [new
Agent/UDP]
$ns attach-agent $n(1) $udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp1
set null0 [new Agent/Null]
$ns attach-agent $n(5) $null0
$ns connect $udp1 $null0
$ns rtproto LS
$ns rtmodel-at 10.0 down $n(11) $n(5)
$ns rtmodel-at 15.0 down $n(7) $n(6)
$ns rtmodel-at 30.0 up $n(11) $n(5)
$ns rtmodel-at 20.0 up $n(7) $n(6)
$udp0 set fid_ 1
$udp1 set fid_ 2
$ns color 1 Red
$ns color 2 Green
$ns at 1.0 "$cbr0 start"
$ns at 2.0 "$cbr1 start"
$ns at 45 "finish"
$ns run
```

Output:


nam: out.nam
File
Views
Analysis
out.nam
4.096446
Step: 2.0ms
0
1
2
3
TIME
down
down

Exercise No. 8 Simulation and Analysis of Multicast Routing

To simulate and analyze multicast routing protocols in a network environment using a network simulator such as NS2. This involves evaluating the performance and efficiency of multicast routing mechanisms in terms of metrics like data delivery ratio, end-to-end delay, and network overhead.

Objectives

- Implement and simulate various multicast routing protocols using NS2.
- Model network topologies suitable for multicast communication.
- Evaluate the data delivery ratio of different multicast routing protocols.
- Measure end-to-end delay in multicast data transmission.
- Analyze the network overhead associated with multicast routing.

Prerequisite / Tools Required:

Tools: NS2 Simulator

Procedure / Algorithm

- Create a simulator object with multicast ON.
- Open a nam trace file and define finish procedure then close the trace file, and execute nam on trace file.
- Create n number of nodes.
- Create duplex links between the nodes.
- Set Dense Mode protocol for multicasting.
- Set two different groups with group address.
- Set UDP Transport agent for the traffic source for group1 and another agent for the traffic source for group 2.
- Create more number of receivers to accept the packets for two different groups.
- Specify more number of nodes join and leave from the group at various time.
- Schedule events and run the program.

Sample Code

```
set ns [new Simulator -multicast on]
#Turn on Tracing
set tf [open output.tr w]
$ns trace-all $tf

# Turn on nam Tracing set
fd [open mcast.nam w]
$ns namtrace-all $fd
```

```
# Create 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] set
n6 [$ns node] set
n7 [$ns node]

# Create links with DropTail Queues
$ns duplex-link $n0 $n2 1.5Mb 10ms DropTail
$ns duplex-link $n1 $n2 1.5Mb 10ms DropTail
$ns duplex-link $n2 $n3 1.5Mb 10ms DropTail
$ns duplex-link $n3 $n4 1.5Mb 10ms DropTail
$ns duplex-link $n3 $n7 1.5Mb 10ms DropTail
$ns duplex-link $n4 $n5 1.5Mb 10ms DropTail
$ns duplex-link $n4 $n6 1.5Mb 10ms DropTail
set mproto DM
set mrthandle [$ns mrtproto $mproto {}]

# Set two groups with group addresses
set group1 [Node allocaddr]
set group2 [Node allocaddr]

# UDP Transport agent for the traffic source for
group1 set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
$udp0 set dst_addr_ $group1
$udp0 set dst_port_ 0
set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp0

# Transport agent for the traffic source for
group2 set udp1 [new Agent/UDP]
$ns attach-agent $n1 $udp1
$udp1 set dst_addr_ $group2
$udp1 set dst_port_ 0
set cbr2 [new Application/Traffic/CBR]
```

```
$cbr2 attach-agent $udp1

# Create receiver to accept the
packets set rcvr1 [new Agent/Null]
$ns attach-agent $n5 $rcvr1
$ns at 1.0 "$n5 join-group $rcvr1
$group1" set rcvr2 [new Agent/Null]
$ns attach-agent $n6 $rcvr2
$ns at 1.5 "$n6 join-group $rcvr2 $group1"
set rcvr3 [new Agent/Null]
$ns attach-agent $n7 $rcvr3
$ns at 2.0 "$n7 join-group
$rcvr3 $group1"
set rcvr4 [new Agent/Null]
$ns attach-agent $n5 $rcvr1
$ns at 2.5 "$n5 join-group $rcvr4 $group2"
set rcvr5 [new Agent/Null]
$ns attach-agent $n6 $rcvr2
$ns at 3.0 "$n6 join-group $rcvr5 $group2"
set rcvr6 [new Agent/Null]
$ns attach-agent $n7 $rcvr3

#The nodes are leaving the group at specified times
$ns at 3.5 "$n7 join-group $rcvr6 $group2"
$ns at 4.0 "$n5 leave-group $rcvr1 $group1"
$ns at 4.5 "$n6 leave-group $rcvr2 $group1"
$ns at 5.0 "$n7 leave-group $rcvr3 $group1"
$ns at 5.5 "$n5 leave-group $rcvr4 $group2"
$ns at 6.0 "$n6 leave-group $rcvr5 $group2"
$ns at 6.5 "$n7 leave-group $rcvr6 $group2"

# Schedule events
$ns at 0.5 "$cbr1 start"
$ns at 9.5 "$cbr1 stop"
$ns at 0.5 "$cbr2 start"
$ns at 9.5 "$cbr2 stop"
```

```
#post-processing
$ns at 10.0 "finish"
proc finish {}
{ global ns tf
$ns flush-trace
close $tf
exec nam mcast.nam
& exit 0
}
$ns set-animation-rate 3.0ms
$ns run
```

Exercise No. 9 Simulation and analysis of Ethernet LAN IEEE 802.3

To simulate and analysis of Ethernet LAN IEEE 802.3 using NS2 simulator.

Objectives

- Implement and simulate an Ethernet LAN using the IEEE 802.3 standard in NS2.
- Model network topologies representative of typical Ethernet LAN environments.
- Analyze the throughput of Ethernet LAN under various traffic loads.
- Measure latency and packet loss in Ethernet LAN communication.
- Evaluate collision rates and their impact on network performance.

Prerequisite / Tools Required:

Tools: NS2 Simulator

Procedure / Algorithm

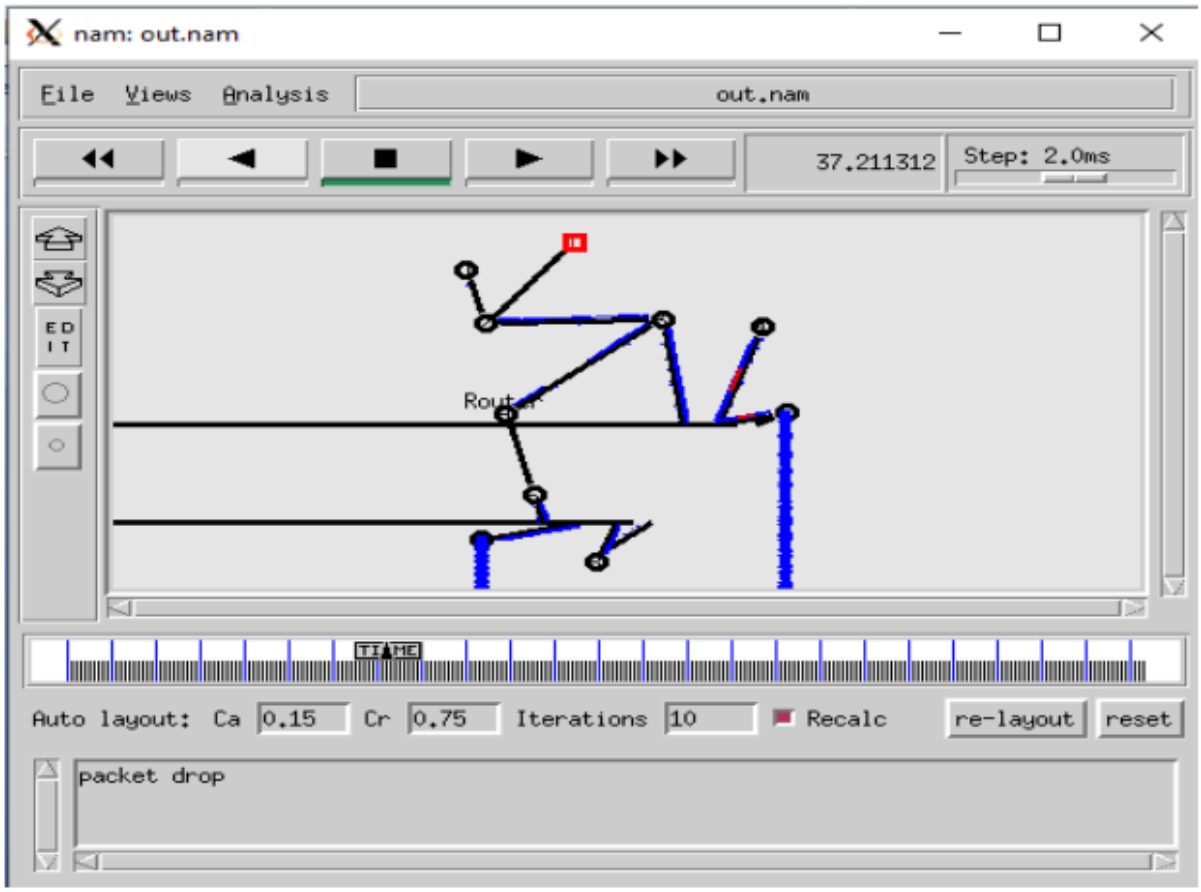
- Create a simulator object.
- Open a nam trace file and define finish procedure then close the trace file, and execute nam on trace file.
- Create 7 number of nodes.
- Create duplex links between the nodes and connect all nodes with same capacity .
- Define mode configuration assign MAC as CSMA/CD.
- Create bus like topology.
- Set and attach, FTP agent at node A and TCP agent at node C .
- Set E node as sink for both the traffic.
- Start transmission at 0.1 in node-0.
- Start transmission at 0.4 in node-5.
- Schedule events and run the program.

Sample Code

```
set ns [new Simulator]
#Open the NAM trace file

set nf [open prog.nam w]
$ns namtrace-all $nf
#Open the trace file
set nd [open prog.tr w]
$ns trace-all $nd
#Define a finish
```

```
procedure proc finish
{} {
global ns nf nd
$ns flushtrace
close
$nf close $nd
exec nam prog.nam
& exit 0
}
#Create 6 nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
#Create link between the nodes
$ns make-lan "$n0 $n1 $n2 $n3 $n4 $n5 $n6" 0.2Mb 40ms LL Queue/DropTail
Mac/802_3

#Setup a TCP connection
set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n5 $sink
$ns connect $tcp $sink

#Setup a FTP over a TCP connection
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 1.0 "$ftp start"
$ns at 5.0 "$ftp stop"
$ns at 5.5 "finish"
$ns run
```

Output:


nam: out.nam
File Views Analysis
out.nam
37.211312
Step: 2.0ms
EDIT
TIME
Auto layout: Ca 0.15 Cr 0.75 Iterations 10 Recalc
re-layout
reset
packet drop

Experiment 10: Simulation and analysis of Wireless LAN IEEE 802.11

To simulate and analysis of Wireless LAN IEEE 802.11 using NS2 simulator.

Objectives

- Implement and simulate a Wireless LAN using the IEEE 802.11 standard in NS2.
- Model network topologies representative of typical Wireless LAN environments.
- Analyze the throughput of Wireless LAN under various traffic loads.
- Measure latency and packet loss in Wireless LAN communication.
- Evaluate the impact of interference and signal attenuation on network performance.

Prerequisite / Tools Required:

Tools: NS2 Simulator

Procedure / Algorithm

- Create a simulator object.
- Open a nam trace file and define finish procedure then close the trace file, and execute nam on trace file.
- Create n number of nodes.
- Create duplex links between the nodes and connect all nodes with same capacity.
- Define mode configuration assign MAC as Mac/802_11.
- Create GOD topology.
- Set and attach, TCP agent.
- Set TCP Sink to nodes.
- Schedule events and run the program.
- Calculating the average throughput.

Sample Code:

```
Mac/802_11 set dataRate_ 1Mb

set val(chan) Channel/WirelessChannel ;# channel type

set val(prop) Propagation/TwoRayGround ;# radio-propagation model

set val(ant) Antenna/OmniAntenna ;# Antenna type

set val(ll) LL ;# Link layer type

set val(ifq) Queue/DropTail/PriQueue ;# Interface queue type set
```

```
val(ifqlen) 50 ;# max packet in ifq
set val(netif) Phy/WirelessPhy ;# network interface type
set val(mac) Mac/802_11 ;# MAC type
set val(nn) 15 ;# number of mobilenodes
set val(rp) AODV ;# routing protocol
set val(x) 800
set val(y) 800

# Creating simulation object set
ns [new Simulator] #creating
Output trace files
set f [open complexdcf.tr w]
$ns trace-all $f
set namtrace [open complexdcf.nam w]
$ns namtrace-all-wireless $namtrace $val(x)
$val(y) set f0 [open C_DCF_AT.tr w]
set topo [new Topography]
$topo load_flatgrid 800 800 #
Defining Global Variables
create-god $val(nn)
set chan_1 [new $val(chan)]

# setting the wireless nodes parameters
$ns node-config -adhocRouting $val(rp) \
-llType $val(ll) \
-macType $val(mac) \
-ifqType $val(ifq) \
-ifqLen $val(ifqlen) \
-antType $val(ant) \
```

```
-propType $val(prop) \
-phyType $val(netif) \
-topoInstance $topo \
-agentTrace OFF \
-routerTrace ON \
-macTrace ON \
-movementTrace OFF \
-channel $chan_1 \
proc finish {} {
global ns f f0 namtrace # global variables #
Closing the trace files
$ns flush-trace
close $namtrace
close $f0
exec nam -r 5m complexdcf.nam &   # Running the animator
exit 0
}

# Defining a procedure to calculate the throughout
proc record {} {
global sink1 sink3 sink7 sink10 sink11 f0
set ns [Simulator instance]
set time 0.5
set bw0 [$sink3 set bytes_]
set bw3 [$sink3 set bytes_]
set bw7 [$sink7 set bytes_]
set bw10 [$sink10 set bytes_]
set bw11 [$sink11 set bytes_]
set now [$ns now]
```

```
puts $f0 "$now [expr
($bw0+$bw3+$bw7+$bw10+$bw11)/$time*8/1000000]"

# Calculating the average throughput
$sink1 set bytes_ 0
$sink3 set bytes_ 0
$sink7 set bytes_ 0
$sink10 set bytes_ 0
$sink11 set bytes_ 0
$ns at [expr $now+$time] "record"
}

#Creating the wireless Nodes
for {set i 0} {$i < $val(nn) } {incr i} {
set n($i) [$ns node]
$n($i) random-motion 0 ;
}

#Setting the initial position for the nodes
for {set i 0} {$i < $val(nn)} {incr i} {
$ns initial_node_pos $n($i) 30+i*100
}
for {set i 0} {$i < $val(nn)} {incr i} {
$n($i) set X_ 0.0
$n($i) set Y_ 0.0
$n($i) set Z_ 0.0
}
```

```
# Making some nodes move in the topography
$ns at 0.0 "$n(0) setdest 100.0 100.0 3000.0"
$ns at 0.0 "$n(1) setdest 200.0 200.0 3000.0"
$ns at 0.0 "$n(2) setdest 300.0 200.0 3000.0"
$ns at 0.0 "$n(3) setdest 400.0 300.0 3000.0"
$ns at 0.0 "$n(4) setdest 500.0 300.0 3000.0"
$ns at 0.0 "$n(5) setdest 600.0 400.0 3000.0"
$ns at 0.0 "$n(6) setdest 600.0 100.0 3000.0"
$ns at 0.0 "$n(7) setdest 600.0 200.0 3000.0"
$ns at 0.0 "$n(8) setdest 600.0 300.0 3000.0"
$ns at 0.0 "$n(9) setdest 600.0 350.0 3000.0"
$ns at 0.0 "$n(10) setdest 700.0 100.0 3000.0"
$ns at 0.0 "$n(11) setdest 700.0 200.0 3000.0"
$ns at 0.0 "$n(12) setdest 700.0 300.0 3000.0"
$ns at 0.0 "$n(13) setdest 700.0 350.0 3000.0"
$ns at 0.0 "$n(14) setdest 700.0 400.0 3000.0"
$ns at 2.0 "$n(5) setdest 100.0 400.0 500.0"
$ns at 1.5 "$n(3) setdest 450.0 150.0 500.0"
$ns at 50.0 "$n(7) setdest 300.0 400.0 500.0"
$ns at 2.0 "$n(10) setdest 200.0 400.0 500.0"
$ns at 2.0 "$n(11) setdest 650.0 400.0 500.0"

#Creating receiving sinks with monitoring ability to monitor the incoming bytes
# LossMonitor objects are a subclass of agent objects that implement a traffic sink.
set sink1 [new Agent/LossMonitor]
set sink3 [new Agent/LossMonitor]
set sink7 [new Agent/LossMonitor]
set sink10 [new Agent/LossMonitor]
set sink11 [new Agent/LossMonitor]
```

```
$ns attach-agent $n(1) $sink1
$ns attach-agent $n(3) $sink3
$ns attach-agent $n(7) $sink7
$ns attach-agent $n(10) $sink10
$ns attach-agent $n(11) $sink11

# Setting TCP as the transmission protocol over the connections set
tcp0 [new Agent/TCP]
$ns attach-agent $n(0) $tcp0
set tcp2 [new Agent/TCP]
$ns attach-agent $n(2) $tcp2
set tcp4 [new Agent/TCP]
$ns attach-agent $n(4) $tcp4
set tcp5 [new Agent/TCP]
$ns attach-agent $n(5) $tcp5
set tcp9 [new Agent/TCP]
$ns attach-agent $n(9) $tcp9
set tcp13 [new Agent/TCP]
$ns attach-agent $n(13) $tcp13
set tcp6 [new Agent/TCP]
$ns attach-agent $n(6) $tcp6
set tcp14 [new Agent/TCP]
$ns attach-agent $n(14) $tcp14
set tcp8 [new Agent/TCP]
$ns attach-agent $n(8) $tcp8 set
tcp12 [new Agent/TCP]
$ns attach-agent $n(12) $tcp12 #
Setting FTP connections
set ftp9 [new Application/FTP]
```

```
$ftp9 attach-agent $tcp9
$ftp9 set type_ FTP
set ftp13 [new Application/FTP]
$ftp13 attach-agent $tcp13
$ftp13 set type_ FTP
set ftp6 [new Application/FTP]
$ftp6 attach-agent $tcp6
$ftp6 set type_ FTP
set ftp14 [new Application/FTP]
$ftp14 attach-agent $tcp14
$ftp14 set type_ FTP
set ftp8 [new Application/FTP]
$ftp8 attach-agent $tcp8
$ftp8 set type_ FTP
set ftp12 [new Application/FTP]
$ftp12 attach-agent $tcp12
$ftp12 set type_ FTP

#connecting the nodes
$ns connect $tcp0 $sink3
$ns connect $tcp5 $sink3
$ns connect $tcp2 $sink1
$ns connect $tcp4 $sink1
$ns connect $tcp9 $sink7
$ns connect $tcp13 $sink7
$ns connect $tcp6 $sink10
$ns connect $tcp14 $sink10
$ns connect $tcp8 $sink11
$ns connect $tcp12 $sink11
```

```
# Defining CBR procedure with the required parameters
proc attach-CBR-traffic { node sink size interval } {
set ns [Simulator instance]
set cbr [new Agent/CBR]
$ns attach-agent $node $cbr
$cbr set packetSize_ $size
$cbr set interval_ $interval
$ns connect $cbr $sink
return $cbr
}
set cbr0 [attach-CBR-traffic $n(0) $sink3 1000 .015] set
cbr1 [attach-CBR-traffic $n(5) $sink3 1000 .015] set
cbr2 [attach-CBR-traffic $n(2) $sink1 1000 .015] set
br3 [attach-CBR-traffic $n(4) $sink1 1000 .015]

# Setting the begining and ending time of each connection
$ns at 0.0 "record"
$ns at 20.0 "$cbr0 start"
$ns at 20.0 "$cbr2 start"
$ns at 800.0 "$cbr0 stop"
$ns at 850.0 "$cbr2 stop"
$ns at 30.0 "$cbr1 start"
$ns at 30.0 "$cbr3 start"
$ns at 850.0 "$cbr1 stop"
$ns at 870.0 "$cbr3 stop"
$ns at 25.0 "$ftp6 start"
$ns at 25.0 "$ftp14 start"
$ns at 810.0 "$ftp6 stop"
$ns at 860.0 "$ftp14 stop"
```

$ns at 35.0 "$ftp9 start"

$ns at 35.0 "$ftp13 start"

$ns at 830.0 "$ftp9 stop"

$ns at 889.0 "$ftp13 stop"

$ns at 40.0 "$ftp8 start"

$ns at 40.0 "$ftp12 start"

$ns at 820.0 "$ftp8 stop"

$ns at 890.0 "$ftp12 stop"

$ns at 900.0 "finish"

Runnning the simulation

puts "Start of simulation.."

$ns run

Output: