ANNAMALAI UNIVERSITY

FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

B.E. COMPUTER SCIENCE

(ARTIFICIAL INTELLIGENGENCE AND MACHINE LEARNINING)

V – SEMESTER

AICP507 – COMPUTER NETWORKS LAB

Name : ______________________________

Reg. No. : ______________________________

ANNAMALAI UNIVERSITY

FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

B.E. COMPUTER SCIENCE

(ARTIFICIAL INTELLIGENGENCE AND MACHINE LEARNINING)

V – SEMESTER

AICP507 – COMPUTER NETWORKS LAB

Certified that this is a bonafide record of work done by

Mr./Ms.____

Reg. No.____

of B.E. Computer Science(Artificial Intelligence and Machine Learning) in the AICP507 – Computer Networks Lab during the odd semester of the academic year 2021 – 2022.

Staff in-charge Internal Examiner External Examiner

Annamalai Nagar

Date: / / 2021

Ex: 1 Study of Network commands

DATE:

AIM :

To study the important network related commands.

C:\>ping: Ping is the most basic TCP/IP command, and it's the same as placing a phone call to your best friend. You pick up your telephone and dial a number, expecting your best friend to reply with ―Hello‖ on the other end. Computers make phone calls to each other over a network by using a Ping command. The Ping commands main purpose is to place a phone call to another computer on the network, and request an answer. Ping has 2 options it can use to place a phone call to another computer on the network. It can use the computers name or IP address.

C:\>arp –a: ARP is short form of address resolution protocol, It will show the IP address of your computer along with the IP address and MAC address of your router.

C:\>hostname: This is the simplest of all TCP/IP commands. It simply displays the name of your computer.

C:\>ipconfig: The ipconfig command displays information about the host (the computer your sitting at)computer TCP/IP configuration.

C:\>ipconfig /all: This command displays detailed configuration information about your TCP/IP connection including Router, Gateway, DNS, DHCP, and type of Ethernet adapter in your system.

C:\>Ipconfig /renew: Using this command will renew all your IP addresses that you are currently (leasing) borrowing from the DHCP server. This command is a quick problem solver if you are having connection issues, but does not work if you have been configured with a static IP address.

C:\>Ipconifg /release: This command allows you to drop the IP lease from the DHCP server.

C:\>ipconfig /flushdns: This command is only needed if you're having trouble with your networks DNS configuration. The best time to use this command is after networkconfiguration frustration sets in, and you really need the computer to reply with flushed.

C:\>nbtstat –a: This command helps solve problems with NetBIOS name resolution. (Nbt stands for NetBIOS over TCP/IP)

C:\>netdiag: Netdiag is a network testing utility that performs a variety of network diagnostic tests, allowing you to pinpoint problems in your network. Netdiag isn't installed by default, but can be installed from the Windows XP CD after saying no to the install. Navigate to the CD ROM drive letter and open the support\tools folder on the XPCD and click the setup.exe icon in the support\tools folder.

C:\>netstat: Netstat displays a variety of statistics about a computers active TCP/IP connections. This tool is most useful when you're having trouble with TCP/IP applications such as HTTP, and FTP.

C:\>nslookup: Nslookup is used for diagnosing DNS problems. If you can access a resource by specifying an IP address but not it's DNS you have a DNS problem.

C:\>pathping: Pathping is unique to Window's, and is basically a combination of the Ping and Tracert commands. Pathping traces the route to the destination address then launches a 25second test of each router along the way, gathering statistics on the rate of data loss along each hop.

C:\>route: The route command displays the computers routing table. A typicalcomputer, with a single network interface, connected to a LAN, with a router is fairly simple and generally doesn't pose any network problems. But if you're having troubleaccessing other computers on your network, you can use the route command to make sure the entries in the routing table are correct.

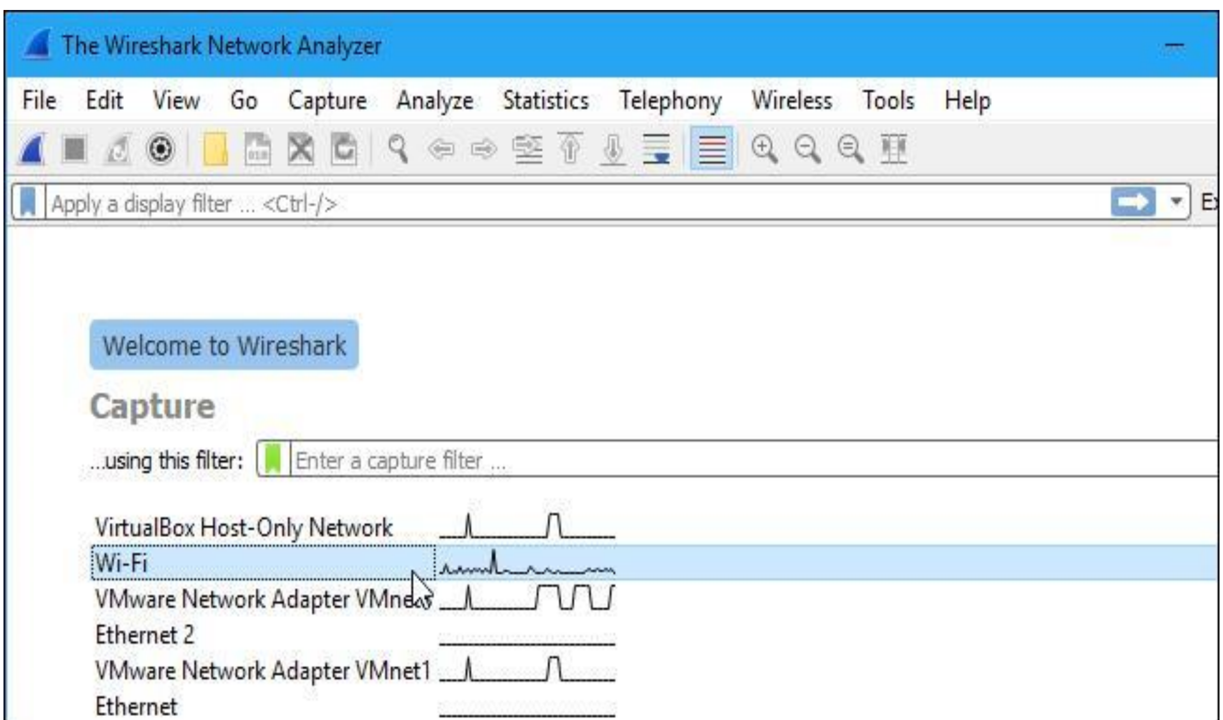
C:\>tracert: The tracert command displays a list of all the routers that a packet has to go through to get from the computer where tracert is run to any other computer on the internet.

Wireshark

Wireshark, a network analysis tool formerly known as Ethereal, captures packets in real time and display them in human-readable format. Wireshark includes filters, color coding, and other features that let you dig deep into network traffic and inspect individual packets.

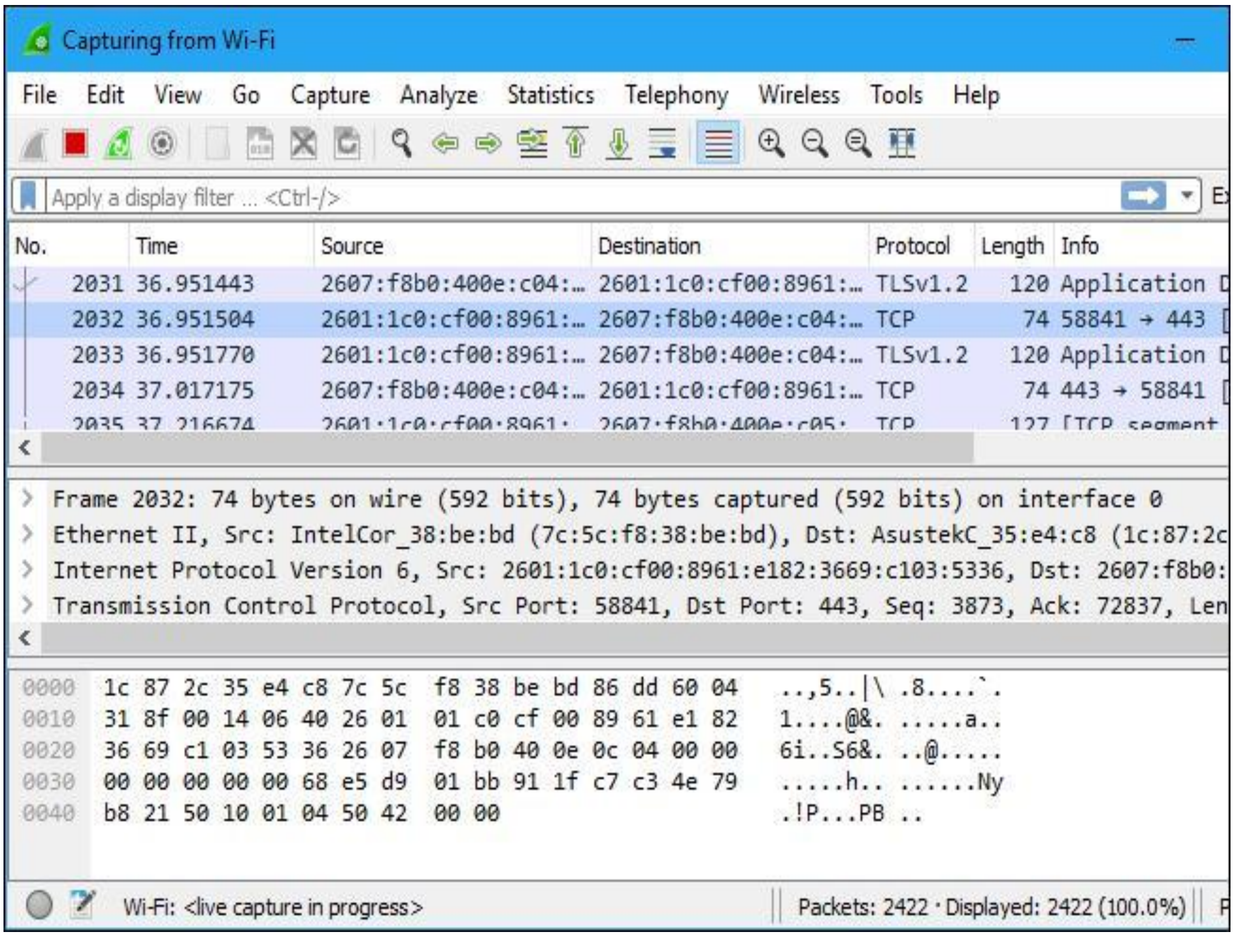
Capturing Packets

After downloading and installing Wireshark, you can launch it and double-click the name of a network interface under Capture to start capturing packets on that interface. For example, if you want to capture traffic on your wireless network, click your wireless interface. You can configure advanced features by clicking Capture > Options, but this isn't necessary for now.

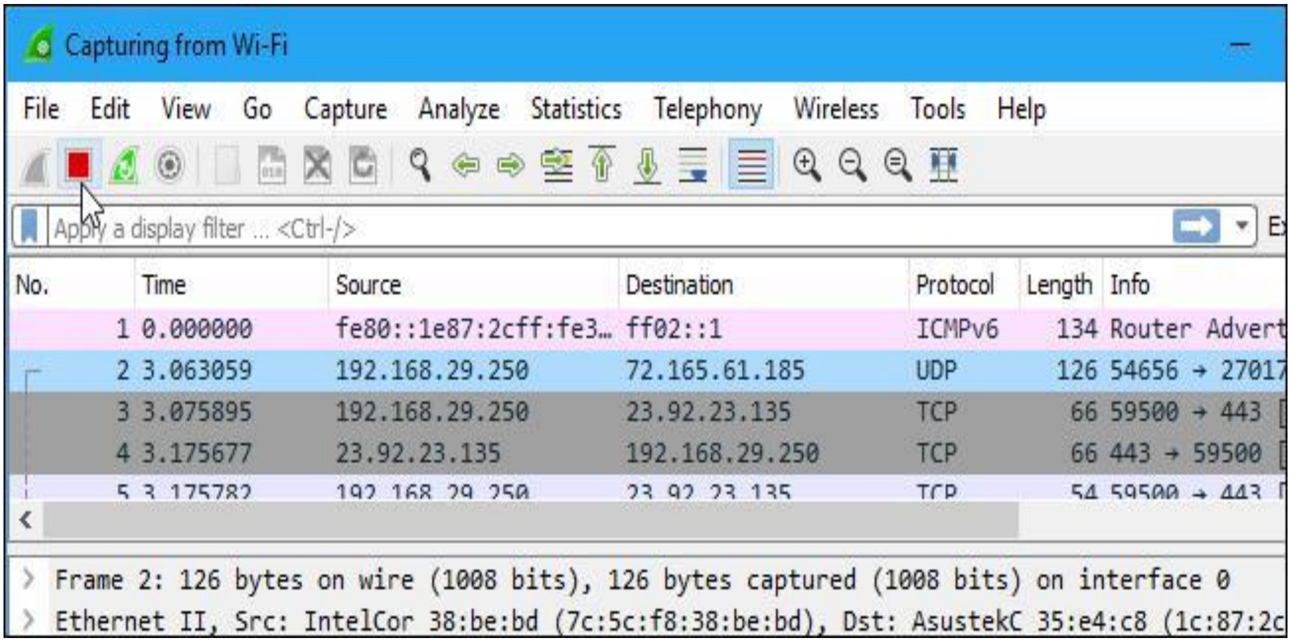


As soon as you click the interface's name, you'll see the packets start to appear in real time. Wireshark captures each packet sent to or from your system.

If you have promiscuous mode enabled—it's enabled by default—you'll also see all the other packets on the network instead of only packets addressed to your network adapter. To check if promiscuous mode is enabled, click Capture > Options and verify the ―Enable promiscuous mode on all interfaces‖ checkbox is activated at the bottom of this window.



Click the red ―Stop‖ button near the top left corner of the window when you want to stop capturing traffic.



Result :

Thus the Network Commands are studied in detail.

EX : 2 Implementation of ECHO server

DATE:

Aim :

To implement echo server and client using TCP sockets.

Algorithm:

CLIENT SIDE

1. Create a socket which binds the Ip address of server and the port address to acquire service.
2. After establishing connection send a data to server.
3. Receive and print the same data from server.
4. Close the socket.

SERVER SIDE

1. Create a server socket to activate the port address.
2. Create a socket for the server socket which accepts the connection.
3. After establishing connection receive the data from client.
4. Print and send the same data to client.
5. Close the socket.

pyEchoClient.py

```
import socket

HOST = '127.0.0.1'

PORT = 9999

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

s.connect((HOST, PORT))

str ='Hello, world'

b    = str.encode('utf-

8') s.send(b)

data = s.recv(1024)

s.close()

print ('Received', data)
```

pyEchoServere.py

```
import socket

HOST = '127.0.0.1'

PORT = 9999

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

s.bind((HOST, PORT))

s.listen(1)

conn, addr = s.accept()

print ('Connected by', addr)

while 1:

    data = conn.recv(1024)

    print(data)

    if not data: break

    conn.send(data)

conn.close()
```

Sample Input/Output :

D:\NW_LAB>py pyEchoerver.py Connected by ('127.0.0.1', 51252) 'Hello, world'	D:\NW_LAB>py pyEchoClient.py Received 'Hello, world'

Result:

Thus data from client to server is echoed back to the client.

EX :3 Implementation of TCP client/server chat

DATE:

Aim :

To implement chatting between client and server using TCP.

Algorithm:

Server

Step1: Start the program and create server and client sockets.

Step2: Use input streams to get the message from user.

Step3: Use output streams to send message to the client.

Step4: Wait for client to display this message and write a new one to be displayed by the server.

Step5: Display message given at client using input streams read from socket.

Step6: Stop the program.

Client

Step1: Start the program and create a client socket that connects to the required host and port.

Step2: Use input streams read message given by server and print it.

Step3: Use input streams; get the message from user to be given to the server.

Step4: Use output streams to write message to the server.

Step5: Stop the program.

<u>Pyserver.py</u>

```
import socket

def server_program(): # get the hostname

    host = socket.gethostname()

    port = 5000          # initiate port no above 1024

    server_socket = socket.socket()              # get instance

    #     look closely. The bind() function takes tuple as

    argument server_socket.bind((host, port))

    # bind host address and porttogether

    # configure how many client the server can listen simultaneously
```

```
    server_socket.listen(2)

    conn, address = server_socket.accept()

    # accept new connection print("Connection from: " +

    str(address)) while True:

    # receive data stream. it won't accept data packet greater than 1024

        bytes data = conn.recv(1024).decode()

        if not data:

            # if data is not received

            break break

        print("from connected user: " + str(data))

        data = input(' -> ')

        conn.send(data.encode()) # send data to the client

    conn.close() # close the connection

if __name__ == '__main__':

    server_program()
```

<u>pyclient.py</u>

```
import socket

def client_program():

    print('type bye to terminate')

    host = socket.gethostname() # as both code is running on same

    pc port = 5000 # socket server port number client_socket =

    socket.socket() # instantiate

    client_socket.connect((host, port)) # connect to the

    server message = input(" -> ") # take input while

    message.lower().strip() != 'bye':
```

```
        client_socket.send(message.encode()) # send message
        data = client_socket.recv(1024).decode() # receive
        response print('Received from server: ' + data)# show in
        terminal message = input(" -> ") # again take input
    client_socket.close() # close the connection
if __name__ == '__main__':
    client_program()
```

Sample Input/Output :

D:\NW_LAB>py pyclient.py type bye to terminate ->hai server Received from server: hai client -> how r u Received from server: imfine . how r u -> fine thanks Received from server: bye -> bye D:\NW_LAB>	D:\NW_LAB>py pyserver.py Connection from: ('192.168.56.1', 50141) from connected user: hai server ->hai client from connected user: how r u ->im fine . how r u from connected user: fine thanks -> bye D:\NW_LAB>

Result :

Thus chatting between client and server is implemented successfully.

Ex :4 **Executing Remote Command**

DATE:

Aim :

To execute command from the remote host.

Algorithm :

Step1: Create a server(sender) socket at the server side.

Step2: Create a socket at the client side and the connection is set to accept by the server socket using the accept () method.

Step3: In the client side the remote command to be executed is given as input.

Step4: The command is obtained using the WMI

Step5: execute the command using conn.Win32_Process.

pip install wmi

How do I use it?

```
import wmi

c = wmi.WMI()

for s in c.Win32_Service(StartMode="Auto", State="Stopped"):

ifraw_input("Restart %s? " % s.Caption).upper() == "Y":

s.StartService()
```

RemoteSender.py

```
#     importing socket

module import socket

UDP_IP = "localhost"

UDP_PORT = 8080 MESSAGE

= "notepad" print ("message:",

MESSAGE)

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

```
sock.sendto(bytes(MESSAGE, "utf-8"), (UDP_IP, UDP_PORT))
```

<u>RemoteReceiver.py</u>

```
# importing socket
module import socket
import wmi
UDP_IP = "localhost"
UDP_PORT = 8080
sock = socket.socket(socket.AF_INET,
socket.SOCK_DGRAM) sock.bind((UDP_IP, UDP_PORT)) #
while True: # buffer size is 1024 bytes
    data, addr = sock.recvfrom(1024)
    str=data.decode("utf-8")
    print ("Received message:",
    str) print(" opening ", str);
conn = wmi.WMI()
pid, returnval= conn.Win32_Process.Create(CommandLine=str)
```

Sample Input /Output :

D:\NW_LAB>py RemoteSender.py message: notepad	D:\NW_LAB>py RemoteReceiver.py Received message: notepad opening notepad

Result :

Thus the command from remote host executed successfully.

Ex:5 ARP/RARP implementation

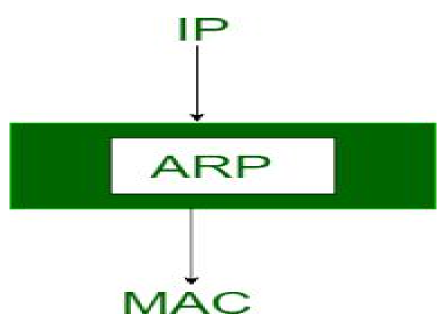
DATE:

Aim :

To know the Physical Address of the node when communication taken place between host to host using ARP.

Introduction :

Most of the computer programs/applications use **logical address (IP address)** to send/receive messages, however the actual communication happens over the **physical address (MAC address)** i.e from layer 2 of OSI model. So our mission is to get the destination MAC address which helps in communicating with other devices. This is where ARP comes into the picture, its functionality is to translate IP address to physical address.



The acronym ARP stands for **Address Resolution Protocol** which is one of the most important protocols of the Network layer in the OSI model.

Note: ARP finds the hardware address, also known as Media Access Control (MAC) address, of a host from its known IP address.

Algorithm:

Step 1: Initialize the string of ip addresses.

Step 2: For every ip address, assign an ethernetaddress .

Step 3: To Perform ARP, enter the ip address.

Step 4: The equivalent ethernet address is displayed.

Step 5: If the ethernet address is not found, then 'not found' message is displayed.

Step 6: To Perform RARP, enter the ethernet address.

Step 7: The equivalent ip address is displayed.

Step 8: If the ip address is not found, then 'not found' message is displayed.

Step 9: Provide option to perform both ARP and RARP.

Step 10:Choose Exit option to terminate the program.

ARPClient.py

```
#     importing socket
module import socket
UDP_IP = "localhost"
UDP_PORT = 8080 MESSAGE =
"172.16.1.8" print ("message:",
MESSAGE)
sock    =    socket.socket(socket.AF_INET,    socket.SOCK_DGRAM)
sock.sendto(bytes(MESSAGE, "utf-8"), (UDP_IP, UDP_PORT))
```

ARPserver.py

```
import socket
UDP_IP = "localhost"
UDP_PORT = 8080
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((UDP_IP, UDP_PORT))
ip=["172.16.1.9","172.16.1.8"]
mac=["6A:08:AA:C2","8A:BC:E3:FA"]
while True:
    # buffer size is 1024 bytes
    data, addr = sock.recvfrom(1024)
    str1 = data.decode('utf-8')
    l = len(data)
    if l != 0:
        print("Received message:", str1)
        break
for x in ip:
    if str1 in x:
        ind=ip.index(str1)
```

```
print("the MAC address is: ",mac[ind])
```

Sample Input / Output:

D:\NW_LAB>py arp_client.py message: 172.16.1.8

D:\NW_LAB>py arp_server.py Received message: 172.16.1.8

the MAC address is: 8A:BC:E3:FA

Result :

Thus the MAC address is displayed using ARP.

EX : 6 Downloading a Web Page using Socket

DATE:

Aim :

To download the Web page using Socket.

procedure:

Import urllib.request package. If not available download using the command **pip install urllib**

Coding.py

```
import urllib.request

urllib.request.urlretrieve ("http://www.example.org/", "webpage.html")


# read local file

for line in open('webpage.html'): print(line.strip())
```

Sample Input /output:

```
D:\NW_LAB>py page_down.py
<!doctype html>
<html>
<head>
<title>Example Domain</title>
<meta charset="utf-8" />
<meta http-equiv="Content-type" content="text/html; charset=utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1" />
<style type="text/css">
body {
background-color: #f0f0f2; margin: 0;
padding: 0;
font-family: -apple-system, system-ui, BlinkMacSystemFont, "Segoe UI", "Open Sans", "Helvetica Neue", Helvetica, Arial, sans-serif;
}
```

```
div {
width: 600px; margin: 5em auto; padding: 2em;
background-color: #fdfdff; border-radius: 0.5em;
box-shadow: 2px 3px 7px 2px rgba(0,0,0,0.02);
}
a:link, a:visited { color: #38488f;
text-decoration: none;
}
@media (max-width: 700px) { div {
margin: 0 auto; width: auto;
}
}
</style>
</head>

<body>
<div>
<h1>Example Domain</h1>
<p>This domain is for use in illustrative examples in documents. You may use this domain in literature without prior coordination or asking for permission.</p>
<p><a href="https://www.iana.org/domains/example">More information...</a></p>
</div>
</body>
</html>
```

Result :

Thus the Web Page downloaded successfully using python socket.

Ex :7 **Encryption /Decryption**

DATE:

Aim :

To Encrypt and decrypt the message.

Introduction :

The Caesar Cipher technique is one of the earliest and simplest method of encryption technique. It's simply a type of substitution cipher, i.e., each letter of a given text is replaced by a letter some fixed number of positions down the alphabet. For example with a shift of 1, A would be replaced by B, B would become C, and so on. The method is apparently named after Julius Caesar, who apparently used it to communicate with his officials.

Thus to cipher a given text we need an integer value, known as shift which indicates the number of position each letter of the text has been moved down.

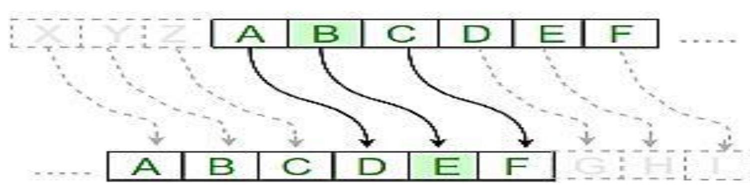
The encryption can be represented using modular arithmetic by first transforming the letters into numbers, according to the scheme, A = 0, B = 1,…, Z = 25. Encryption of a letter by a shift n can be described mathematically as.

$$E_n(x) = (x + n) mod\ 26$$

(Encryption Phase with shift n)

$$D_n(x) = (x - n) mod\ 26$$

(Decryption Phase with shift n)



Algorithm:

For Encryption

1. Read the string to be encrypted.
2. Input the shift positions
3. If the character in the text is uppercase then
4. cipher = cipher + chr((ord(char) + shift - 65) % 26 + 65)
5. else

6. cipher = cipher + chr((ord(char) + shift - 97) % 26 + 97)
7. print cipher

For Decryption:

1. Read the ciphertext to be decrypted.
2. Input the shift positions
3. If the character in the text is uppercase then
4. cipher = cipher + chr((ord(char) - shift - 65) % 26 + 65)
5. else
6. cipher = cipher + chr((ord(char) -shift - 97) % 26 + 97)
7. print original text

<u>Encrypt.py</u>

```
def encrypt(string, shift):
    cipher = ''
    for char in string:
        if char == ' ':
            cipher = cipher + char
        elif char.isupper():
            cipher = cipher + chr((ord(char) + shift - 65) % 26 + 65)
        else:
            cipher = cipher + chr((ord(char) + shift - 97) % 26 + 97)
    return cipher
text = input("enter string: ")
s = int(input("enter shift number: "))
print("original string: ", text)
print("after encryption: ", encrypt(text, s))
```

Decrypt.py

```
def encrypt(string, shift):
    cipher = ''
    for char in string:
        if char == ' ':
            cipher = cipher + char
        elif char.isupper():
            cipher = cipher + chr((ord(char) - shift - 65) % 26 + 65)
        else:
            cipher = cipher + chr((ord(char) -shift - 97) % 26 + 97)
    return cipher
text = input("enter string: ")
s = int(input("enter shift number: "))
print("original string: ", text)
print("after encryption: ", encrypt(text, s))
```

Sample Input / Output :

G:\NW\NW_LAB>py encrypt.py enter string: Annamalai University enter shift number: 4 original string: Annamalai University after encryption: ErreqepemYrmzivwmxc	G:\NW\NW_LAB>py decrypt.py enter string: ErreqepemYrmzivwmxc enter shift number: 4 original string: ErreqepemYrmzivwmxc after decryption: Annamalai University

Result :

Thus the message is Encrypted and Decrypted Successfully.

Ex : 8 **Study of Network Simulator**

DATE:

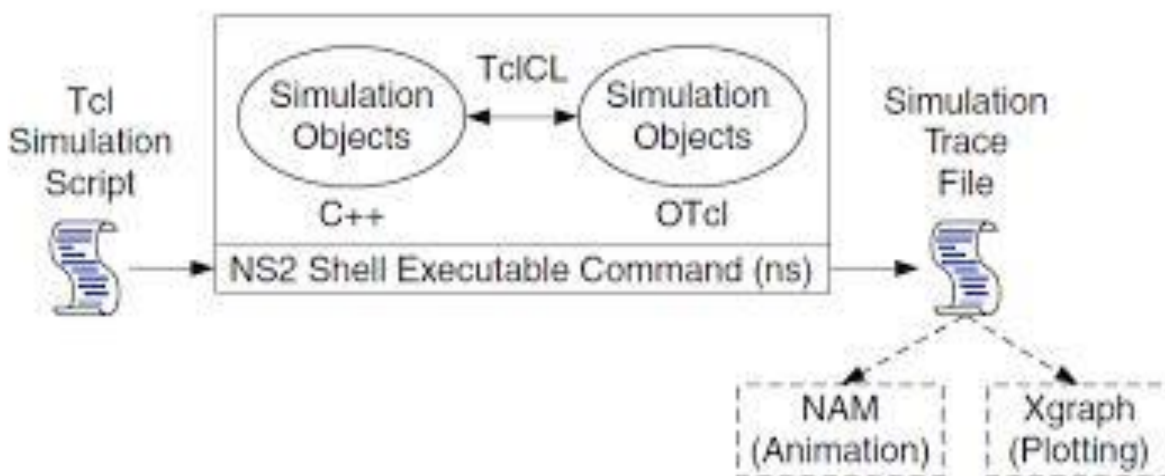
Aim :

To study in detail about the Network Simulator.

Theory :

Network Simulator (Version 2), widely known as NS2, is simply an event driven simulation tool that has proved useful in studying the dynamic nature of communication networks. Simulation of wired as well as wireless network functions and protocols (e.g., routing algorithms, TCP, UDP) can be done using NS2. In general, NS2 provides users with a way of specifying such network protocols and simulating their corresponding behaviors. Due to its flexibility and modular nature, NS2 has gained constant popularity in the networking research community since its birth in 1989. Ever since, several revolutions and revisions have marked the growing maturity of the tool, thanks to substantial contributions from the players in the field. Among these are the University of California and Cornell University who developed the REAL network simulator,1 the foundation which NS is based on. Since 1995 the Defense Advanced Research Projects Agency (DARPA) supported development of NS through the Virtual Inter Network Testbed (VINT) project . Currently the National Science Foundation (NSF) has joined the ride in development. Last but not the least, the group of Researchers and developers in the community are constantly working to keep NS2 strong andversatile.

BasicArchitecture :



BASIC Architecture of NS

Figure above shows the basic architecture of NS2. NS2 provides users with an executable command ns which takes on input argument, the name of a Tcl simulation scripting file. Users are feeding the name of a Tcl simulation script (which sets up a simulation) as an input argument of an NS2 executable command ns.

In most cases, a simulation trace file is created, and is used to plot graph and/or to create animation. NS2 consists of two key languages: C++ and Object-oriented Tool Command Language (OTcl). While the C++ defines the internal mechanism (i.e., a backend) of the simulation objects, the OTcl sets up simulation by assembling and configuring the objects as well as scheduling discrete events (i.e., a frontend).

The C++ and the OTcl are linked together using TclCL. Mapped to a C++ object, variables in the OTcl domains are sometimes referred to as handles. Conceptually, a handle (e.g., n as a Node handle) is just a string (e.g.,_o10) in the OTcl domain, and does not contain any functionality. Instead, the functionality (e.g., receiving a packet) is defined in the mapped C++ object (e.g., of class Connector). In the OTcl domain, a handle acts as a frontend which interacts with users and other OTclobjects. It may defines its own procedures and variables to facilitate the interaction. Note that the member procedures and variables in the OTcl domain are called instance procedures (instprocs) and instance variables (instvars), respectively. Before proceeding further, the readers are encouraged to learn C++ and OTcl languages. We refer the readers to [14] for the detail of C++, while a brief tutorial of Tcl and OTcl tutorial.

NS2 provides a large number of built-in C++ objects. It is advisable to use these C++ objects to set up a simulation using a Tcl simulation script. However, advance users may find these objects insufficient. They need to develop their own C++ objects, and use aOTcl configuration interface to put together these objects. After simulation, NS2 outputs either text-based or animation-based simulation results. To interpret these results graphically and interactively, tools such as NAM (Network AniMator) and XGraph are used. To analyze a particular behaviour of the network, users can extract a relevant subset of text-based data and transform it to a more conceivable presentation.

Concept Overview:

NS uses two languages because simulator has two different kinds of things it needs to do. On one hand,detailed simulations of protocols requires a systems programming language which can efficiently manipulate bytes, packet headers, and implement algorithms that run over large data sets. For these tasks run-time speed is important and turn-around time (run simulation, find bug, fix bug, recompile, re-run) is less important. On the other hand, a large part of network research involves slightly varying parameters or configurations, or quickly exploring a number of scenarios.

In these cases, iteration time (change the model and re-run) is more important. Since configuration runs once (at the beginning of the simulation), run-time of this part of the task is less important. ns meets both of these needs with two languages, C++ and OTcl.

Tcl scripting

Tcl is a general purpose scripting language. [Interpreter]

- Tcl runs on most of the platforms such as Unix, Windows, andMac.
- The strength of Tcl is itssimplicity.
- It is not necessary to declare a data type for variable prior to theusage.

Basics of TCL

Syntax: command arg1 arg2 arg3

Hello World!

putsstdout{Hello, World!} Hello, World!

Variables Command Substitution

set a 5 set len [string length foobar]

set b $a set len [expr [string length foobar] + 9]

Wired TCL Script Components

Create the event scheduler

Open new files & turn on the tracing Create the nodes

Setup the links

Configure the traffic type (e.g., TCP, UDP, etc)

Set the time of traffic generation (e.g., CBR, FTP)

Terminate the simulation

NS Simulator Preliminaries.

1. Initialization and termination aspects of the nssimulator.
2. Definition of network nodes, links, queues andtopology.
3. Definition of agents and ofapplications.
4. The nam visualizationtool.
5. Tracing and randomvariables.

Initialization and Termination of TCL Script in NS-2

An ns simulation starts with the command

set ns [new Simulator]

Which is thus the first line in the tcl script. This line declares a new variable as using the set command, you can call this variable as you wish, In general people declares it as ns because it is an instance of the Simulator class, so an object the code[new Simulator] is indeed the installation of the class Simulator using the reserved word new.

In order to have output files with data on the simulation (trace files) or files used for visualization (nam files), we need to create the files using ―open command:

```
#Open the Trace file
        set tracefile1 [open out.tr w]
        $ns trace-all $tracefile1
 #Open the NAM trace file

    setnamfile [open out.nam w]
    $ns namtrace-all $namfile
```

The above creates a dta trace file called out.tr and a nam visualization trace file called out.nam.Within the tcl script, these files are not called explicitly by their names, but instead by pointers that are declared above and called ―tracefile1 and ―namfile respectively. Remark that they begins with a # symbol. The second line open the file ―out.tr to be used for writing, declared with the letter ―w. The third line uses a simulator method called trace-all that have as parameter the name of the file where the traces will go.

Define a "finish" procedure

Proc finish { } {
global ns tracefile1 namfile
$ns flush-trace
Close $tracefile1
Close $namfile
Exec namout.nam&
Exit 0
}

Definition of a network of links and nodes

The way to define a node is

set n0 [$ns node]

Once we define several nodes, we can define the links that connect them. An example of a definition of a link is:

$ns duplex-link $n0 $n2 10Mb 10ms DropTail

Which means that $n0 and $n2 are connected using a bi-directional link that has 10ms of propagation delay and a capacity of 10Mb per sec for each direction.

To define a directional link instead of a bi-directional one, we should replace ―duplex-link by―simplex-link.

In ns, an output queue of a node is implemented as a part of each link whose input is that node. We should also define the buffer capacity of the queue related to each link. An example would be:

#set Queue Size of link (n0-n2) to 20
$ns queue-limit $n0 $n2 20

FTP over TCP

TCP is a dynamic reliable congestion control protocol. It uses Acknowledgements created by the destination to know whether packets are well received.

There are number variants of the TCP protocol, such as Tahoe, Reno, NewReno, Vegas. The type of agent appears in the first line:

settcp [new Agent/TCP]

The command **$ns attach-agent $n0 $tcp**defines the source node of the tcp connection. The command **set sink [new Agent /TCPSink]** Defines the behavior of the destination node of TCP and assigns to it a pointer called sink.

#Setup a UDP
connection set
udp [new
Agent/UDP]
$ns attach-agent
$n1 $udp set null
[new Agent/Null]
$ns attach-agent $n5 $null
$ns connect $udp $null
$udp set fid_2

#setup a CBR over UDP connection
The below shows the definition of a CBR application using a UDP agent
The command **$ns attach-agent $n4 $sink** defines the destination node. The command **$ns connect**
$tcp $sink finally makes the TCP connection between the source and destination nodes.

setcbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set packetsize_ 100
$cbr set rate_ 0.01Mb
$cbr set random_ false

TCP has many parameters with initial fixed defaults values that can be changed if mentioned explicitly. For example, the default TCP packet size has a size of 1000bytes.This can be changed to another value, say 552bytes, using the command **$tcp set packetSize_ 552**.
When we have several flows, we may wish to distinguish them so that we can identify them with different colors in the visualization part. This is done by the command **$tcp set fid_ 1** that assigns to theTCPconnectionaflowidentificationof―1.Weshalllatergivetheflowidentificationof―2‖to the UDPconnection.

Result:
Thus the Network Simulator is studied in detail.

Ex :9 **Network Topology**

DATE:

Aim :

To create scenario and study the performance of token bus, token Ring and token Star protocol through simulation.

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.

Token ring is a LAN protocol operating in the MAC layer. Token ring is standardized as per IEEE 802.5. Token ring can operate at speeds of 4mbps and 16 mbps. The operation of token ring is as follows: When there is no traffic on the network a simple 3-byte token circulates the ring. If the token is free (no reserved by a station of higher priority as explained later) then the station may seize the token and start sending the data frame. As the frame travels around the ring ach station examines the destination address and is either forwarded (if the recipient is another node) or copied. After copying4 bits of the last byte is changed. This packet then continues around the ring till it reaches the originating station. After the frame makes a round trip the sender receives the frame and releases a new token onto the ring.

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 :

Token Bus

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.

Token Ring :

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 to form a Ring Topology.
6. Setup TCP Connection between n(1) and n(3)
7. Apply CBR Traffic over TCP
8. Schedule events and run the program.

Token Star :

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 :

Token Bus

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

#Open the nam trace file
setnf [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 namout.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]

#CreateLan 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
```

Token Ring

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

#Open the nam trace file
setnf [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
        execnamout.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]
```

```
set n5 [$ns node]

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

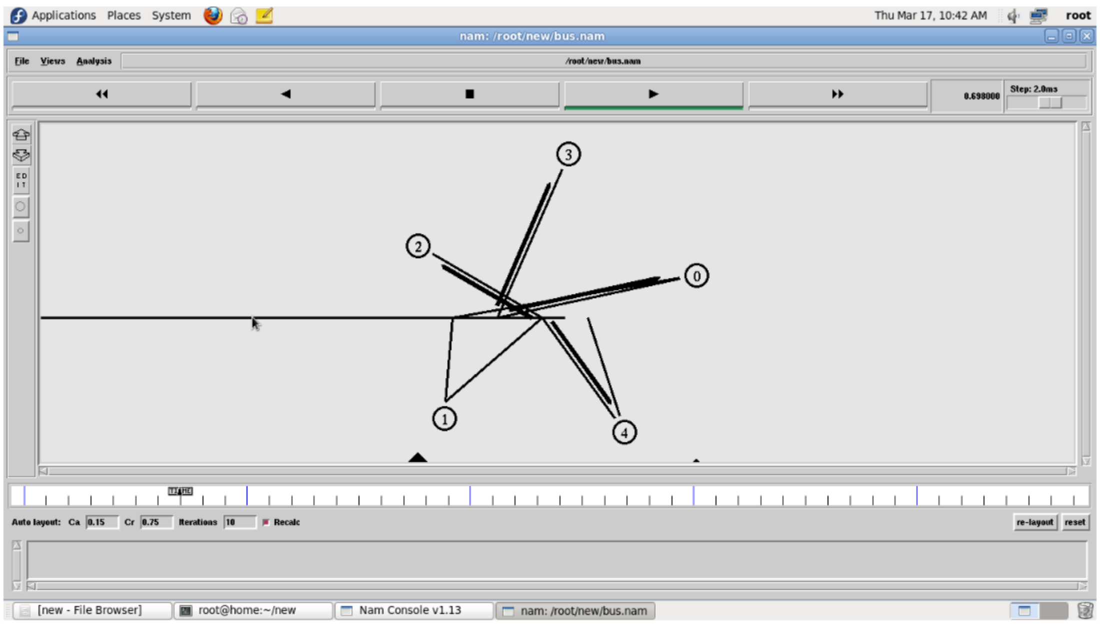
Token Star :

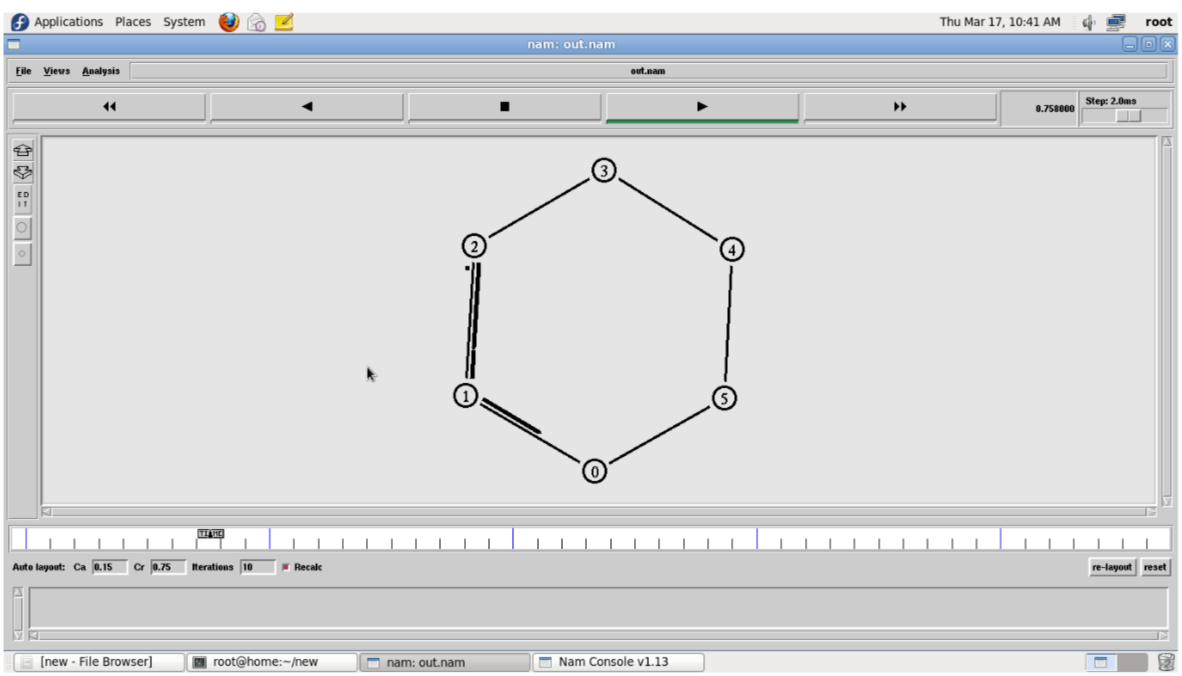
```
#Create a simulator object
set ns [new Simulator]
```

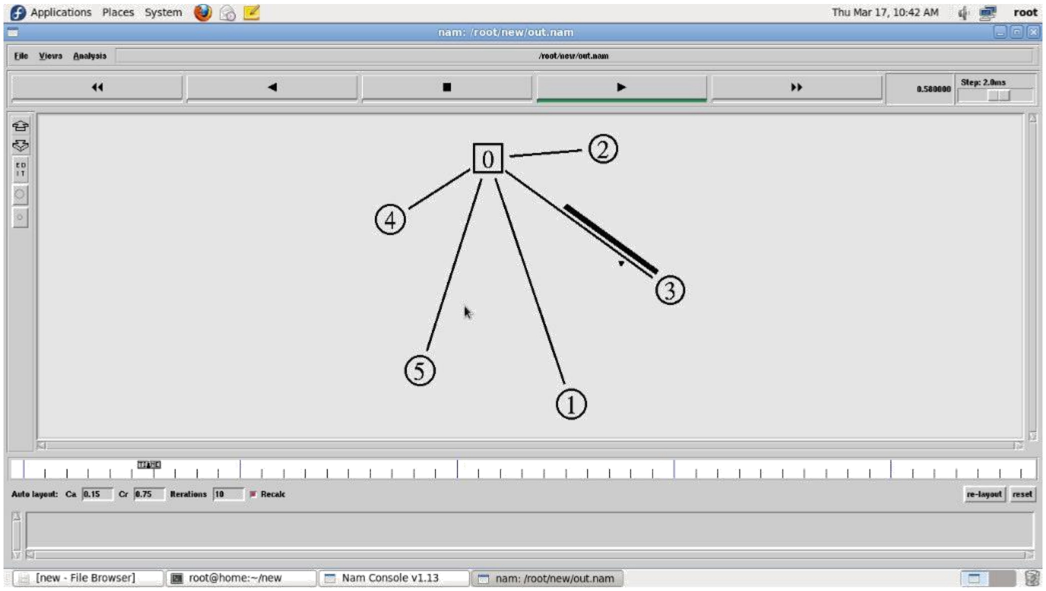
```
#Open the nam trace file
setnf [open out.nam w]
$ns namtrace-all $nf

#Define a 'finish' procedure
proc finish {} {
    global ns nf
    $ns flush-trace
    #Close the trace file
    close $nf
    #Executenam on the trace file
    execnamout.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
```

Sample Input / Output :







Result :

Thus the Bus, Ring and star Topology was simulated and studied.

Ex :10(a)

Simulation of Go Back N Protocol

DATE:

Aim:

To Simulate and to study of Go Back N protocol

Theory:

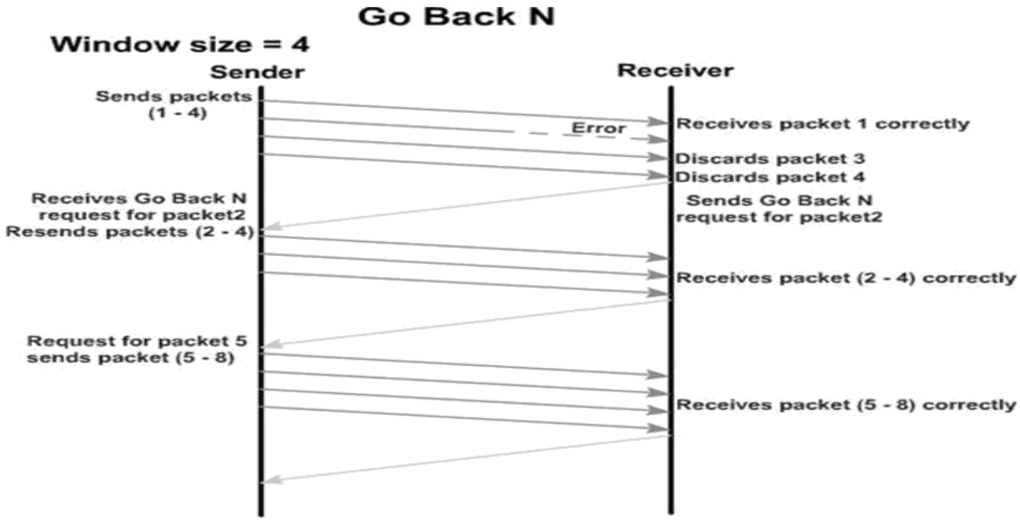
Go Back N is a connection oriented transmission. The sender transmits the frames continuously. Each frame in the buffer has a sequence number starting from 1 and increasing up to the window size. The sender has a window i.e. a buffer to store the frames. This buffer size is the number of frames to be transmitted continuously. The size of the window depends on the protocol designer.

Operations:

1. A station may send multiple frames as allowed by the window size.
2. Receiver sends an ACK i if frame i has an error. After that, the receiver discards all incoming frames until the frame with error is correctly retransmitted.
3. If sender receives an ACK i it will retransmit frame i and all packets i+1, i+2,... which have been sent, but not been acknowledged.

Algorithm for Go BackN :

1. The source node transmits the frames continuously.
2. Each frame in the buffer has a sequence number starting from 1 and increasing up to the window size.
3. The source node has a window i.e. a buffer to store the frames. This buffer size is the number of frames to be transmitted continuously.
4. The size of the window depends on the protocol designer.
5. For the first frame, the receiving node forms a positive acknowledgement if the frame is received without error.
6. If subsequent frames are received without error (up to window size) cumulative positive acknowledgement is formed.
7. If the subsequent frame is received with error, the cumulative acknowledgment error-free frames are transmitted. If in the same window two frames or more frames are received with error, the second and the subsequent error frames are neglected. Similarly even the frames received without error after the receipt of a frame with error are neglected.



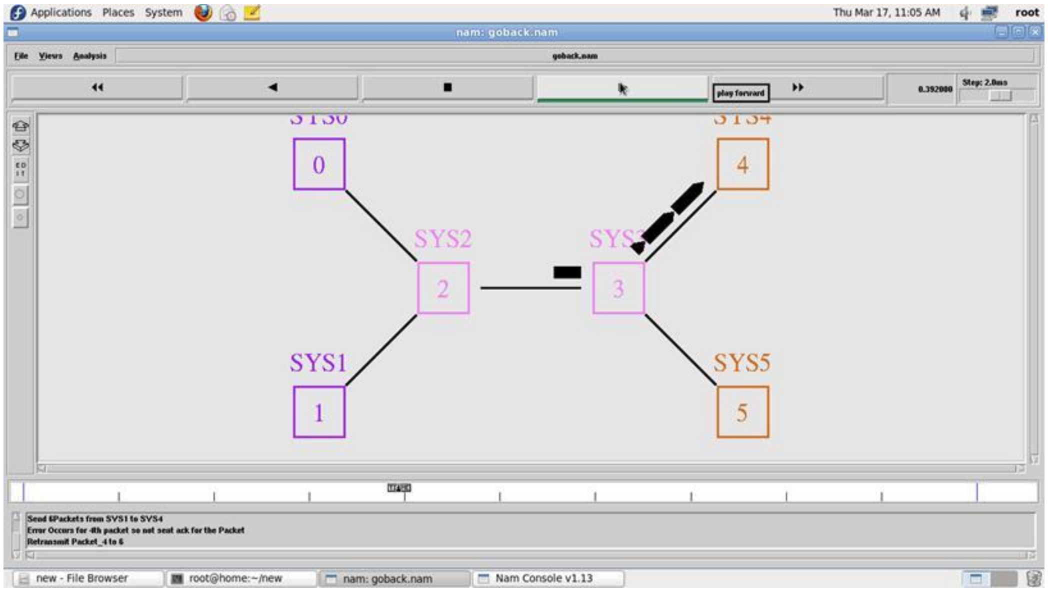
8. The source node retransmits all frames of window from the first error frame.
9. If the frames are errorless in the next transmission and if the acknowledgment is error free, the window slides by the number of error-free frames being transmitted.
10. If the acknowledgment is transmitted with error, all the frames of window at source are retransmitted, and window doesn’t slide.
11. This concept of repeating the transmission from the first error frame in the window is called as GOBACKNtransmission flow control protocol.

Program :

```
#send packets one by one set ns [new Simulator] 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 color "purple"
$n1 color "purple"
$n2 color "violet"
$n3 color "violet"
$n4 color "chocolate"
$n5 color "chocolate"
$n0 shape box ;
$n1 shape box ;
$n2 shape box ;
$n3 shape box ;
$n4 shape box ;
$n5 shape box ;
$ns at 0.0 "$n0 label SYS0"
$ns at 0.0 "$n1 label SYS1"
$ns at 0.0 "$n2 label SYS2"
$ns at 0.0 "$n3 label SYS3"
$ns at 0.0 "$n4 label SYS4"
$ns at 0.0 "$n5 label SYS5" set nf [open goback.nam w]
$ns namtrace-all $nf set f [open goback.tr w]
```

```
$ns trace-all $f
$ns duplex-link $n0 $n2 1Mb 20ms DropTail
$ns duplex-link-op $n0 $n2 orient right-down
$ns queue-limit $n0 $n2 5
$ns duplex-link $n1 $n2 1Mb 20ms DropTail
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link $n2 $n3 1Mb 20ms DropTail
$ns duplex-link-op $n2 $n3 orient right
$ns duplex-link $n3 $n4 1Mb 20ms DropTail
$ns duplex-link-op $n3 $n4 orient right-up
$ns duplex-link $n3 $n5 1Mb 20ms DropTail
$ns duplex-link-op $n3 $n5 orient right-down Agent/TCP set_nam_tracevar_true
settcp [new Agent/TCP]
$tcp set fid 1
$ns attach-agent $n1 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n4 $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp

$ns at 0.05 "$ftp start"
$ns at 0.06 "$tcp set windowlnit 6"
$ns at 0.06 "$tcp set maxcwnd 6"
$ns at 0.25 "$ns queue-limit $n3 $n4 0"
$ns at 0.26 "$ns queue-limit $n3 $n4 10"
$ns at 0.305 "$tcp set windowlnit 4"
$ns at 0.305 "$tcp set maxcwnd 4"
$ns at 0.368 "$ns detach-agent $n1 $tcp ; $ns detach-agent $n4 $sink"
$ns at 1.5 "finish"
$ns at 0.0 "$ns trace-annotate \"Goback N end\""
$ns at 0.05 "$ns trace-annotate \"FTP starts at 0.01\""
$ns at 0.06 "$ns trace-annotate \"Send 6Packets from SYS1 to SYS4\""
$ns at 0.26 "$ns trace-annotate \"Error Occurs for 4th packet so not sent ack for the Packet\""
$ns at 0.30 "$ns trace-annotate \"Retransmit Packet_4 to 6\""
$ns at 1.0 "$ns trace-annotate \"FTP stops\"" proc finish {} {
global ns nf
$ns flush-trace close $nf
puts "filtering..."
#exec tclsh../bin/namfilter.tclgoback.nam #puts "running nam..."
execnamgoback.nam& exit 0
}

$ns run
```

Sample Input /Output :



Result:

Thus the Go Back N Protocols are Simulated and studied.

Ex :10(b) # Simulation of Selective Repeat Protocol

DATE:

Aim :

To Simulate and to study of Go Back N protocol

Theory:

Selective Repeat ARQ is a specific instance of the Automatic Repeat-reQuest (ARQ) Protocol. It may be used as a protocol for the delivery and acknowledgement of message units, or it may be used as a protocol for the delivery of subdivided message sub-units. When used as the protocol for the delivery of messages, the sending process continues to send a number of frames specified by a window size even after a frame loss. Unlike Go- Back-N ARQ, the receiving process will continue to accept and acknowledge frames sent after an initial error. The receiver process keeps track of the sequence number of the earliest frame it has not received, and sends that number with every ACK it sends. If a frame from the sender does not reach the receiver, the sender continues to send subsequent frames until it has emptied its window. The receiver continues to fill its receiving window with the subsequent frames, replying each time with an ACK containing the sequence number of the earliest missing frame. Once the sender has sent all the frames in its window, it re-sends the frame number given by the ACKs, and then continues where it left off. The size of the sending and receiving windows must be equal, and half the maximum sequence number (assuming that sequence numbers are numbered from 0 to n-1) to avoid miscommunication in all cases of packets being dropped. To understand this, consider the case when all ACKs are destroyed. If the receiving window is larger than half the maximum sequence number, some, possibly even all, of the packages that are resent after timeouts are duplicates that are not recognized as such. The sender moves its window for every packet that is acknowledged.

Advantage over Go Back N:

1. Fewer retransmissions.

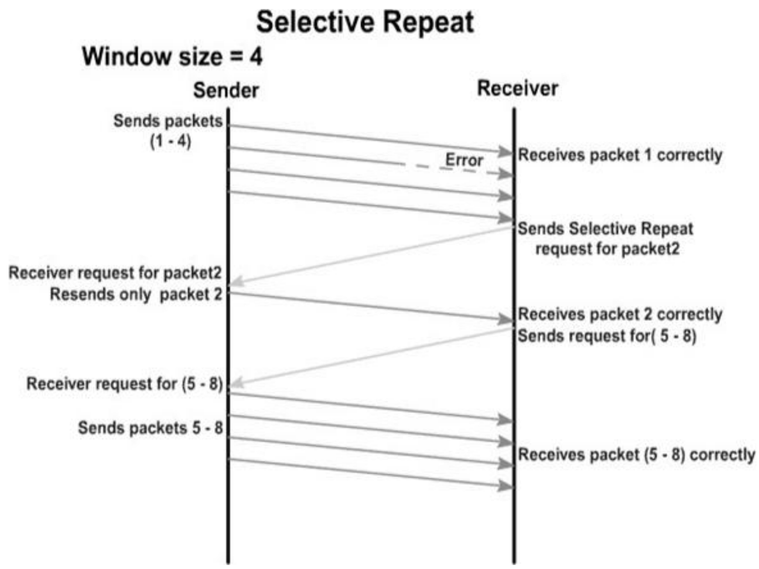
Disadvantages:

1. More complexity at sender and receiver
2. Receiver may receive frames out of sequence

Algorithm : SELECTIVE REPEAT

1. The source node transmits the frames continuously.
2. Each frame in the buffer has a sequence number starting from 1 and increasing up to the window size.
3. The source node has a window i.e. a buffer to store the frames. This buffer size is the number of frames to be transmitted continuously.
4. The receiver has a buffer to store the received frames. The size of the buffer depends upon the window size defined by the protocol designer.
5. The size of the window depends according to the protocol designer.

6. The source node transmits frames continuously till the window size is exhausted. If any of the frames are received with error only those frames are requested for retransmission (with a negative acknowledgement)
7. If all the frames are received without error, a cumulative positive acknowledgement is sent.
8. If there is an error in frame 3, an acknowledgement for the frame 2 is sent and then only Frame 3 is retransmitted. Now the window slides to get the next frames to the window.
9. If acknowledgment is transmitted with error, all the frames of window are retransmitted. Else ordinary window sliding takes place. (* In implementation part, Acknowledgment error is not considered)
10. If all the frames transmitted are errorless the next transmission is carried out for the new window.
11. This concept of repeating the transmission for the error frames only is called Selective Repeat transmission flow control protocol.



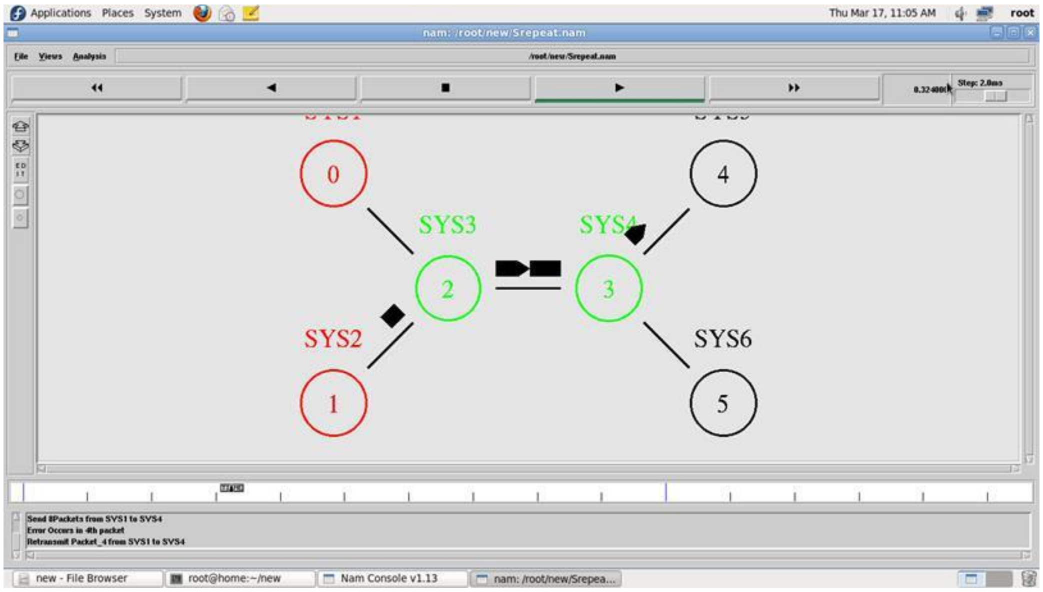
#PROGRAM FOR SELECTIVE REPEAT:

```
#send packets one by one set ns [new Simulator] 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 color "red"
$n1 color "red"
$n2 color "green"
$n3 color "green"
$n4 color "black"
$n5 color "black"
$n0 shape circle ;
$n1 shape circle ;
$n2 shape circle ;
$n3 shape circle ;
$n4 shape circle ;
$n5 shape circle ;
$ns at 0.0 "$n0 label SYS1"
$ns at 0.0 "$n1 label SYS2"
```

```
$ns at 0.0 "$n2 label SYS3"
$ns at 0.0 "$n3 label SYS4"
$ns at 0.0 "$n4 label SYS5"
$ns at 0.0 "$n5 label SYS6" set nf [open Srepeat.nam
w] $ns namtrace-all $nf
set f [open Srepeat.tr w]
$ns trace-all $f
$ns duplex-link $n0 $n2 1Mb 10ms DropTail
$ns duplex-link-op $n0 $n2 orient right-down
$ns queue-limit $n0 $n2 5
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link $n2 $n3 1Mb 10ms DropTail
$ns duplex-link-op $n2 $n3 orient right
$ns duplex-link $n3 $n4 1Mb 10ms DropTail
$ns duplex-link-op $n3 $n4 orient right-up
$ns duplex-link $n3 $n5 1Mb 10ms DropTail
$ns duplex-link-op $n3 $n5 orient right-down Agent/TCP
set_nam_tracevar_true settcp [new Agent/TCP]
$tcp set fid 1
$ns attach-agent $n1 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n4 $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 0.05 "$ftp start"
$ns at 0.06 "$tcp set windowlnit 8"
$ns at 0.06 "$tcp set maxcwnd 8"
$ns at 0.25 "$ns queue-limit $n3 $n4 0"
$ns at 0.26 "$ns queue-limit $n3 $n4 10"
$ns at 0.30 "$tcp set windowlnit 1"
$ns at 0.30 "$tcp set maxcwnd 1"
$ns at 0.30 "$ns queue-limit $n3 $n4 10"
 $ns at 0.47 "$ns detach-agent $n1 $tcp;$ns detach-agent $n4 $sink"
$ns at 1.75 "finish"
$ns at 0.0 "$ns trace-annotate \"Select and repeat\""
$ns at 0.05 "$ns trace-annotate \"FTP starts at 0.01\""
$ns at 0.06 "$ns trace-annotate \"Send 8Packets from SYS1 to SYS4\""
$ns at 0.26 "$ns trace-annotate \"Error Occurs in 4th packet \""
$ns at 0.30 "$ns trace-annotate \"Retransmit Packet_4 from SYS1 to SYS4\""
$ns at 1.5 "$ns trace-annotate \"FTP stops\"" proc finish {} {
global ns nf
$ns flush-trace close $nf
puts "filtering..."
#exec tclsh../bin/namfilter.tclsrepeat.nam #puts "running nam..."
execnamSrepeat.nam& exit 0
}
```

$ns run

Sample Input /Output :



Result :

Thus the Selective Repeat Protocols are Simulated and studied.

EX :11 **Simulation of TCP/IP Data Transfer**

DATE:

Aim :

To transfer data between client and server using TCP/IP.

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 3
5. Create duplex links between the nodes to form a Topology
6. Setup TCP Connection between n(0) and n(3)
7. Apply FTP Traffic over TCP
8. Schedule events and run the program.

Program :

Tcp.tcl:

```
set ns [new Simulator]
setnf [open out.nam w]
$ns namtrace-all $nf
settr [open out.tr w]
$ns trace-all $tr
proc finish {} {
globalnf ns tr
$ns flush-trace close $tr
execnamout.nam& exit 0
}
set n0 [$ns node] set n1 [$ns node] set n2 [$ns node] set n3 [$ns node]
$ns duplex-link $n0 $n1 10Mb 10ms DropTail $ns duplex-link $n1 $n3
10Mb 10ms DropTail
$ns duplex-link $n2 $n1 10Mb 10ms DropTail
$ns duplex-link-op $n0 $n1 orient right-down
```

```
$ns duplex-link-op $n1 $n3 orient right
$ns duplex-link-op $n2 $n1 orient right-up

settcp [new Agent/TCP]
$ns attach-agent $n0 $tcp

set ftp [new Application/FTP]
$ftp attach-agent $tcp

set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink

setudp [new Agent/UDP]
$ns attach-agent $n2 $udp

setcbr [new Application/Traffic/CBR]
$cbr attach-agent $udp

set null [new Agent/Null]
$ns attach-agent $n3 $null

$ns connect $tcp $sink
$ns connect $udp $null

$ns rtmodel-at 1.0 down $n1 $n3
$ns rtmodel-at 2.0 up $n1 $n3

$ns rtproto DV
$ns at 0.0 "$ftp start"
$ns at 0.0 "$cbr start"
```

```
$ns at 5.0 "finish"
$ns run
```

Result :

Thus the data has been transfer between client and server using TCP/IP is executed successfully.

EX:12 (a) **Simulationof Distance Vector Routing Algorithm**

DATE:

Aim:

To simulate and study the Distance Vector routing algorithm using simulation.

Theory:

Distance Vector Routing is one of the routing algorithm in a Wide Area Network for computing shortest path between source and destination. The Router is one main devices used in a wide area network. The main task of the router is Routing. It forms the routing table and delivers the packets depending upon the routes in the table-either directly or via an intermediate devices.

Each router initially has information about its all neighbors. Then this information will be shared among nodes.

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 n number of nodes using for loop
5. Create duplex links between the nodes
6. Setup UDP Connection between n(0) and n(5)
7. Setup another UDP connection between n(1) and n(5)
8. Apply CBR Traffic over both UDP connections
9. Choose distance vector routing protocol to transmit data from sender to receiver.
10. Schedule events and run the program.

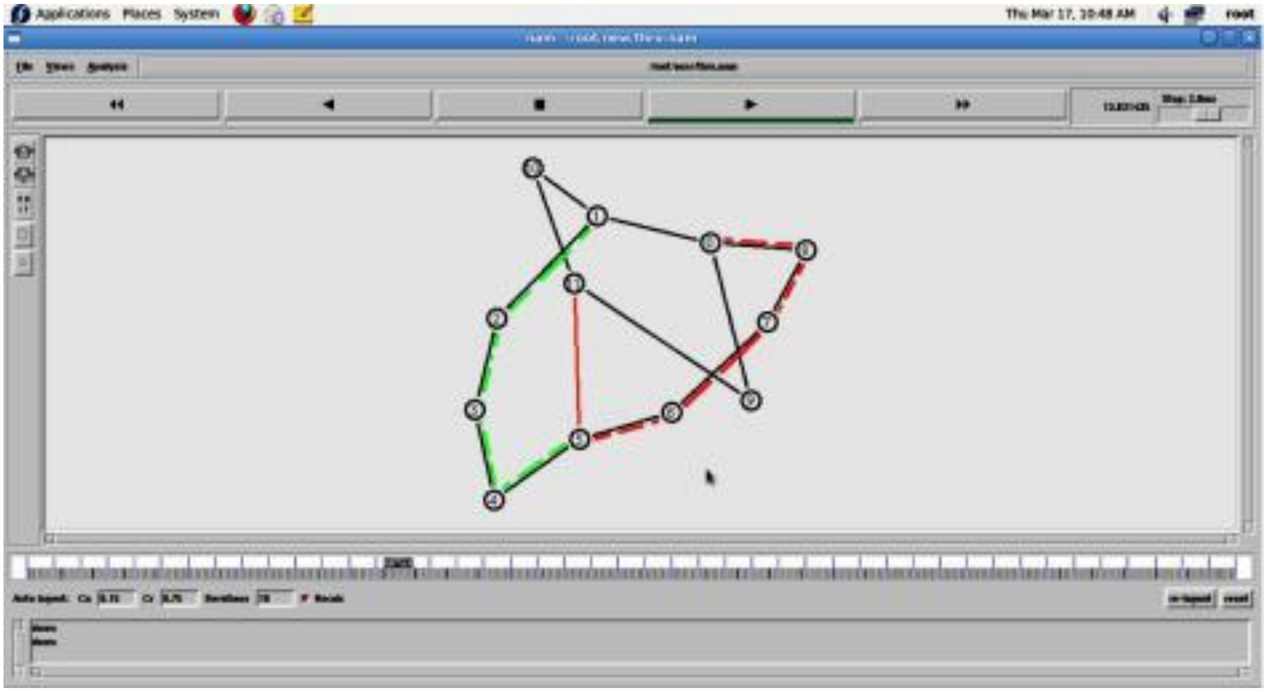
Program: DV.tcl

```
set ns [new Simulator] set nr [open thro.tr w]
$ns trace-all $nr
setnf [open thro.nam w]
$ns namtrace-all $nfproc finish { } { global ns nr nf
$ns flush-trace close $nf close $nr
execnamthro.nam& exit 0
}
for { set i 0 } { $i< 12} { incri 1 } { set n($i) [$ns node]}
for {set i 0} {$i< 8} {incri} {
$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 DV

$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
```

- ns DV.tcl

it will run and open NAM (network Animator)



Click play button to see the animation of Distance –vector routing

Result:

Thus the Distance vector Routing Algorithm was Simulated and studied.

Ex:12(b) Simulation of Link State Routing Algorithm

DATE:

Aim:

To simulate and study the link state routing algorithm using simulation.

Theory:

In link state routing, each router shares its knowledge of its neighborhood with every other router in the internet work. (i) Knowledge about Neighborhood: Instead of sending its entire routing table a router sends info about its neighborhood only. (ii) 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. (iii)Information sharing when there is a change: Each router sends out information about the neighbors when there is change.

Procedure:

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.

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 n number of nodes using for loop
5. Create duplex links between the nodes
6. Setup UDP Connection between n(0) and n(5)
7. Setup another UDP connection between n(1) and n(5)
8. Apply CBR Traffic over both UDP connections
9. Choose Link state routing protocol to transmit data from sender to receiver.
10. Schedule events and run the program.

Program: Ls.tcl

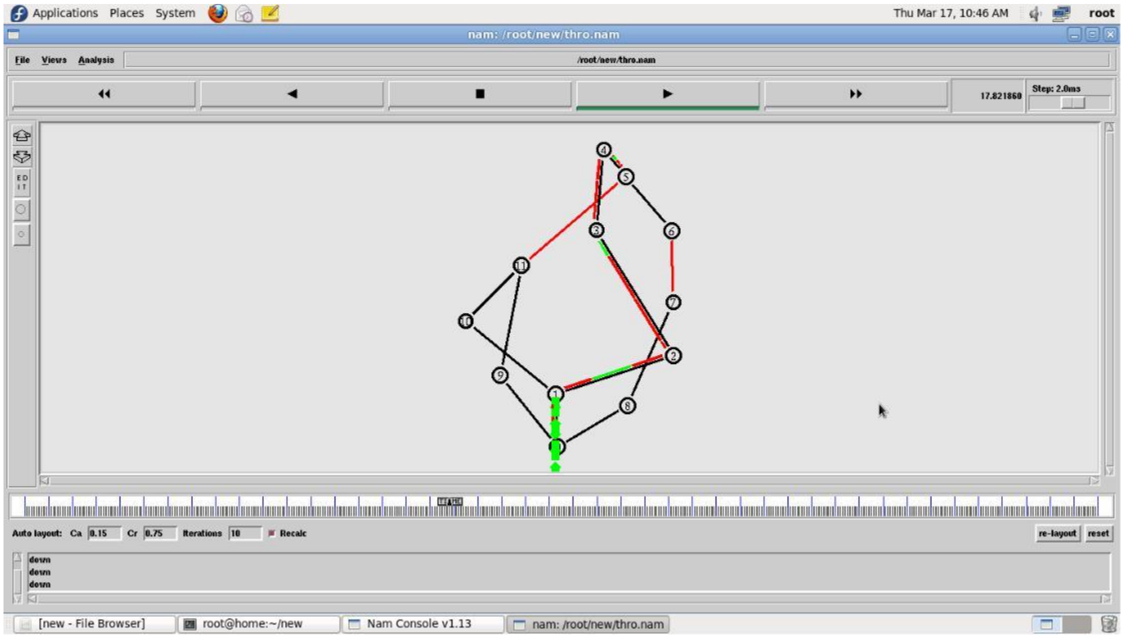
```
set ns [new Simulator] set nr [open thro.tr w]

$ns trace-all $nr
setnf [open thro.nam w]

$ns namtrace-all $nf

proc finish { } { global ns nr nf
$ns flush-trace close $nf close $nr
execnamthro.nam& exit 0
}
for { set i 0 } { $i< 12} { incri 1 } { set n($i) [$ns node]}

for {set i 0} {$i< 8} {incri} {

$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
```

<u>Run the above ls.tcl</u>

it will run and open NAM (network Animator)

Sample Input /Output :



Click play button to see the animation of Distance –vector routing

Result:

Thus the Link State Routing Algorithm was Simulated and studied.

Ex :13 **Simulate a Mobile Adhoc Network**

DATE:

Aim:

To simulate a Mobile Adhoc network (MANET).

Theory :

A mobile ad hoc network or MANET does not depend on a fixed infrastructure for its networking operation. MANET is an autonomous and short-lived association of group of mobile nodes that communicate with each other over wireless links. A node can directly communicate to the nodes that lie within its communication range. If a node wants to communicate with a node that is not directly within its communication range, it uses intermediate nodes as routers.

Algorithm:

1. Create a simulator object
2. Set the values for the parameter
3. Open a nam trace file and define finish procedure then close the trace file, and execute nam on trace file.
4. Create the nodes that forms a network numbered from 0 to 3
5. Schedule events and run the program.

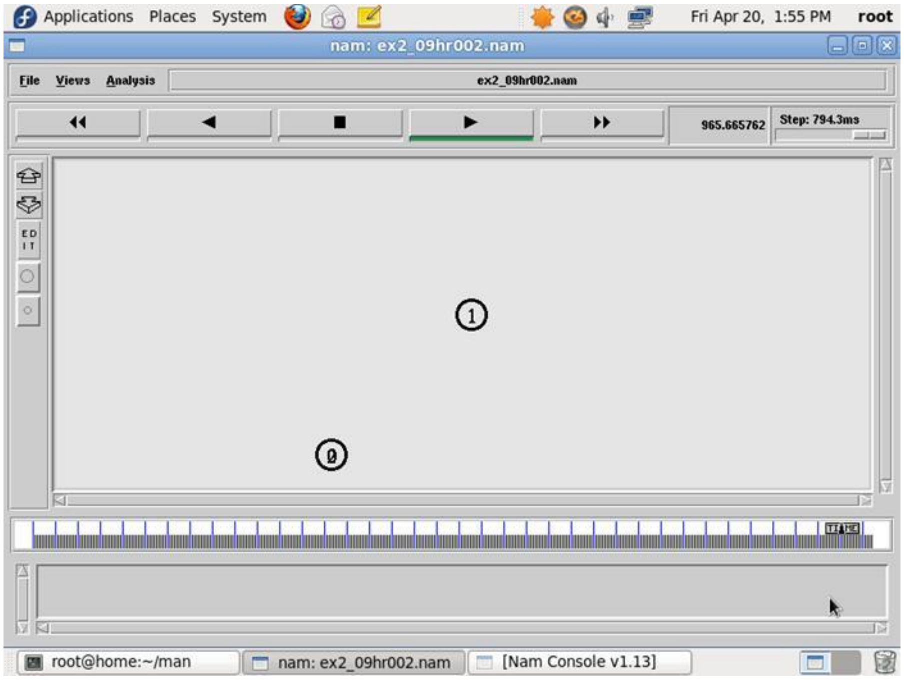
Program :

```
setval(chan) Channel/WirelessChannel set val(prop) Propagation/TwoRayGround set
val(netif) Phy/WirelessPhy
setval(mac) Mac/802_11
setval(ifq)    Queue/DropTail/PriQueue    set    val(ll)    LL
setval(ant)    Antenna/OmniAntenna    set    val(ifqlen)    50
setval(nn) 3
setval(rp) DSDV
set ns [new Simulator]

settf [open output.tr w]
$ns trace-all $tf

set tf1 [open output.nam w]
$ns namtrace-all-wireless $tf1 100 100

settopo [new Topography]
$topoload_flatgrid 100 100 create-god $val(nn)
$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) \
-channelType $val(chan) \

-topoInstance $topo \
-agentTrace ON \
-routerTrace OFF \
-macTrace OFF \
-movementTrace OFF

set node0 [$ns node] set node1 [$ns node] set node2 [$ns node]

$ns initial_node_pos $node0 10
$ns initial_node_pos $node1 10
$ns initial_node_pos $node2 10

$node0 set X_ 25.0
$node0 set Y_ 50.0
$node0 set Z_ 0.0

$node1 set X_ 50.0
$node1 set Y_ 50.0
$node1 set Z_ 0.0

$node2 set X_ 65.0
$node2 set Y_ 50.0
$node2 set Z_ 0.0

set tcp1 [new Agent/TCP]
$ns attach-agent $node0 $tcp1

set ftp [new Application/FTP]
$ftp attach-agent $tcp1

set sink1 [new Agent/TCPSink]
$ns attach-agent $node2 $sink1

$ns connect $tcp1 $sink1

$ns at 10.0 "$node1 setdest 50.0 90.0 0.0"
$ns at 50.0 "$node1 setdest 50.0 10.0 0.0"
```

```
$ns at 0.5 "$ftp start"
$ns at 1000 "$ftp stop"

$ns at 1000 "finish" proc finish {} { global ns tf tf1
$ns flush-trace close $tf
execnamoutput.nam& exit 0
}

$ns run
```

Sample Input / Output :



Result:

Thus the mobile adhoc network is simulated successfully

Ex :14 **Implement TransportControl Protocol in Sensor Network**

DATE:

Aim:

To implement a Transport Control Protocol in sensor network through the simulator.

Theory:

The Transmission Control Protocol (TCP) is one of the main protocols of the Internet protocol suite. It originated in the initial network implementation in which it complemented the Internet Protocol (IP). Therefore, the entire suite is commonly referred to as TCP/IP. TCP provides reliable, ordered, and error-checked delivery of a stream of octets (bytes) between applications running on hosts communicating by an IP network. Major Internet applications such as the World Wide Web, email, remote administration, and file transfer rely on TCP. Applications that do not require reliable data stream service may use the User Datagram Protocol (UDP), which provides a connectionless datagram service that emphasizes reduced latency over reliability.

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 and add Orientation to the nodes for setting a LAN topology
6. Setup TCP Connection between n(0) and n(4)
7. Apply FTP Traffic over TCP
8. Setup UDP Connection between n(1) and n(5)
9. Apply CBR Traffic over UDP.
10. Schedule events and run the program.

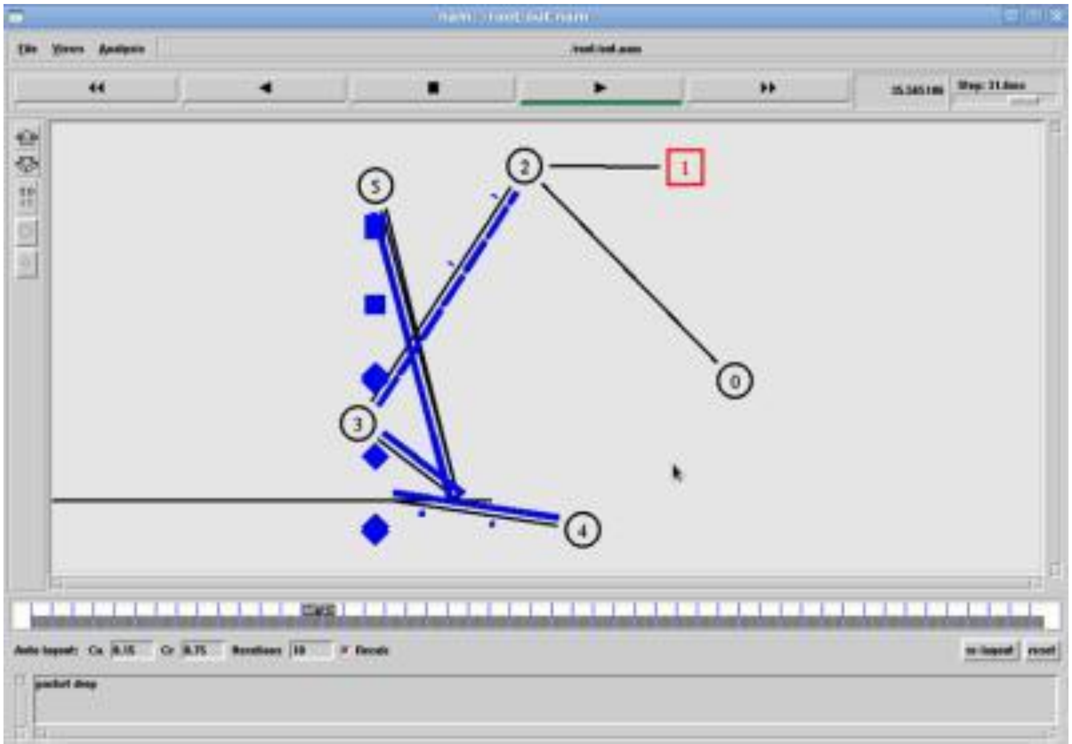
Program:

```
set ns [new Simulator]
#Define different colors for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Red #Open the Trace files
set file1 [open out.tr w]
setwinfile [open WinFile w]
$ns trace-all $file1
```

```
#Open the NAM trace file set file2 [open out.nam w]
$ns namtrace-all $file2 #Define a 'finish' procedure proc finish {} { global ns
file1 file2
$ns flush-trace close $file1 close $file2
execnamout.nam& exit 0
}
#Create six nodes set n0 [$ns node] set n1 [$ns node]

set n2 [$ns node] set n3 [$ns node] set n4 [$ns node] set n5 [$ns node] $n1 color
red
$n1 shape box
#Create links between the nodes
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns simplex-link $n2 $n3 0.3Mb 100ms DropTail
$ns simplex-link $n3 $n2 0.3Mb 100ms DropTail
setlan [$ns newLan "$n3 $n4 $n5" 0.5Mb 40ms LL Queue/DropTail
MAC/Csma/Cd Channel] #Setup a TCP connection
settcp [new Agent/TCP/Newreno]
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink/DelAck]
$ns attach-agent $n4 $sink
$ns connect $tcp $sink
$tcp set fid_ 1
$tcp set window_ 8000
$tcp set packetSize_ 552
#Setup a FTP over TCP connection set ftp [new Application/FTP] $ftp
attach-agent $tcp
$ftp set type_ FTP

$ns at 1.0 "$ftp start"
$ns at 124.0 "$ftp stop"

#next procedure gets two arguments: the name of the # tcp source node, will be called here
"tcp",
# and the name of output file.
procplotWindow {tcpSource file} { global ns
set time 0.1
set now [$ns now]
setcwnd [$tcpSource set cwnd_] set wnd [$tcpSource set window_] puts $file "$now
$cwnd"
$ns at [expr $now+$time] "plotWindow $tcpSource $file" } $ns at 0.1
"plotWindow $tcp $winfile"
$ns at 5 "$ns trace-annotate \"packet drop\"" # PPP $ns at
125.0 "finish"
```

$ns run

Sample Input / Output :



Result :

Thus the Transport control protocol in sensor network is simulated successfully.