

EXPERIMENT-9

Aim: Installation of NS-2/3 Network Simulator: Basics of Network Simulation.

NS-2:

- It stands for Network Simulator-2. It is an event driven simulation tool that is widely used for studying the dynamic nature of the communication networks.
- It can be used for simulating wired as well as wireless network function and protocols like TCP/IP, UDP etc.
- It provides a method for the user to specify network protocols and simulate their corresponding behaviour.

Features of NS-2:

1. It simulates wired and wireless networks.
2. It is primarily Unix-based.
3. It is a discrete simulator.
4. It provides support to a variety of protocols: TCP, FTP, UDP, http and DSR.
5. Uses TCL as its scripting language.

Basic Structure of NS-2:

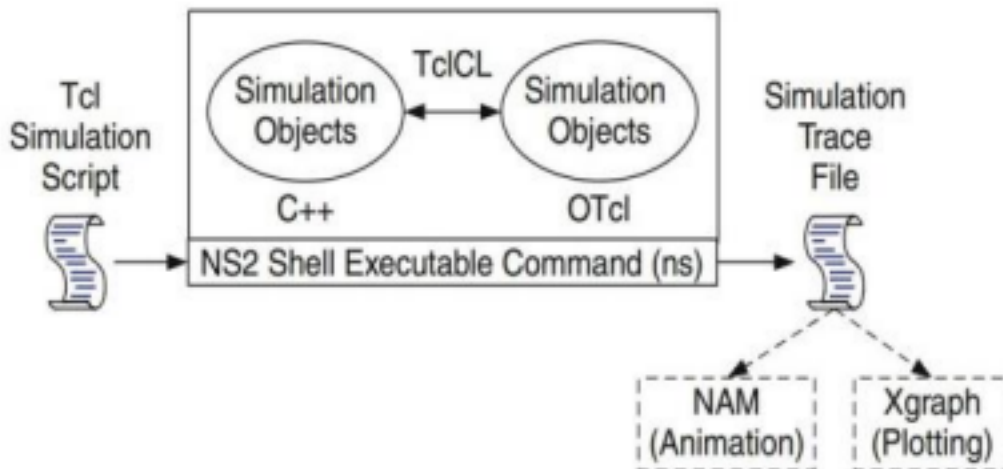
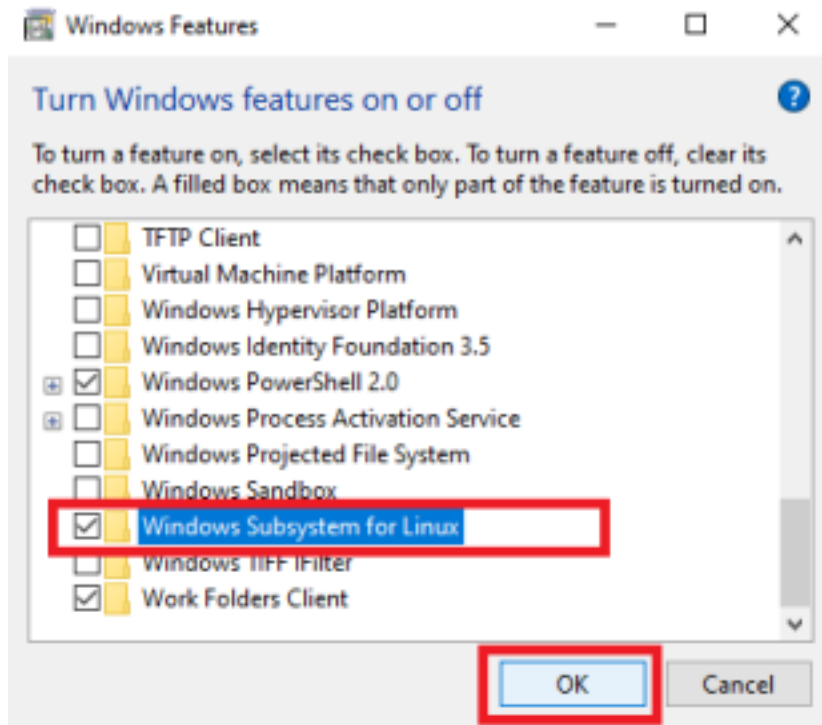


Figure shows the basic architecture of NS2. NS2 provides users with an executable command ns which takes on input argument, the name of a Tcl simulation scripting file. Users are feeding the name of a Tcl simulation script (which sets up a simulation) as an input argument of an NS2 executable command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

Installation of NS-2 on Windows:

Step-1: Open Windows Features and Enable Windows Subsystem for Linux:



Step-2: Open Command

Prompt and write: `bash`

You will get a link from where you can Download Ubuntu app for windows subsystem.

OR

Install Ubuntu from Microsoft Store:

Click on search and type “Microsoft Store” and click on Open.

In the search bar at the top, type “Ubuntu” and click on “Install” button

Step- 3: Install Xming.

Open Google in a browser and type “Xming Download” or go to this URL <https://sourceforge.net/project/s/xming/> and click on the “Download” button.

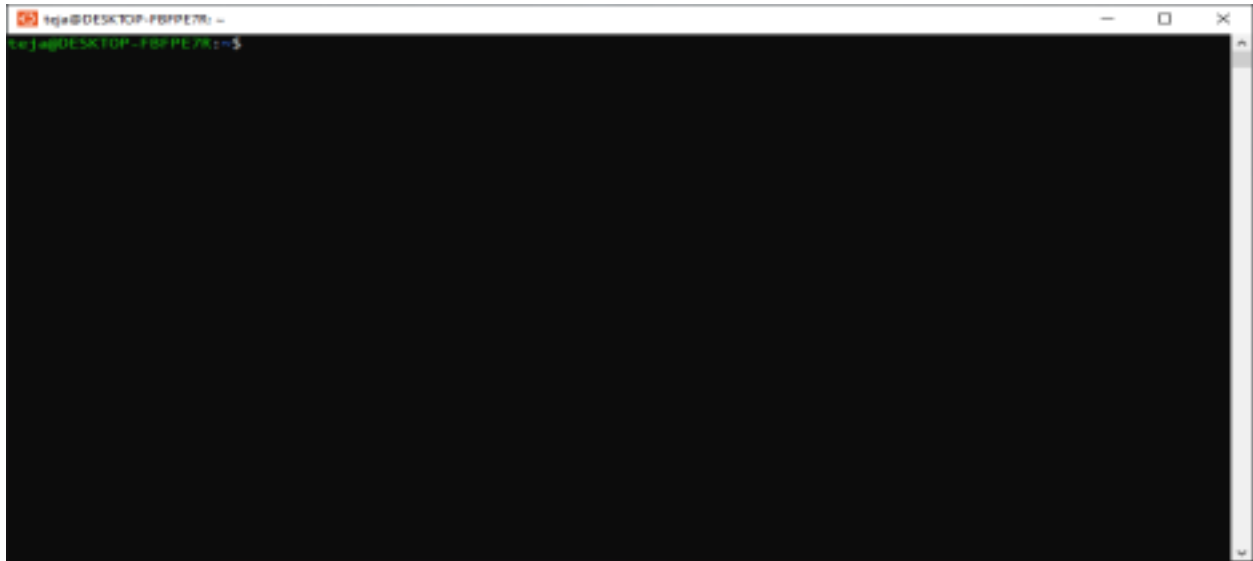
After downloading Xming, install it.

Step- 4: Open search and type “Ubuntu” and click on Open.

As it is the first time you are opening Ubuntu, it will ask for Username and Password.

Give any username and password and hit enter key. Remember this password as it will be asked every time when we do start installation.

Now you should see a prompt as shown below:



Step- 5: Type the following commands one by one and install prerequisite packages:

- sudo apt update
- sudo apt-get install ns2
- sudo apt-get install nam
- sudo apt-get install gedit
- sudo apt install tcl

If your Internet connection is slow, this step will take some time to complete. When asked for username and password provide them as given before.

Step- 6: Changing the location to working directory.

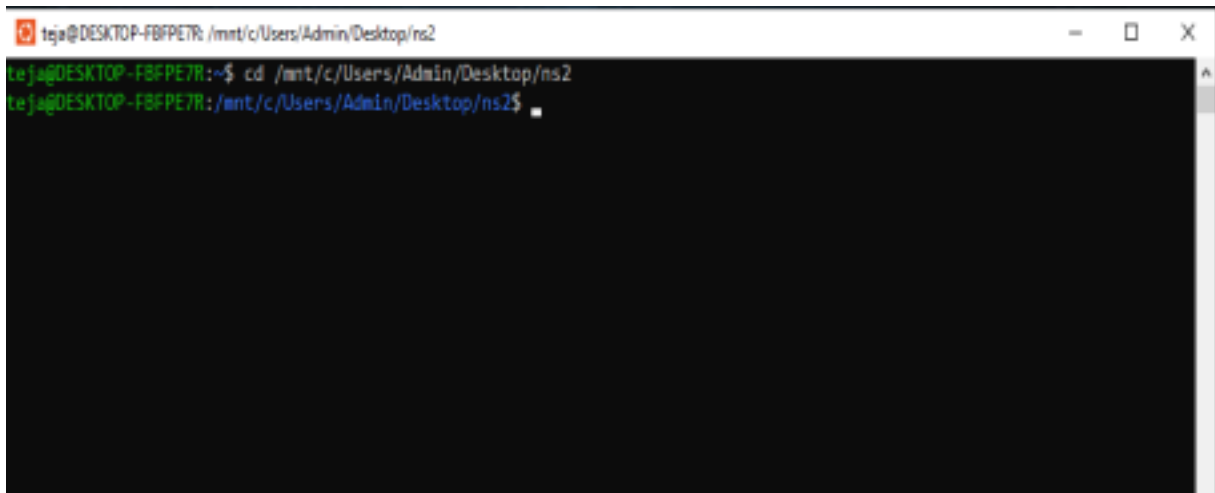
For this example, for working directory name is “ns2” and it is located on my “Desktop”.

The complete path to “ns2” directory on my PC is “C:\Users\Admin\Desktop\ns2”. To

change the location to the above directory, type the following command and hit Enter key. cd

/mnt/c/Users/Admin/Desktop/ns2/

You can see the above command in action in the below figure:

A terminal window titled 'teja@DESKTOP-FBFPETR: /mnt/c/Users/Admin/Desktop/ns2'. The prompt is 'teja@DESKTOP-FBFPETR:~\$'. The first command entered is 'cd /mnt/c/Users/Admin/Desktop/ns2', and the second is 'teja@DESKTOP-FBFPETR:/mnt/c/Users/Admin/Desktop/ns2\$' followed by a cursor.

```
teja@DESKTOP-FBFPETR: /mnt/c/Users/Admin/Desktop/ns2
teja@DESKTOP-FBFPETR:~$ cd /mnt/c/Users/Admin/Desktop/ns2
teja@DESKTOP-FBFPETR:/mnt/c/Users/Admin/Desktop/ns2$
```

Note the path may vary from system to system. For Example, let suppose ns2 programs are going to be saved in the “ns2” folder as shown above.

Step- 7: Opening gedit and typing the program.

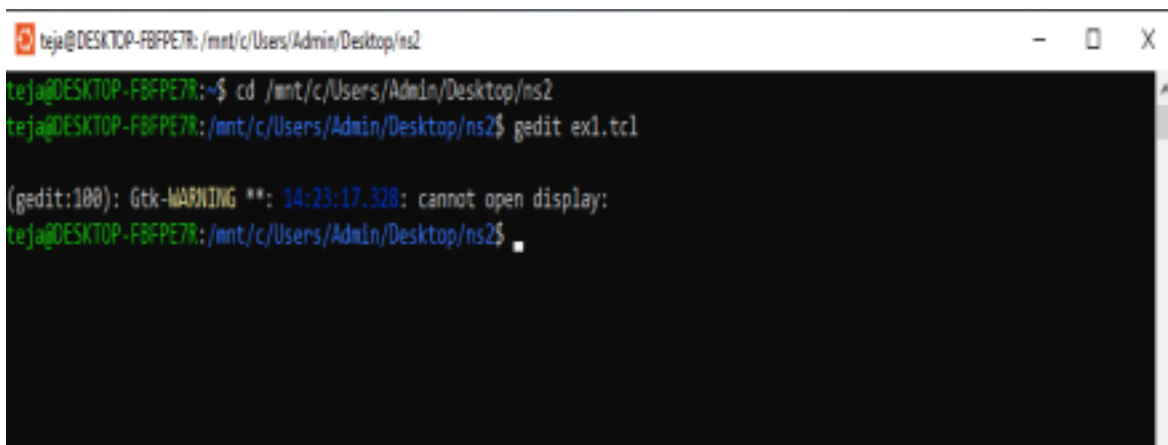
Type the command: ls

You will get the list of files available here. Let's file with name ***exmpl1.tcl*** will display

Then type the following command in the terminal and hit Enter key.

gedit exmpl1.tcl

You might get a warning message as shown in below figure:

A terminal window titled 'teja@DESKTOP-FBFPETR: /mnt/c/Users/Admin/Desktop/ns2'. The prompt is 'teja@DESKTOP-FBFPETR:~\$'. The first command entered is 'cd /mnt/c/Users/Admin/Desktop/ns2', and the second is 'teja@DESKTOP-FBFPETR:/mnt/c/Users/Admin/Desktop/ns2\$ gedit ex1.tcl'. Below the command, a warning message is displayed: '(gedit:100): Gtk-WARNING **: 14:23:17.328: cannot open display:'. The prompt then returns to 'teja@DESKTOP-FBFPETR:/mnt/c/Users/Admin/Desktop/ns2\$' with a cursor.

```
teja@DESKTOP-FBFPETR: /mnt/c/Users/Admin/Desktop/ns2
teja@DESKTOP-FBFPETR:~$ cd /mnt/c/Users/Admin/Desktop/ns2
teja@DESKTOP-FBFPETR:/mnt/c/Users/Admin/Desktop/ns2$ gedit ex1.tcl

(gedit:100): Gtk-WARNING **: 14:23:17.328: cannot open display:
teja@DESKTOP-FBFPETR:/mnt/c/Users/Admin/Desktop/ns2$
```

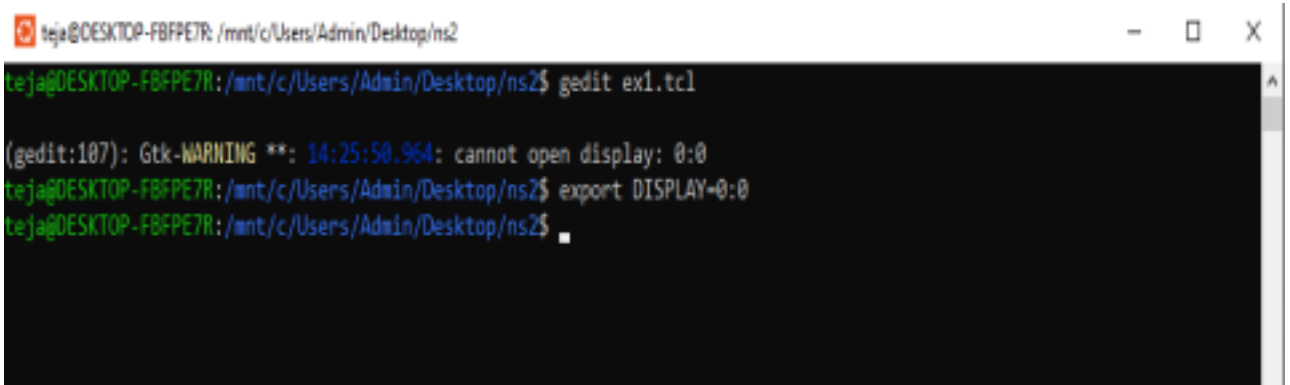
Step- 8: Opening Xming and check the task bar.

For opening the “gedit” application we have to start the “Xming” application.

Open search and type “Xming” and click on “Open”.

The “Xming” server will be started and it will be available in the system tray on the taskbar. Sometimes it might get hidden in the system tray and is not always visible.

Now, in the terminal type the following command as shown in the figure below and hit enter key.



```
teja@DESKTOP-FBFPE7R: /mnt/c/Users/Admin/Desktop/ns2
teja@DESKTOP-FBFPE7R: /mnt/c/Users/Admin/Desktop/ns2$ gedit ex1.tcl
(gedit:107): Gtk-WARNING **: 14:25:50.964: cannot open display: 0:0
teja@DESKTOP-FBFPE7R: /mnt/c/Users/Admin/Desktop/ns2$ export DISPLAY=0:0
teja@DESKTOP-FBFPE7R: /mnt/c/Users/Admin/Desktop/ns2$
```

Now, type the following command in the terminal (black window) and hit Enter key.

`gedit exmp1.tcl`

You should be able to see a blank gedit window as shown in below image.



Type the following sample program and save (CTRL + s) the file. Now close gedit window.

```
#Create global variables
set ns [new Simulator]
```

```
#setting nam trace
set namf [open wired1.nam w]
```

```

$ns namtrace-all $namf

#open the trace file
set tracef [open wired1.tr w]
$ns trace-all $tracef
set proto rlm

#setting the color values
$ns color 1 blue
$ns color 2 yellow
$ns color 3 red

#----- creating client- router- end server
node-----# set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]

#establish the link between the nodes
$ns duplex-link $n0 $n1 2Mb 100ms DropTail
$ns duplex-link $n1 $n2 200Kb 100ms DropTail

#Label the nodes
$ns at 0.0 "$n0 label Client1"
$ns at 0.0 "$n1 label Server"
$ns at 0.0 "$n2 label Client2"

#setting the color for nodes
$n0 color blue
$n1 color red
$n2 add-mark pradeep green square

#Shaping the nodes for differentiation
$n1 shape hexagon
$n2 shape square

#finish procedure
proc finish {} {
    global ns tracefnamf
    $ns flush-trace
    close $tracef
    close $namf
    puts "Opening nam..."
    exec nam wired1.nam &
    exit 0
}
#Calling finish procedure
$ns at 2.0 "finish"
$ns run

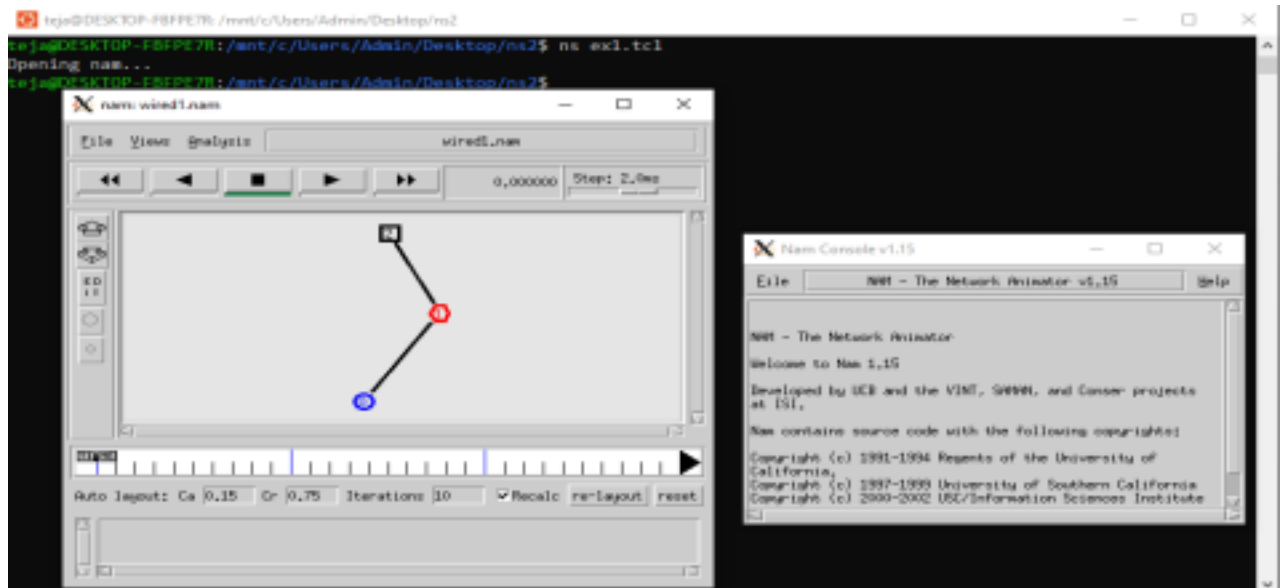
```

Step- 8: Running the program using “ns” command.

Type the following command in the terminal to run the program and see the output.

```
ns exmpl.tcl
```

Now, you should be able to see the network animator (nam) window with the topology as shown in the below figure.



The above output, shows that you have installed ns2 successfully on your system.

EXPERIMENT-11

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

HARDWARE / SOFTWARE REQUIREMENTS:

NS-2

THEORY:

Token bus is a LAN protocol operating in the MAC layer. Token bus is standardized as per IEEE

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

ALGORITHM:

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

PROGRAM:

```
#Create a simulator object
```

```
set ns [new Simulator]
```

```
#Open the nam trace file
```

```
set nf [open out.nam w]
```

```
$ns namtrace-all $nf
```

```
#Define a 'finish' procedure
```

```
proc finish {} {
```

```
global ns nf
```

```
$ns flush-trace
```

```
#Close the trace file
```



```

close $nf
#Executenam on the trace file
exec nam out.nam &
exit 0
}
#Create five nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
#Create Lan between the nodes
set lan0 [$ns newLan "$n0 $n1 $n2 $n3 $n4" 0.5Mb 40ms LL Queue/DropTail MAC/Csma/Cd
Channel]

#Create a TCP agent and attach it to node n0
set tcp0 [new Agent/TCP]
$tcp0 set class_ 1
$ns attach-agent $n1 $tcp0
#Create a TCP Sink agent (a traffic sink) for TCP and attach it to node n3 set
sink0 [new Agent/TCPSink]
$ns attach-agent $n3 $sink0
#Connect the traffic sources with the traffic sink $ns
connect $tcp0 $sink0
# Create a CBR traffic source and attach it to tcp0 set
cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.01
$cbr0 attach-agent $tcp0
#Schedule events for the CBR agents
$ns at 0.5 "$cbr0 start"

```

\$ns at 4.5 "\$cbr0 stop"

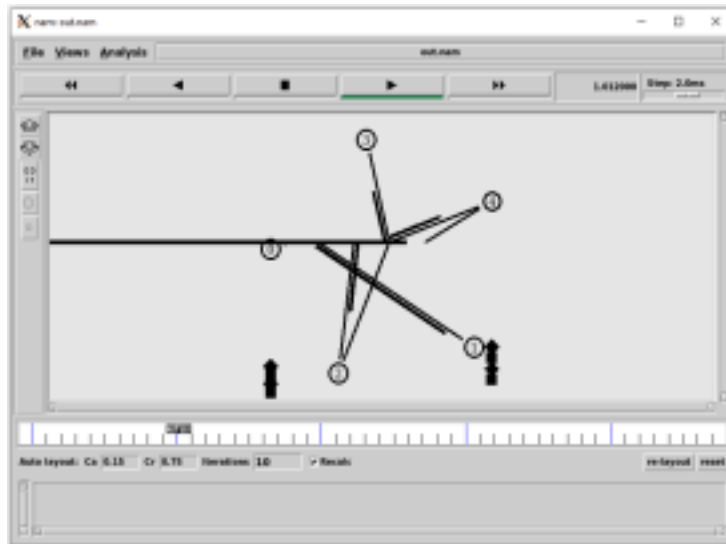
#Call the finish procedure after 5 seconds of simulation time \$ns

at 5.0 "finish"

#Run the simulation

\$ns run

OUTPUT:



EXPERIMENT-12

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

HARDWARE / SOFTWARE REQUIREMENTS:

NS-2

THEORY:

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 each station examines the destination address and is either forwarded (if the recipient is another node) or copied. After copying 4 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.

ALGORITHM:

1. Create a simulator object
2. Define different colors for different data flows
3. Open a nam trace file and define finish procedure then close the trace file, and execute nam on trace file.
4. Create five nodes that forms a network numbered from 0 to 4
5. Create duplex links between the nodes 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.

PROGRAM:

```
#Create a simulator object

set ns [new Simulator]

#Open the nam trace file

set nf [open out.nam w]

$ns namtrace-all $nf

#Define a 'finish' procedure

proc finish {} {
    global ns nf
    $ns flush-trace
    #Close the trace file
    close $nf
    #Executenam on the trace file
    exec nam out.nam &
    exit 0
}

#Create five nodes

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
```

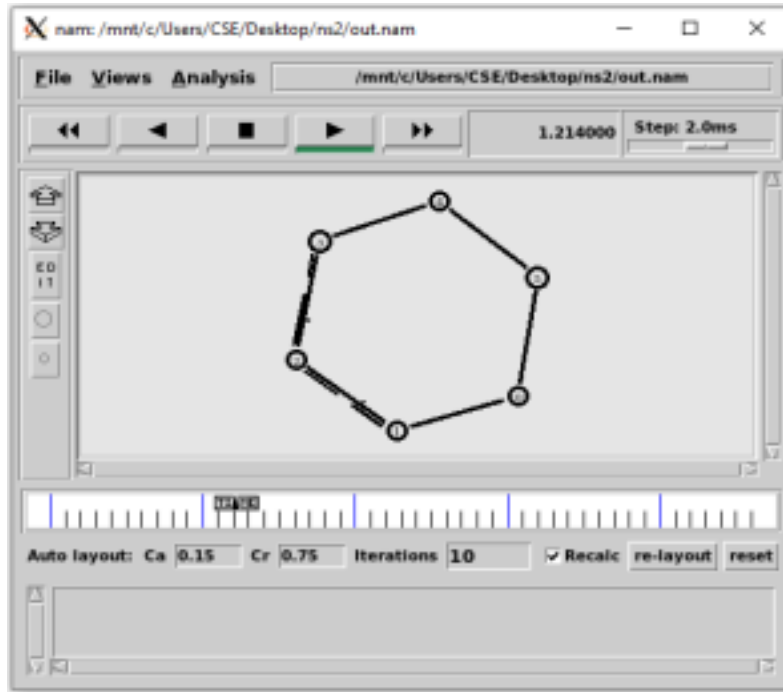
```

set n4 [$ns node]
set n5 [$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

OUTPUT:



EXPERIMENT-13

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

HARDWARE / SOFTWARE REQUIREMENTS:

NS-2

THEORY:

Star networks are one of the most common computer network topologies. In its simplest form, a star network consists of one central switch, hub or computer, which acts as a conduit to transmit messages. This consists of a central node, to which all other nodes are connected; this central node provides a common connection point for all nodes through a hub. In star topology, every node (computer workstation or any other peripheral) is connected to a central node called a hub or switch. The switch is the server and the peripherals are the clients. Thus, the hub and leaf nodes, and the transmission lines between them, form a graph with the topology of a star. If the central node is passive, the originating node must be able to tolerate the reception of an echo of its own

transmission, delayed by the two-way transmission time (i.e. to and from the central node) plus any delay generated in the central node. An active star network has an active central node that usually has the means to prevent echo-related problems.

The star topology reduces the damage caused by line failure by connecting all of the systems to a central node. When applied to a bus-based network, this central hub rebroadcasts all transmissions received from any peripheral node to all peripheral nodes on the network, sometimes including the originating node. All peripheral nodes may thus communicate with all others by transmitting to, and receiving from, the central node only. The failure of a transmission line linking any peripheral node to the central node will result in the isolation of that peripheral node from all others, but the rest of the systems will be unaffected.

ALGORITHM:

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

PROGRAM:

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

#Open the nam trace file
set nf [open out.nam w]

$ns namtrace-all $nf

#Define a 'finish' procedure
proc finish {} {
    global ns nf
    $ns flush-trace
    #Close the trace file
    close $nf
    #Executenam on the trace file
    exec nam out.nam &
    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]

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

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

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

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

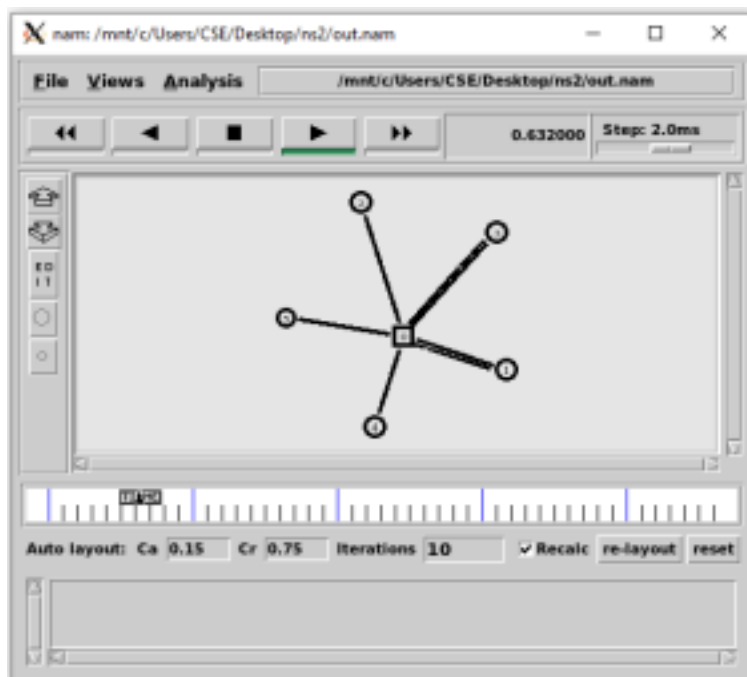
#Connect the traffic sources with the traffic sink $ns
connect $tcp0 $sink0

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

```

```
$cbr0 attach-agent $tcp0
#Schedule events for the CBR agents
$ns at 0.5 "$cbr0 start"
$ns at 4.5 "$cbr0 stop"
#Call the finish procedure after 5 seconds of simulation time $ns
at 5.0 "finish"
#Run the simulation
$ns run
```

OUTPUT:



19115081

EXPERIMENT-14

Aim: Implementation of various MAC protocols.

Algorithm:

- 1) Create a Simulator object.
- 2) Define different colours for different data flows.
- 3) Open a nam trace file and define procedure, after which close the trace file and execute nam on trace file.
- 4) Create six nodes and form a network, 0,1,2,3,4,5.
- 5) Create duplex links between the nodes and add orientation to the nodes for topology.
- 6) Setup TCP connection between n(0) and n(4).
- 7) Apply FTP traffic over TCP.
- 8) Setup UDP Connections between n(1) and n(5).
- 9) Apply CBR traffic over UDP.
- 10) Apply CSMA/CA and CSMA/CD mechanisms and study their performance
- 11) Schedule the events and run the program.

Program Code:

```
#csma.tcl
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]
set winfile [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
    exec nam out.nam &
    exit 0
}
#create six nodes
```

```
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
```

37
19115081

```
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
$ns1 color Red
$ns1 shape box
#create link between 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 $n2 0.3Mb 100ms DropTail
set lan [$ns newLan "$n3 $n4 $n5" 0.5Mb 40ms LL Queue/Droptail MAC/Csma/Ca Channel]
#setup a TCP connection
set tcp [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 FTP over TCP connection
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP
#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 connexion
```

```

set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packet_size_ 1000
$cbr set rate_ 0.01mb
$cbr set random_ false
$ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 123.0 "$ftp stop"
$ns at 124.5 "$cbr stop"
#next procedure gets two arguments: the name of tcp source node will be called here tcp
#and the name of output file

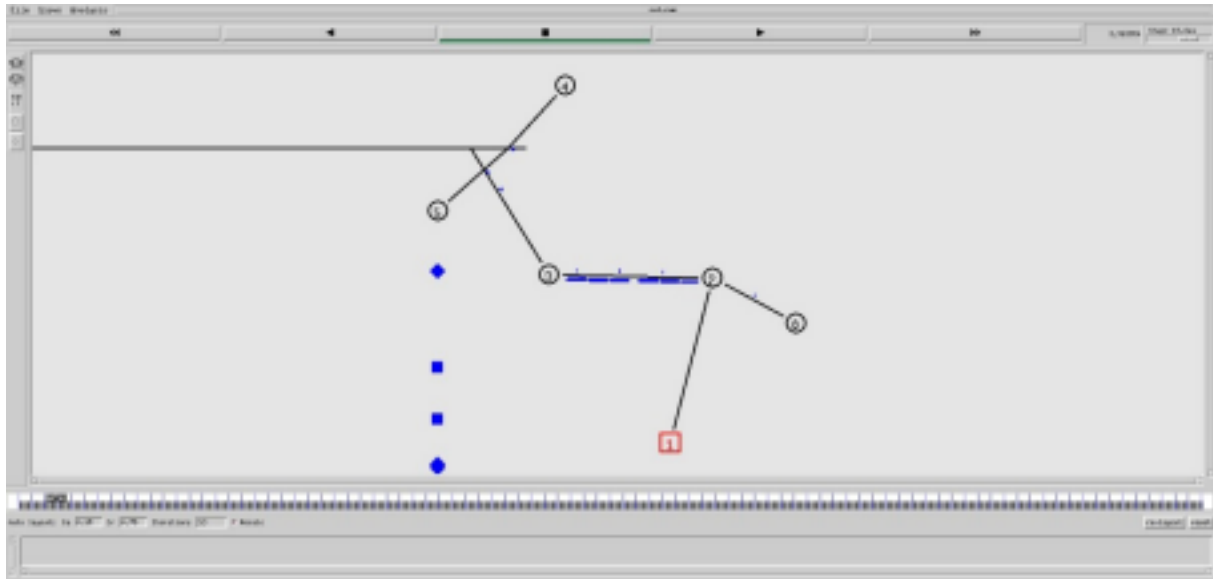
```

```

proc plotWindow {tcpSource file} {
    global ns
    set time 0.1
    set now [$ns now]
    set cwnd [$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

```

Output:



39

EXPERIMENT-15

Aim: To Simulate and to study stop and Wait protocol

HARDWARE / SOFTWARE REQUIREMENTS:

NS-2

THEORY:

Stop and Wait is a reliable transmission flow control protocol. This protocol works only in Connection Oriented (Point to Point) Transmission. The Source node has window size of ONE. After transmission of a frame the transmitting (Source) node waits for an Acknowledgement from the destination node. If the transmitted frame reaches the destination without error, the destination transmits a positive acknowledgement. If the transmitted frame reaches the Destination with error, the receiver destination does not transmit an acknowledgement. If the transmitter receives a positive acknowledgement it transmits the next frame if any. Else if its acknowledgement receive timer expires, it retransmits the same frame.

1. Start with the window size of 1 from the transmitting (Source) node
2. After transmission of a frame the transmitting (Source) node waits for a reply

(Acknowledgement) from the receiving (Destination) node.

3. If the transmitted frame reaches the receiver (Destination) without error, the receiver (Destination) transmits a Positive Acknowledgement.

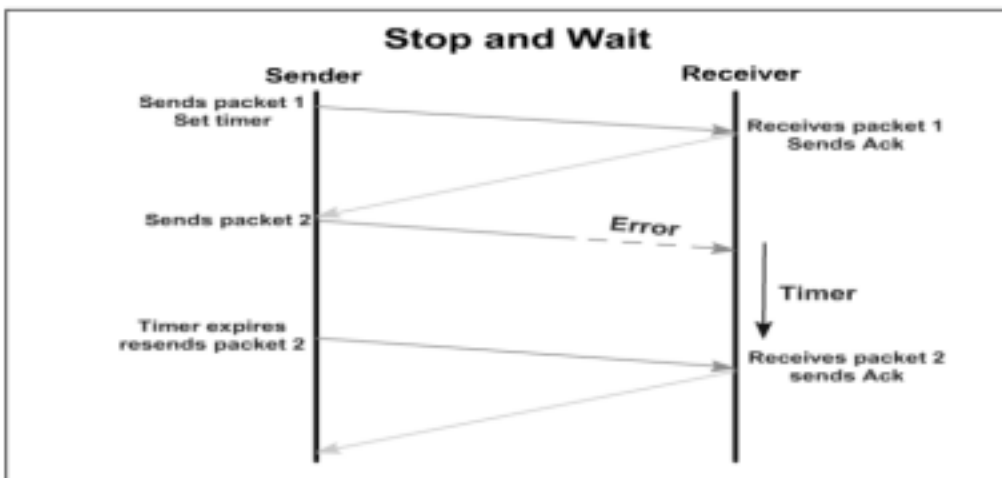
4. If the transmitted frame reaches the receiver (Destination) with error, the receiver (Destination) do not transmit acknowledgement.

5. If the transmitter receives a positive acknowledgement it transmits the next frame if any. Else if the transmission timer expires, it retransmits the same frame again.

6. If the transmitted acknowledgment reaches the Transmitter (Destination) without error, the Transmitter (Destination) transmits the next frame if any.

7. If the transmitted frame reaches the Transmitter (Destination) with error, the Transmitter (Destination) transmits the same frame.

8. This concept of the Transmitting (Source) node waiting after transmission for a reply from the receiver is known as STOP and WAIT.



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 two nodes that forms a network numbered 0 and 1
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:

stop and wait protocol in normal situation

features : labeling, annotation, nam-graph, and window size monitoring

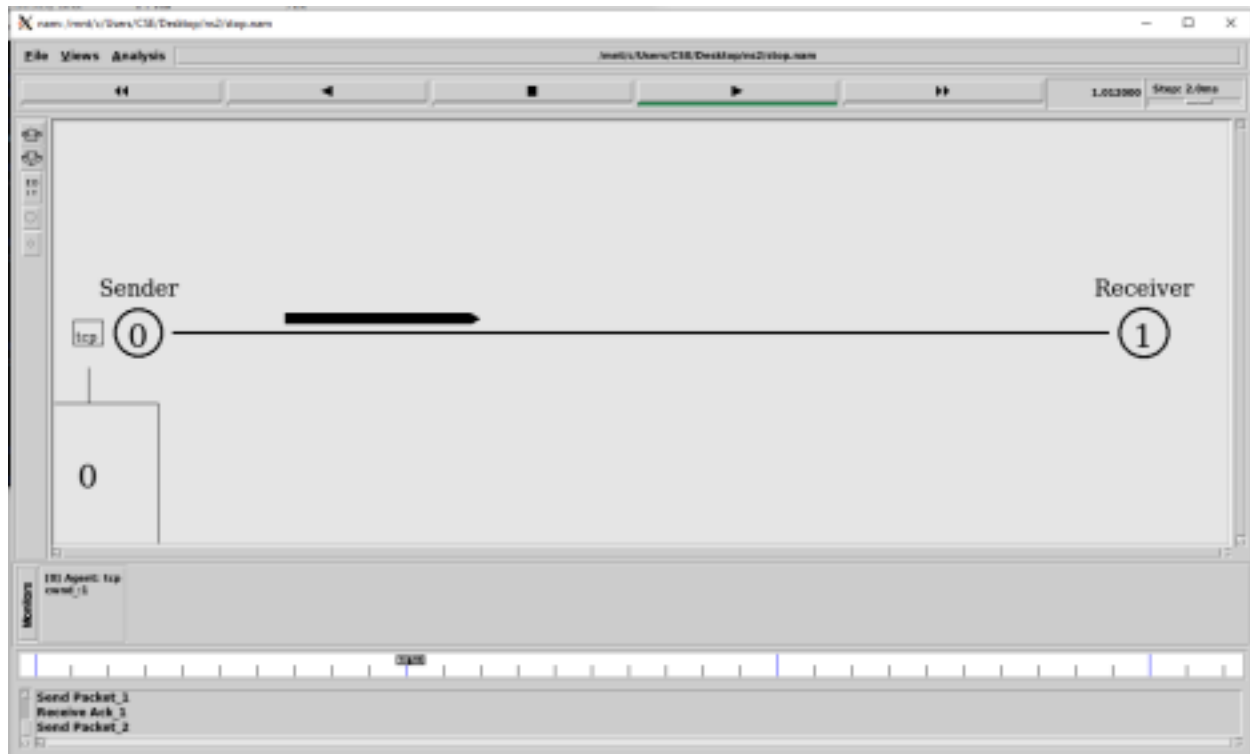
```
set ns [new Simulator]
set n0 [$ns node]
set n1 [$ns node]
$ns at 0.0 "$n0 label Sender"
$ns at 0.0 "$n1 label Receiver"
set nf [open stop.nam w]
$ns namtrace-all $nf
set f [open stop.tr w]
$ns trace-all $f
$ns duplex-link $n0 $n1 0.2Mb 200ms DropTail
$ns duplex-link-op $n0 $n1 orient right
$ns queue-limit $n0 $n1 10
Agent/TCP set nam_tracevar_ true
set tcp [new Agent/TCP]
$tcp set window_ 1
$tcp set maxcwnd_ 1
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n1 $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns add-agent-trace $tcp tcp
$ns monitor-agent-trace $tcp
$tcp tracevar cwnd_
$ns at 0.1 "$ftp start"
$ns at 3.0 "$ns detach-agent $n0 $tcp ; $ns detach-agent $n1
$sink" $ns at 3.5 "finish"
```

```

$ns at 0.0 "$ns trace-annotate \"Stop and Wait with normal
operation\"" $ns at 0.05 "$ns trace-annotate \"FTP starts at 0.1\""
$ns at 0.11 "$ns trace-annotate \"Send Packet_0\""
$ns at 0.35 "$ns trace-annotate \"Receive Ack_0\""
$ns at 0.56 "$ns trace-annotate \"Send Packet_1\""
$ns at 0.79 "$ns trace-annotate \"Receive Ack_1\""
$ns at 0.99 "$ns trace-annotate \"Send Packet_2\""
$ns at 1.23 "$ns trace-annotate \"Receive Ack_2\""
$ns at 1.43 "$ns trace-annotate \"Send Packet_3\""
$ns at 1.67 "$ns trace-annotate \"Receive Ack_3\""
$ns at 1.88 "$ns trace-annotate \"Send Packet_4\""
$ns at 2.11 "$ns trace-annotate \"Receive Ack_4\""
$ns at 2.32 "$ns trace-annotate \"Send Packet_5\""
$ns at 2.55 "$ns trace-annotate \"Receive Ack_5\""
$ns at 2.75 "$ns trace-annotate \"Send Packet_6\""
$ns at 2.99 "$ns trace-annotate \"Receive Ack_6\""
$ns at 3.1 "$ns trace-annotate \"FTP stops\""
proc finish {} {
global ns nf
$ns flush-trace
close $nf
puts "running nam..."
exec nam stop.nam &
exit 0
}
$ns run

```

OUTPUT:



EXPERIMENT-16

Aim: To Simulate and to study Sliding Window protocol

HARDWARE / SOFTWARE REQUIREMENTS:

NS-2

THEORY:

A sliding window protocol is a feature of packet-based data transmission protocols. Sliding window protocols are used where reliable in-order delivery of packets is required, such as in the Data Link Layer (OSI model) as well as in the Transmission Control Protocol (TCP).

Conceptually, each portion of the transmission (packets in most data link layers, but bytes in TCP) is assigned a unique consecutive sequence number, and the receiver uses the numbers to place received packets in the correct order, discarding duplicate packets and identifying missing ones. The problem with this is that there is no limit on the size of the sequence number that can be required.

By placing limits on the number of packets that can be transmitted or received at any given time, a sliding window protocol allows an unlimited number of packets to be communicated using fixed-size sequence numbers. The term "window" on the transmitter side represents the logical boundary of the total number of packets yet to be acknowledged by the receiver. The receiver informs the transmitter in each

acknowledgment packet the current maximum receiver buffer size (window boundary). The TCP header uses a 16 bit field to report the receive window size to the sender. Therefore, the largest window that can be used is $2^{16} = 64$ kilobytes. In slow-start mode, the transmitter starts with low packet count and increases the number of packets in each transmission after receiving acknowledgment packets from receiver. For every ack packet received, the window slides by one packet (logically) to transmit one new packet. When the window threshold is reached, the transmitter sends one packet for one ack packet received. If the window limit is 10 packets then in slow start mode the transmitter may start transmitting one packet followed by two packets (before transmitting two packets, one packet ack has to be received), followed by three packets and so on until 10 packets. But after reaching 10 packets, further transmissions are restricted to one packet transmitted for one ack packet received. In a simulation this appears as if the window is moving by one packet distance for every ack packet received. On the receiver side also the window moves one packet for every packet received.

The sliding window method ensures that traffic congestion on the network is avoided. The application layer will still be offering data for transmission to TCP without worrying about the network traffic congestion issues as the TCP on sender and receiver side implement sliding windows of packet buffer. The window size may vary dynamically depending on network traffic.

For the highest possible throughput, it is important that the transmitter is not forced to stop sending by the sliding window protocol earlier than one round-trip delay time (RTT). The limit on the amount of data that it can send before stopping to wait for an acknowledgment should be larger than the bandwidth-delay product of the communications link. If it is not, the protocol will limit the effective bandwidth of the link.

PROGRAM:

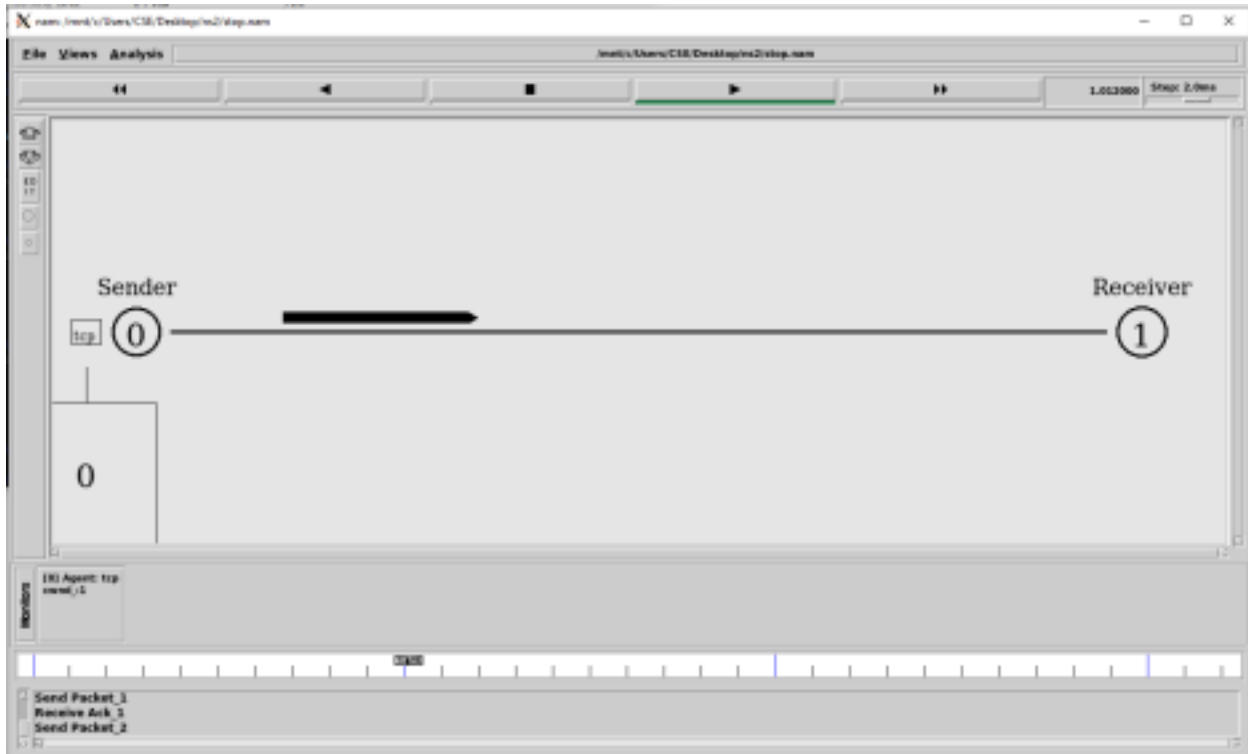
```
# sliding window mechanism with some features
# such as labeling, annotation, nam-graph, and window size
monitoring set ns [new Simulator]
set n0 [$ns node]
set n1 [$ns node]
$ns at 0.0 "$n0 label Sender"
$ns at 0.0 "$n1 label Receiver"
set nf [open sliding.nam w]
$ns namtrace-all $nf
set f [open sliding.tr w]
$ns trace-all $f
$ns duplex-link $n0 $n1 0.2Mb 200ms DropTail
$ns duplex-link-op $n0 $n1 orient right
$ns queue-limit $n0 $n1 10
Agent/TCP set nam_tracevar_ true
set tcp [new Agent/TCP]
$tcp set windowInit_ 4
$tcp set maxcwnd_ 4
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n1 $sink
$ns connect $tcp $sink
```

```

set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns add-agent-trace $tcp tcp
$ns monitor-agent-trace $tcp
$tcp tracevar cwnd_
$ns at 0.1 "$ftp start"
$ns at 3.0 "$ns detach-agent $n0 $tcp ; $ns detach-agent $n1 $sink"
$ns at 3.5 "finish"
$ns at 0.0 "$ns trace-annotate \"Sliding Window with window size 4 (normal
operation)\"" $ns at 0.05 "$ns trace-annotate \"FTP starts at 0.1\""
$ns at 0.11 "$ns trace-annotate \"Send Packet_0,1,2,3\""
$ns at 0.34 "$ns trace-annotate \"Receive Ack_0,1,2,3\""
$ns at 0.56 "$ns trace-annotate \"Send Packet_4,5,6,7\""
$ns at 0.79 "$ns trace-annotate \"Receive Ack_4,5,6,7\""
$ns at 0.99 "$ns trace-annotate \"Send Packet_8,9,10,11\""
$ns at 1.23 "$ns trace-annotate \"Receive Ack_8,9,10,11\""
$ns at 1.43 "$ns trace-annotate \"Send Packet_12,13,14,15\""
$ns at 1.67 "$ns trace-annotate \"Receive Ack_12,13,14,15\""
$ns at 1.88 "$ns trace-annotate \"Send Packet_16,17,18,19\""
$ns at 2.11 "$ns trace-annotate \"Receive Ack_16,17,18,19\""
$ns at 2.32 "$ns trace-annotate \"Send Packet_20,21,22,23\""
$ns at 2.56 "$ns trace-annotate \"Receive Ack_24,25,26,27\""
$ns at 2.76 "$ns trace-annotate \"Send Packet_28,29,30,31\""
$ns at 3.00 "$ns trace-annotate \"Receive Ack_28\""
$ns at 3.1 "$ns trace-annotate \"FTP stops\""
proc finish {} {
global ns
$ns flush-trace
# close $nf
puts "running nam..."
exec nam sliding.nam &
exit 0
}
$ns run

```

OUTPUT:



19115081

EXPERIMENT-17

Aim: Performance Evaluation of routing Protocol.

Routing Protocol:

This protocol is a set of rules used by the routers to communicate between the source and destination. Each protocol has its own algorithm to choose path.

There are two kinds of routing:

- 1) *Unicast Routing*: It is a type of information transfer and used when there is a participation of single sender and a single recipient.
- 2) *Multicast Routing*: It used in case of multiple senders and recipients.

Performance Analysis of routing Protocol:

The analysis of routing Protocol can be assessed by factors such as:

- 1) *Packet delivery Ratio*: It is measured as the ratio of number of packets delivered in

total to the number of packets sent from the source node to destination node. 2) *Energy Consumptions*: Energy consumption is measured as the total energy used to perform the complete process.

3) *Throughput*: The number of packets transferred per unit time.

4) *Average Delay*: It is the delay per packet transfer from source node to the destination node.

All of the above factors and some others affect the performance of routing protocol and can be used to assess its performance.

Program Code:

The following program shows how an XGraph can be used to plot the bandwidth of two nodes connected through the duplex wired link (An XGraph program draws a graph on an x display given data read from either data file or standard input):

```
#Create a simulator object
set ns [new Simulator]
#Open the output trace file
set f0 [open out0.tr w]
#Create 2 nodes
set n0 [$ns node]
set n1 [$ns node]
#Connect the nodes using duplex link
$ns duplex-link $n0 $n1 1Mb 100ms DropTail
#Define a 'finish' procedure
proc finish {} {
    global f0
    #Close the output files
    close $f0
    #Call xgraph to display the results
```

42

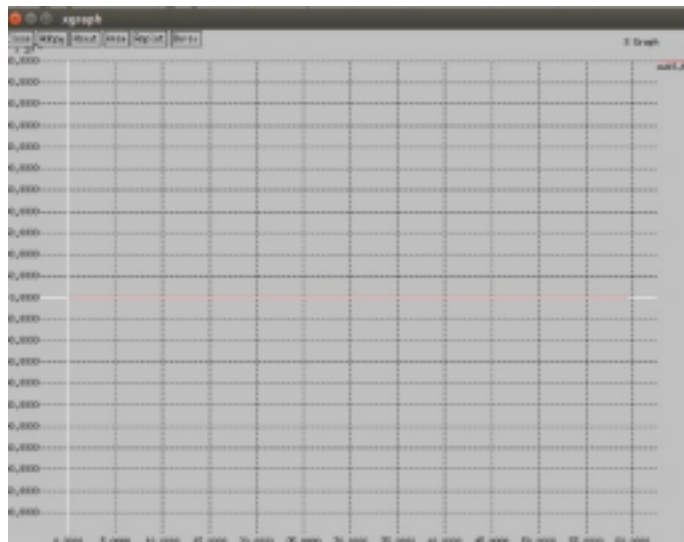
```
exec xgraph out0.tr -geometry 800x400 &
exit 0
}
#Define a procedure which periodically records the bandwidth received by the
global sink0 f0
proc record {} {
```

```

#Get an instance of the simulator
set ns [Simulator instance]
#Set the time after which the procedure should be called again set time 0.5
#How many bytes have been received by the traffic sinks?
set bw0 [$sink0 set bytes_]
#Get the current time
set now [$ns now]
#Calculate the bandwidth (in MBit/s) and write it to the files
puts $f0 "$now [expr $bw0/$time*8/1000000]"
#Reset the bytes_ values on the traffic sinks
$sink0 set bytes_ 0
#Re-schedule the procedure
$ns at [expr $now+$time] "record"
}
#Create three traffic sinks and attach them to the node n4
set sink0 [new Agent/LossMonitor]
#Start logging the received bandwidth
$ns at 0.0 "record"
$ns at 60.0 "finish"
#Run the simulation
$ns run

```

Output:



EXPERIMENT-18

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

HARDWAREE / SOFTWARE REQUIREMENTS:

NS-2

THEORY:

Distance Vector Routing is one of the routing algorithms in a Wide Area Network for computing shortest path between source and destination. The Router is one main device 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 device. 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:

```
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 { } {
```

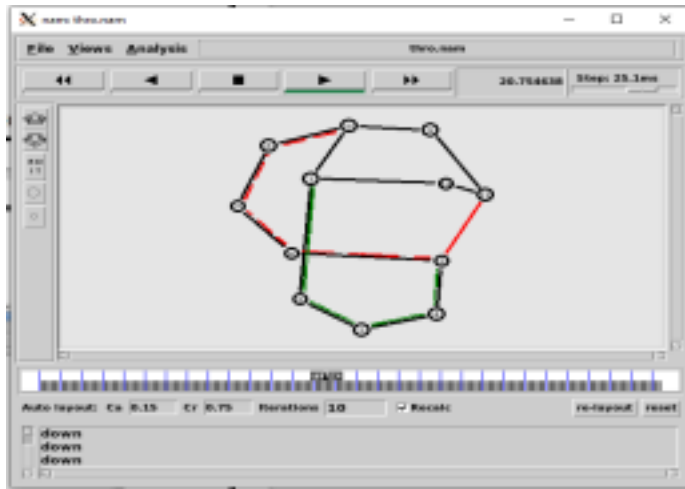
```

global ns nr nf
$ns flush-trace
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 null1 [new Agent/Null]
$ns attach-agent $n(5) $null1
$ns connect $udp1 $null1
$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

OUTPUT:



EXPERIMENT-19

Aim: To simulate and study the link state routing algorithm using simulation

HARDWARE / SOFTWARE REQUIREMENTS:

NS-2

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 by identifying its roots. The root router's tree is 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:

```
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 flush-trace
    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:



EXPERIMENT- 20

Aim: To create scenario and study the performance of CSMA / CD protocol through simulation.

HARDWARE / SOFTWARE REQUIREMENTS:

NS-2

THEORY:

Ethernet is a LAN (Local area Network) protocol operating at the MAC (Medium Access Control) layer. Ethernet has been standardized as per IEEE 802.3. The underlying protocol in Ethernet is known as the CSMA / CD – Carrier Sense Multiple Access / Collision Detection. The working of the Ethernet protocol is as explained below, A node which has data to transmit senses the channel. If the channel is idle then, the data is transmitted. If the channel is busy then, the station defers transmission until the channel is sensed to be idle and then immediately transmitted. If more than one node starts data transmission at the same time, the data collides. This collision is heard by the transmitting nodes which enter into contention phase. The contending nodes resolve contention using an algorithm called Truncated binary exponential back off.

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. Apply CSMA/CA and CSMA/CD mechanisms and study their performance
11. 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]
set winfile [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 $file2
    exec nam out.nam &
    exit 0
}
#Create six nodes
set n0 [$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 $n2 0.3Mb 100ms DropTail
set lan [$ns newLan "$n3 $n4 $n5" 0.5Mb 40ms LL Queue/DropTail MAC/Csma/Channel]
Setup a TCP connection
set tcp [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
#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
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packet_size_ 1000
$cbr set rate_ 0.01mb
$cbr set random_ false
$ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 124.0 "$ftp stop"
$ns at 124.5 "$cbr stop"
# next procedure gets two arguments: the name of the
# tcp source node, will be called here "tcp",
# and the name of output file.
proc plotWindow {tcpSource file} {
global ns
set time 0.1
set now [$ns now]
set cwnd [$tcpSource set cwnd_]
set wnd [$tcpSource set window_]
puts $file "$now $cwnd"

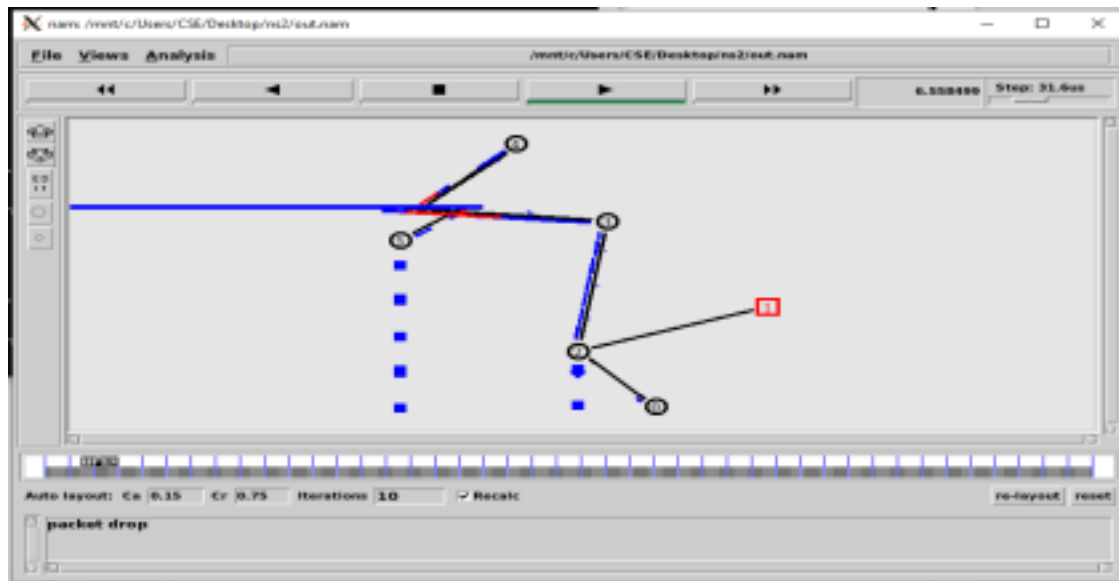
```

```

$ns at [expr $now+$stime] "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

```

OUTPUT:



CSMA/CD

```

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]
set winfile [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
}

```

```

exec nam out.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 $n2 0.3Mb 100ms DropTail
set lan [$ns newLan "$n3 $n4 $n5" 0.5Mb 40ms LL Queue/DropTail MAC/Csma/Cd
Channel] Setup a TCP connection
set tcp [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
#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
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packet_size_ 1000
$cbr set rate_ 0.01mb

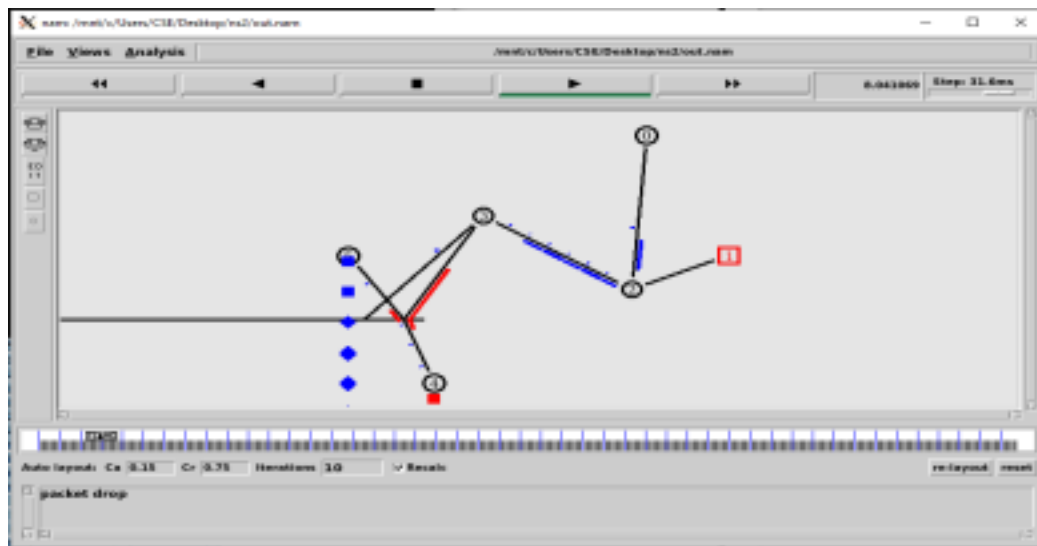
```



```

$scr set random_ false
$ns at 0.1 "$scr start"
$ns at 1.0 "$ftp start"
$ns at 124.0 "$ftp stop"
$ns at 124.5 "$scr stop"
# next procedure gets two arguments: the name of the
# tcp source node, will be called here "tcp",
# and the name of output file.
proc plotWindow {tcpSource file} {
global ns
set time 0.1
set now [$ns now]
set cwnd [$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

```



EXPERIMENT- 21

AIM: To Simulate and to study of Go Back N protocol

SOFTWARE REQUIREMENTS:

1. NS-2 Simulator

THEORY:

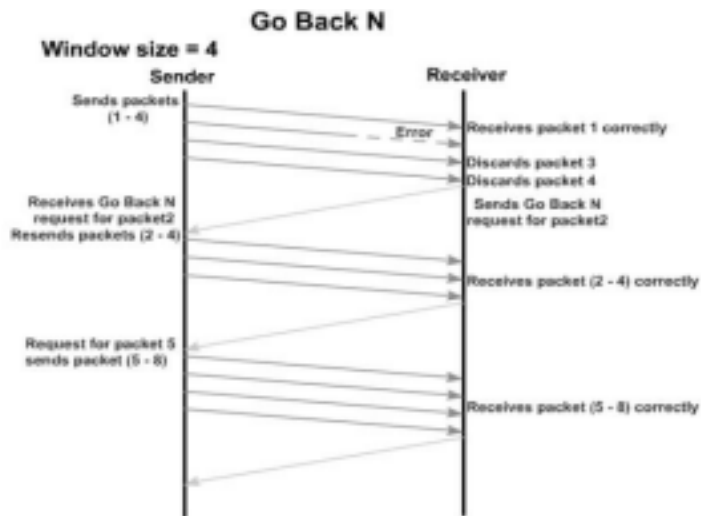
Go Back N is a connection oriented transmission. The sender transmits the frames continuously. Each frame in the buffer has a sequence number starting from 1 and increasing up to the window size. The sender has a window i.e. a buffer to store the frames. This buffer size is the number of frames to be 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 $i+1, i+2, \dots$ which have been sent, but not been acknowledged

ALGORITHM FOR GO BACK N

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 **GOBACKN** transmission 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"
```

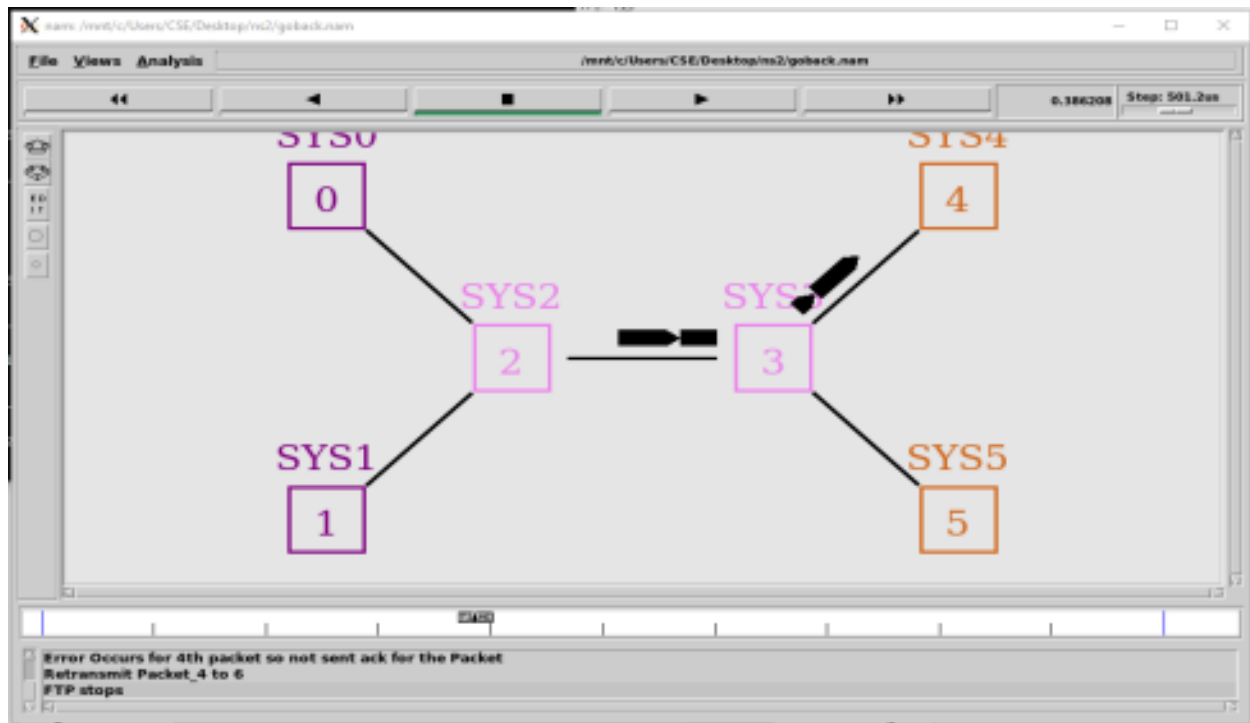
```
set nf [open goback.nam w]
$ns namtrace-all $nf
set f [open goback.tr w]
$ns trace-all $f
$ns duplex-link $n0 $n2 1Mb 20ms DropTail
$ns duplex-link-op $n0 $n2 orient right-down
$ns queue-limit $n0 $n2 5
$ns duplex-link $n1 $n2 1Mb 20ms DropTail
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link $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
set tcp [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.tcl goback.nam
    #puts "running nam..."
    exec nam goback.nam &
    exit 0
}
$ns run

```

OUTPUT:



EXPERIMENT- 22

AIM: To Simulate and to study of **Selective Repeat ARQ** protocol

SOFTWARE REQUIREMENTS:

1. NS-2 Simulator

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 GoBack-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:

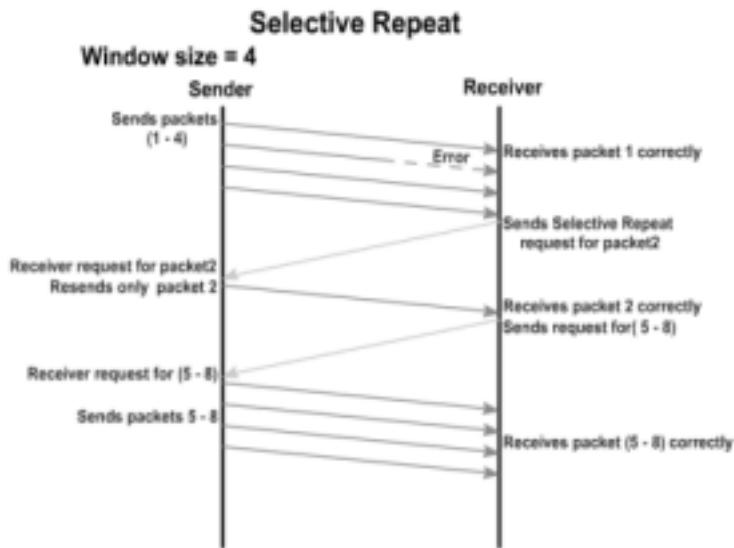
1. Fewer retransmissions.

Disadvantages:

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

ALGORITHM FOR Selective Repeat ARQ

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:

```

#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
  
```



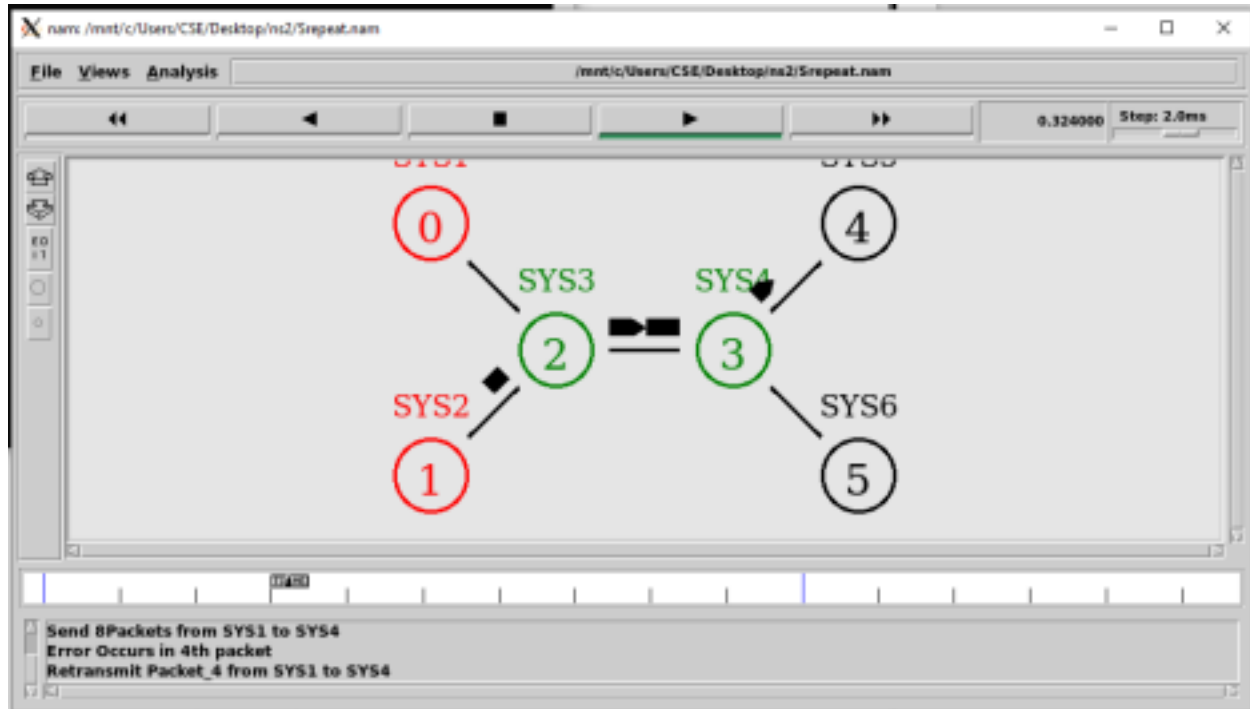
```

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
set tcp [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.tcl srepeat.nam
#puts "running nam..."
exec nam Srepeat.nam &
exit 0

```

```
}  
$ns run
```

OUTPUT:



19115081

EXPERIMENT-10

Aim: Simulating a Local Area Network and LAN topologies.

Instructions for execution:

- 1) We will write a tcl script and simulate it by ns2.
- 2) We begin by specifying the trace files and nam files to be created.
- 3) Define a finish procedure.
- 4) Determine and create the nodes to be used for topology. Here we select 6 nodes:
0,1,2,3,4,5.
- 5) Create links for connecting these nodes.
- 6) Set up the LAN by specifying nodes and assign values for bandwidth, delay, queue

- type and channel to it.
- 7) Set up the TCP and UDP connection(s) and the FTP/CBR (or any other application) that will run over it.
 - 8) Schedule the different events like simulation start and stop, data transmission start and stop.
 - 9) Call the finish procedure and mention the time of end of simulation.
 - 10) Execute the script in terminal by command: **ns script_name.tcl**

Program Code:

```
#lan.tcl
#Lan simulation
set ns [new Simulator]
#define color for data flows
$ns color 1 Blue
$ns color 2 Red
#open tracefiles
set tracefile1 [open out.tr w]
set winfile [open winfile w]
$ns trace-all $tracefile1
#open nam file
set namfile [open out.nam w]
$ns namtrace-all $namfile
#define the finish procedure
proc finish {} {
    global ns tracefile1 namfile
    $ns flush-trace
    close $tracefile1
    close $namfile
    exec nam out.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 $n2 0.3Mb 100ms DropTail
set lan [$ns newLan "$n3 $n4 $n5" 0.5Mb 40ms LL Queue/DropTail MAC/Csma/Cd Channel]
#Give node position
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns simplex-link-op $n2 $n3 orient right
$ns simplex-link-op $n3 $n2 orient left
#set queue size of link(n2-n3) to 20
$ns queue-limit $n2 $n3 20
#setup TCP connection
set tcp [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 packet_size_ 552
#set ftp over tcp connection
set ftp [new Application/FTP]
$ftp attach-agent $tcp
#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

```

```
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
```

35
19115081

```
$cbr set packet_size_ 1000
$cbr set rate_ 0.01Mb
$cbr set random_ false
#scheduling the events
$ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 124.0 "$ftp stop"
$ns at 125.5 "$cbr stop"
proc plotWindow {tcpSource file} {
    global ns
    set time 0.1
    set now [$ns now]
    set cwnd [$tcpSource set cwnd_]
    puts $file "$now $cwnd"
    $ns at [expr $now+$time] "plotWindow $tcpSource $file"
}
$ns at 0.1 "plotWindow $tcp $winfile"
$ns at 125.0 "finish"
$ns run
```

Output:

