GRT INSTITUTE OF ENGINEERING AND TECHNOLOGY, Tiruttani

(Approved by AICTE, New Delhi, Affiliated to Anna University, Chennai)

(An ISO 9001- 2008 certified Institution)

GRT Mahalakshmi Nagar, Chennai-Tirupathi Highway, Tiruttani-631209, Tiruvallur Dt.TN

DEPARTMENT OE ELECTRONICS AND COMMUNICATION ENGINEERING

EC6611-COMPUTER NETWORKS LABORATORY

LAB MANUAL

(Regulation 2013)

Prepared by,

1. **M.KAMESH, AP/ECE**
2. **R.NATHEA, AP/ECE**

Approved by,

Dr.P.SIVAKUMAR, HOD/ECE

GRT INSTITUTE OF ENGINEERING AND TECHNOLOGY, Tiruttani

(Approved by AICTE, New Delhi, Affiliated to Anna University, Chennai)

(An ISO 9001- 2008 certified Institution)

GRT Mahalakshmi Nagar, Chennai-Tirupathi Highway, Tiruttani-631209, Tiruvallur Dt.TN

DEPARTMENT OE ELECTRONICS AND COMMUNICATION ENGINEERING

EC6611-COMPUTER NETWORKS LABORATORY

LAB MANUAL

(Regulation 2013)

Name of the Student : _____

Register Number : _____

Year & Semester : _____

Branch : _____

LIST OF EXPERIMENTS:

1. Implementation of Error Detection / Error Correction Techniques
2. Implementation of Stop and Wait Protocol and sliding window
3. Implementation and study of Goback-N and selective repeat protocols
4. Implementation of High Level Data Link Control
5. Study of Socket Programming and Client – Server model
6. Write a socket Program for Echo/Ping/Talk commands.
7. To create scenario and study the performance of network with CSMA / CA protocol and compare with CSMA/CD protocols.
8. Network Topology - Star, Bus, Ring
9. Implementation of distance vector routing algorithm
10. Implementation of Link state routing algorithm
11. Study of Network simulator (NS) and simulation of Congestion Control Algorithms using NS
12. Encryption and decryption.

CONTENT BEYOND SYLLABUS:

13. Implementing a wireless sensor network.
14. Simulate a Mobile Adhoc Network.
15. Implement Transport Control Protocol in Sensor Network

EXPT.NO.1	STUDY OF NETWORK SIMULATOR (NS)
DATE:	

AIM:
To study about NS2 simulator in detail.

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 and versatile.

BASIC ARCHITECTURE:

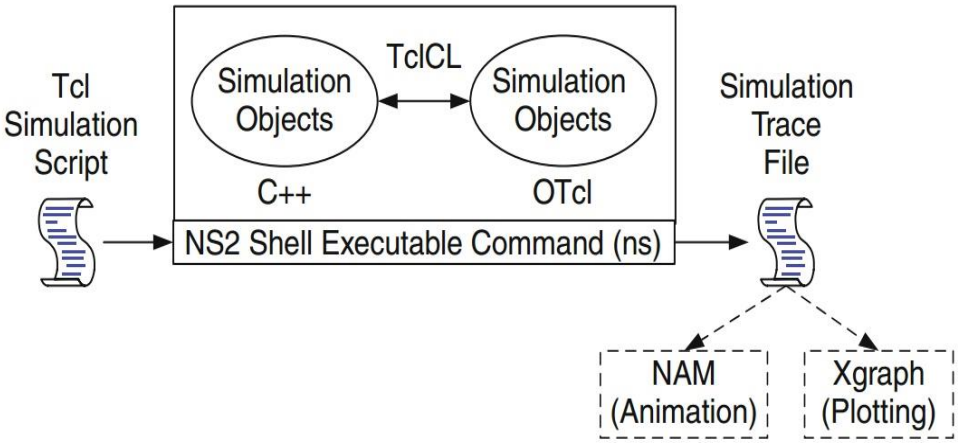


Fig. 2.1. Basic architecture of NS.

Figure 2.1 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 OTcl

objects. 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 are given in Appendices A.1 and A.2, respectively.

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 a OTcl 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, and Mac.
• The strength of Tcl is its simplicity.
• It is not necessary to declare a data type for variable prior to the usage.

Basics of TCL

Syntax: command arg1 arg2 arg3

Hello World!

puts stdout{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 ns simulator.
2. Definition of network nodes, links, queues and topology.
3. Definition of agents and of applications.
4. The nam visualization tool.
5. Tracing and random variables.

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

set namfile [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 nam out.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:

set tcp [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.

set cbr [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 the TCP connection a flow identification of ―1.We shall later give the flow identification of ―2‖ to the UDP connection.

VIVA QUESTIONS:

1. What protocols does ns support?
2. What Is Simulation?
3. Define Network
4. What is meant by Protocol?
5. What are the constituent parts of NS2?

RESULT:

Thus the Network Simulator 2 is studied in detail.

EXPT.NO 2.a	NETWORK TOPOLOGY
DATE:	BUS TOPOLOGY

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

HARDWARE / SOFTWARE REQUIREMENTS:
NS-2

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

ALGORITHM:

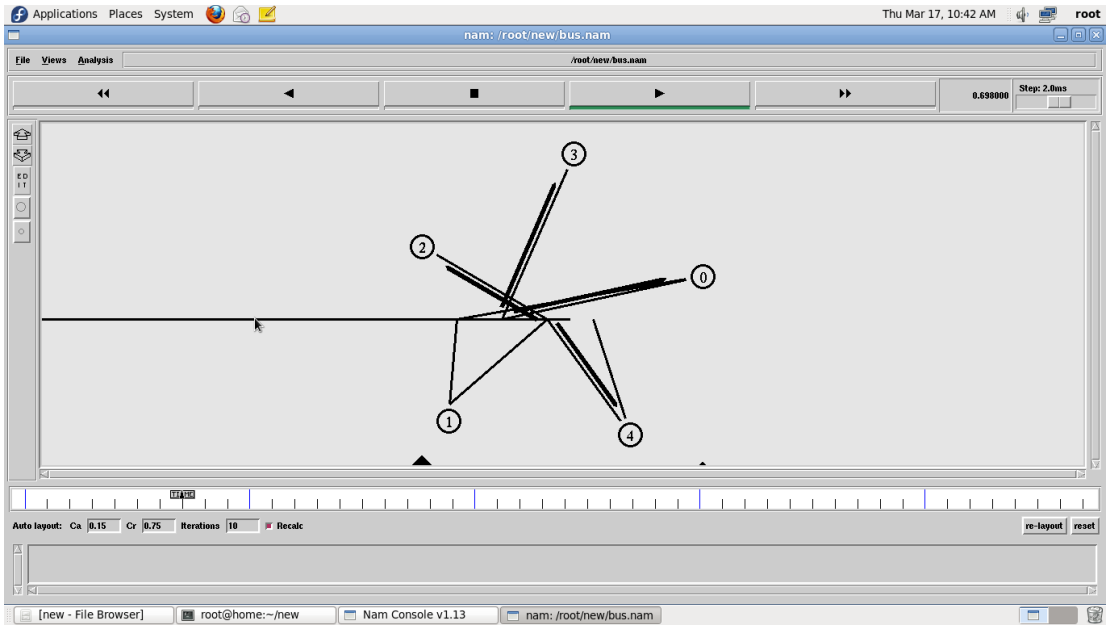
1. Create a simulator object
2. Define different colors for different data flows
3. Open a nam trace file and define finish procedure then close the trace file, and execute nam on trace file.
4. Create five nodes that forms a network numbered from 0 to 4
5. Create duplex links between the nodes and add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

PROGRAM:

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

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

#Define a 'finish' procedure
proc finish {} {
   global ns nf
   $ns flush-trace
   #Close the trace file
   close $nf
   #Executenam on the trace file
   exec nam out.nam &
   exit 0
}
#Create five nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]

#Create Lan between the nodes
set lan0 [$ns newLan "$n0 $n1 $n2 $n3 $n4" 0.5Mb 40ms LL Queue/DropTail MAC/Csma/Cd Channel]
```

```
#Create a TCP agent and attach it to node n0
set tcp0 [new Agent/TCP]
$tcp0 set class_ 1
$ns attach-agent $n1 $tcp0
#Create a TCP Sink agent (a traffic sink) for TCP and attach it to node n3
set sink0 [new Agent/TCPSink]
$ns attach-agent $n3 $sink0
#Connect the traffic sources with the traffic sink
$ns connect $tcp0 $sink0

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

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


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

#Run the simulation
$ns run
```

OUTPUT:



RESULT:
Thus the Bus Topology was Simulated and studied.

EXPT.NO.2b	NETWORK TOPOLOGY
DATE:	RING TOPOLOGY

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 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.

ALGORITHM:

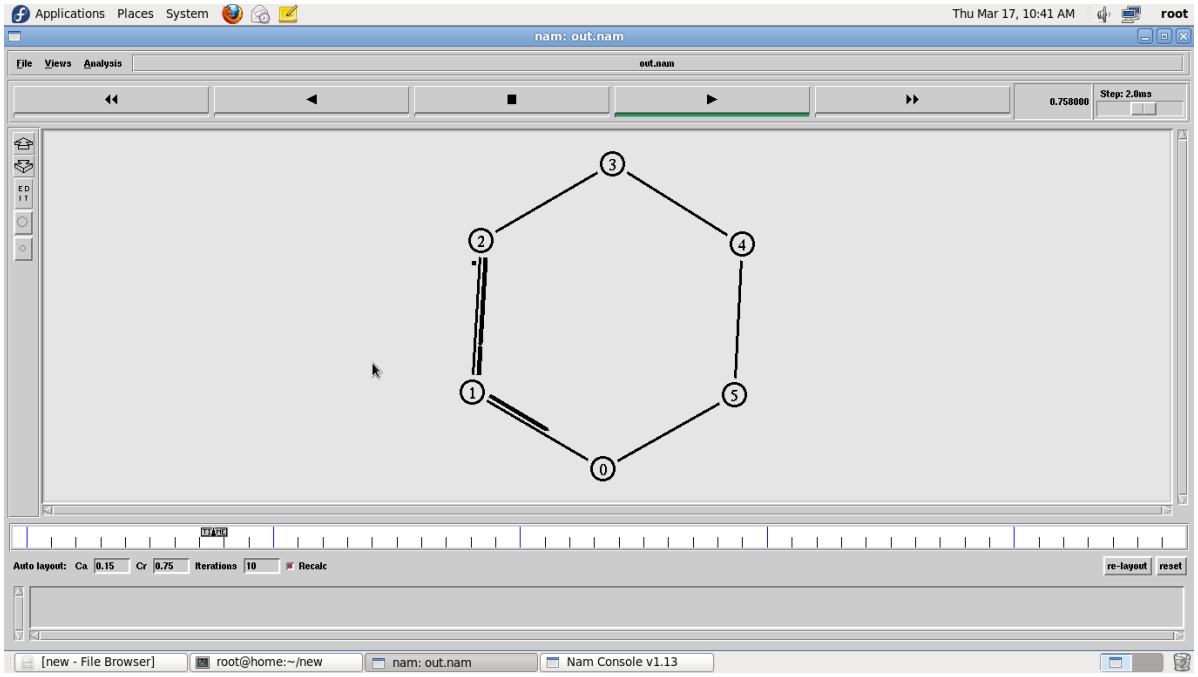
1. Create a simulator object
2. Define different colors for different data flows
3. Open a nam trace file and define finish procedure then close the trace file, and execute nam on trace file.
4. Create five nodes that forms a network numbered from 0 to 4
5. Create duplex links between the nodes to form 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 &
   exit0
}

#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:



RESULT:

Thus the Ring Topology was simulated and studied.

EXPT.NO.2.C	NETWORK TOPOLOGY STAR TOPOLOGY
DATE:	

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

HARDWARE / SOFTWARE REQUIREMENTS:
NS-2

THEORY:
Star networks are one of the most common computer network topologies. In its simplest form, a star network consists of one central switch, hub or computer, which acts as a conduit to transmit messages. This consists of a central node, to which all other nodes are connected; this central node provides a common connection point for all nodes through a hub. In star topology, every node (computer workstation or any other peripheral) is connected to a central node called a hub or switch. The switch is the server and the peripherals are the clients. Thus, the hub and leaf nodes, and the transmission lines between them, form a graph with the topology of a star. If the central node is passive, the originating node must be able to tolerate the reception of an echo of its own transmission, delayed by the two-way transmission time (i.e. to and from the central node) plus any delay generated in the central node. An active star network has an active central node that usually has the means to prevent echo-related problems.

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

ALGORITHM:

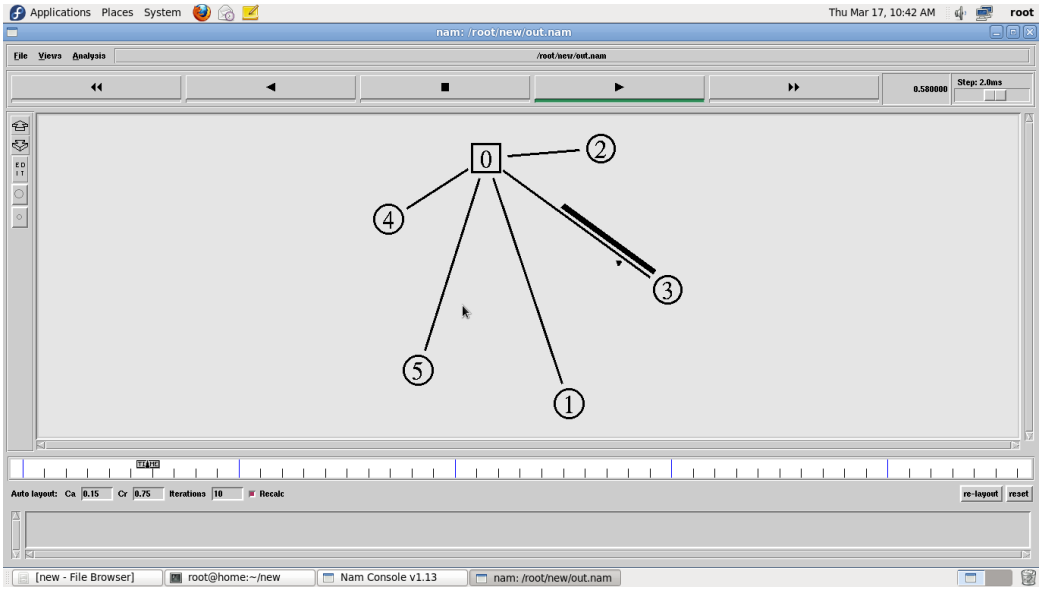
1. Create a simulator object
2. Define different colors for different data flows
3. Open a nam trace file and define finish procedure then close the trace file, and execute nam on trace file.
4. Create six nodes that forms a network numbered from 0 to 5
5. Create duplex links between the nodes to form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

PROGRAM:

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

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

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

```
}

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

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

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

#Create a TCP agent and attach it to node n0
set tcp0 [new Agent/TCP]
$tcp0 set class_ 1
$ns attach-agent $n1 $tcp0
#Create a TCP Sink agent (a traffic sink) for TCP and attach it to node n3
set sink0 [new Agent/TCPSink]
$ns attach-agent $n3 $sink0
#Connect the traffic sources with the traffic sink
$ns connect $tcp0 $sink0
# Create a CBR traffic source and attach it to tcp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.01
$cbr0 attach-agent $tcp0
#Schedule events for the CBR agents
$ns at 0.5 "$cbr0 start"
$ns at 4.5 "$cbr0 stop"
#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"
#Run the simulation
$ns run
```

OUTPUT:



VIVA QUESTIONS:

1.What are the Different topologies available in networks?
2.Which topology requires multipoint conection?
3.Data communication system with in a campus is called as?
4.What is meant by WAN?
5.Explain the working of Ring topology?

RESULT:
Thus the star Topology was simulated and studied.

EXPT.NO.3	SIMULATION OF STOP AND WAIT PROTOCOL AND SLIDING WINDOW PROTOCOL
DATE:	

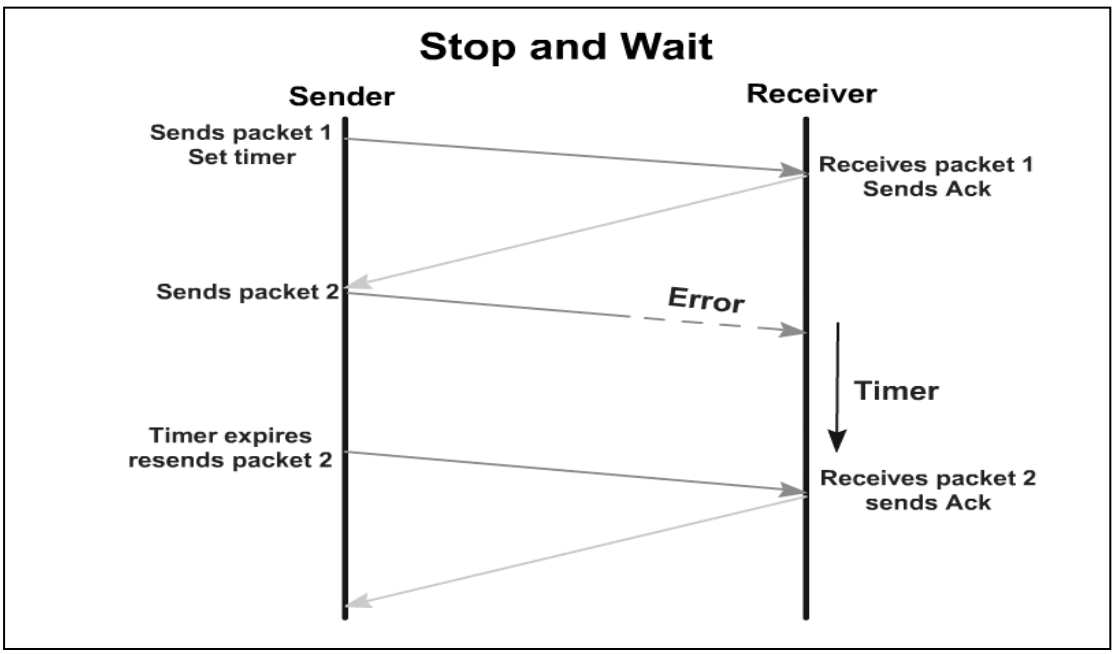
AIM:
To Simulate and to study stop and Wait protocol

SOFTWARE REQUIREMENTS:
NS-2 Simulator

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:

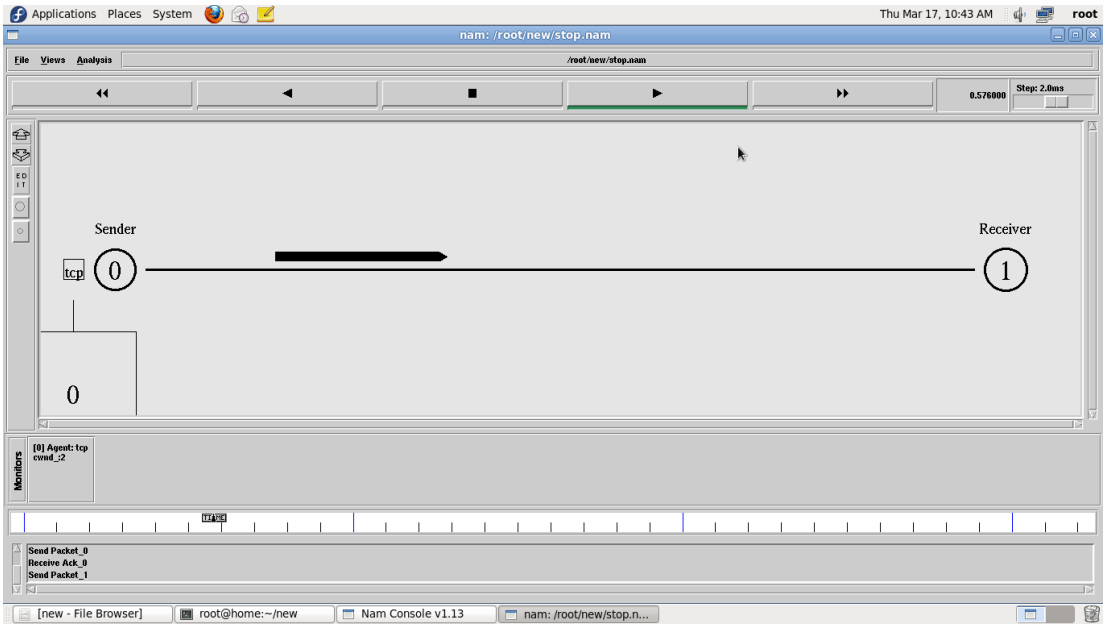
1. Create a simulator object
2. Define different colors for different data flows
3. Open a nam trace file and define finish procedure then close the trace file, and execute nam on trace file.
4. Create two nodes that forms a network numbered 0 and 1
5. Create duplex links between the nodes to form a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

```
# 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:



SLIDING WINDOW PROTOCOL:

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 216 = 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.

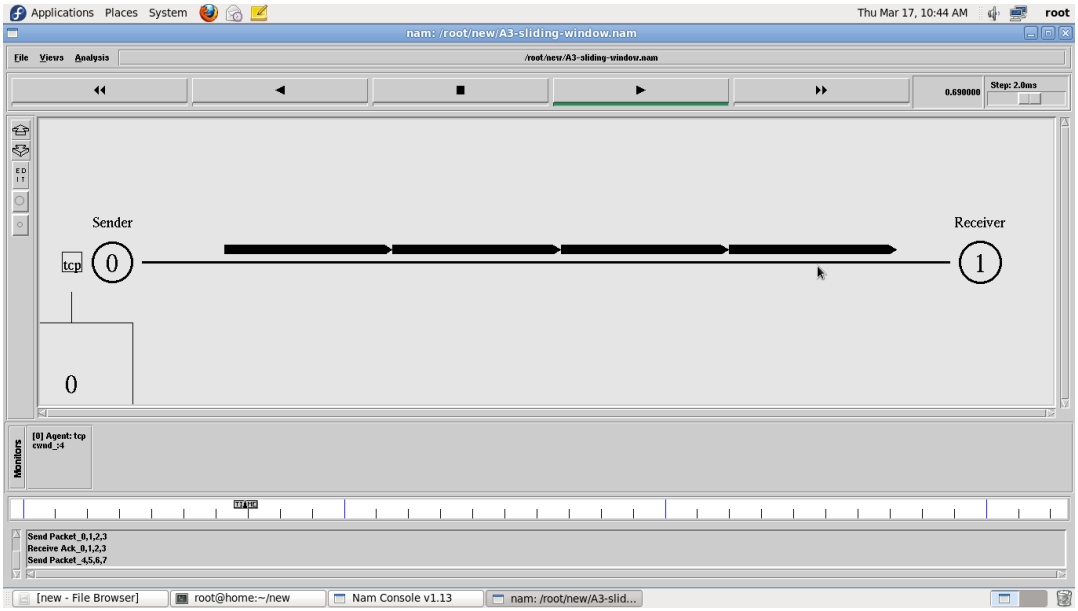
For the highest possible throughput, it is important that the transmitter is not forced to stop sending by the sliding window protocol earlier than one round-trip delay time (RTT). The limit on the amount of data that it can send before stopping to wait for an acknowledgment should be larger than the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

PROGRAM: (Sliding Window Protocol)

```
# 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:



VIVA QUESTIONS:

1.What is ARQ?
2.What is stop and wait protocol?
3.What is stop and wait ARQ?
4.What is usage of sequence number in reliable transmission?
5.What is sliding window?

RESULT:
Thus the Stop and Wait protocol and Sliding window Protocols are Simulated and studied

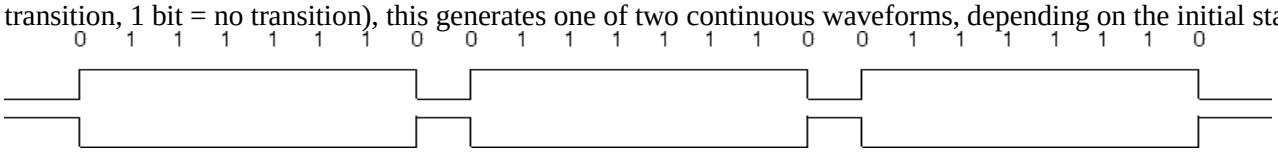
EXPT.NO.4	STUDY OF HIGH-LEVEL DATA LINK CONTROL PROTOCOL(HDLC)
DATE:	

AIM:
To study the concept and different frames of HDLC protocol.

THEORY:
High-Level Data Link Control (**HDLC**) is a bit-oriented code-transparentsynchronous data link layer protocol developed by the International Organization for Standardization (ISO) .
The original ISO standards for HDLC are:
1. ISO 3309 – Frame Structure
2. ISO 4335 – Elements of Procedure
3. ISO 6159 – Unbalanced Classes of Procedure
4. ISO 6256 – Balanced Classes of Procedure

The current standard for HDLC is ISO 13239, which replaces all of those standards.HDLC provides both connection-oriented and connectionless service. HDLC can be used for point to multipoint connections, but is now used almost exclusively to connect one device to another, using what is known asAsynchronous Balanced Mode (ABM). The original master-slave modes Normal Response Mode (NRM) and Asynchronous Response Mode (ARM) are rarely used.HDLC is based on IBM's SDLC protocol, which is the layer 2 protocol for IBM's Systems Network Architecture (SNA). It was extended and standardized by the ITU as LAP, while ANSI named their essentially identical version **ADCCP**.
Derivatives have since appeared in innumerable standards. It was adopted into the X.25 protocol stack as LAPB, into the V.42protocol as LAPM, into the Frame Relay protocol stack as LAPF and into the ISDN protocol stack as LAPD.HDLC was the inspiration for the IEEE 802.2 LLC protocol, and it is the basis for the framing mechanism used with the PPP on synchronous lines, as used by many servers to connect to a WAN, most commonly the Internet. A mildly different version is also used as the control channel for E-carrier (E1) and SONET multichannel telephone lines. Some vendors, such as Cisco, implemented protocols such as Cisco HDLC that used the low-level HDLC framing techniques but added a protocol field to the standard HDLC header. More importantly, HDLC is the default encapsulation for serial interfaces on Cisco routers. It has also been used on Tellabs DXX for destination of Trunk.

FRAMING
HDLC frames can be transmitted over synchronous or asynchronous serial communication links. Those links have no mechanism to mark the beginning or end of a frame, so the beginning and end of each frame has to be identified. This is done by using a frame delimiter, or *flag*, which is a unique sequence of bits that is guaranteed not to be seen inside a frame. This sequence is '01111110', or, in hexadecimal notation, 0x7E. Each frame begins and ends with a frame delimiter. A frame delimiter at the end of a frame may also mark the start of the next frame. A sequence of 7 or more consecutive 1-bits within a frame will cause the frame to be aborted.
When no frames are being transmitted on a simplex or full-duplex synchronous link, a frame delimiter is continuously transmitted on the link. Using the standard NRZI encoding from bits to line levels (0 bit = transition, 1 bit = no transition), this generates one of two continuous waveforms, depending on the initial state:



This is used by modems to train and synchronize their clocks via phase-locked loops. Some protocols allow the 0-bit at the end of a frame delimiter to be shared with the start of the next frame delimiter, i.e. '011111101111110'.
For half-duplex or multi-drop communication, where several transmitters share a line, a receiver on the line will see continuous idling 1-bits in the inter-frame period when no transmitter is active.
Since the flag sequence could appear in user data, such sequences must be modified during transmission to keep the receiver from detecting a false frame delimiter. The receiver must also detect when this has occurred so that the original data stream can be restored before it is passed to higher layer protocols. This can be done using bit stuffing, in which a "0" is added after the occurrence of every "11111" in the data. When the receiver detects these "11111" in the data, it removes the "0" added by the transmitter.

ALGORITHM:

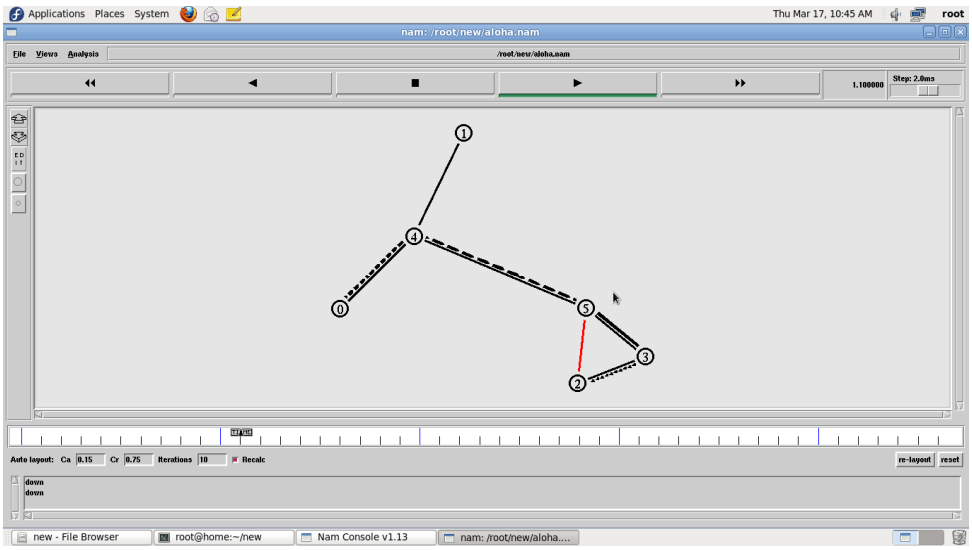
1. Create a simulator object
2. Define different colors for different data flows
3. Open a nam trace file and define finish procedure then close the trace file, and execute nam on trace file.
4. Create six nodes that forms a network numbered from 0 to 5
5. Create duplex links between the nodes
6. Setup UDP Connection between n(0) and n(2)
7. Apply CBR Traffic over UDP
8. Choose distance vector routing protocol as a high level data link control.
9. Make any one of the links to go down to check the working nature of HDLC
10. Schedule events and run the program.

PROGRAM

```
set ns [new Simulator]
#Tell the simulator to use dynamic routing
$ns rtproto DV
$ns macType Mac/Sat/UnslottedAloha
#Open the nam trace file
set nf [open aloha.nam w]
$ns namtrace-all $nf
#Open the output files
set f0 [open aloha.tr w]
$ns trace-all $f0
#Define a finish procedure
proc finish {} {
global ns f0 nf
$ns flush-trace
#Close the trace file
close $f0
close $nf
exec nam aloha.nam &
exit 0
}
# Create six nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
# Create duplex links between nodes with bandwidth and distance
$ns duplex-link $n0 $n4 1Mb 50ms DropTail
$ns duplex-link $n1 $n4 1Mb 50ms DropTail
$ns duplex-link $n2 $n5 1Mb 1ms DropTail
$ns duplex-link $n3 $n5 1Mb 1ms DropTail
$ns duplex-link $n4 $n5 1Mb 50ms DropTail
$ns duplex-link $n2 $n3 1Mb 50ms DropTail
# Create a duplex link between nodes 4 and 5 as queue position
$ns duplex-link-op $n4 $n5 queuePos 0.5
#Create a UDP agent and attach it to node n(0)
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
# Create a CBR traffic source and attach it to udp0
set cbr0 [new Application/Traffic/CBR]
```

```
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0
#Create a Null agent (a traffic sink) and attach it to node n(2)
set null0 [new Agent/Null]
$ns attach-agent $n2 $null0
#Connect the traffic source with the traffic sink
$ns connect $udp0 $null0
#Schedule events for the CBR agent and the network dynamics
$ns at 0.5 "$cbr0 start"
$ns rtmodel-at 1.0 down $n5 $n2
$ns rtmodel-at 2.0 up $n5 $n2
$ns at 4.5 "$cbr0 stop"
#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"
#Run the simulation
$ns run
```

OUTPUT:



VIVA QUESTIONS:

1.What is meant by HDLC?
2.What type of connection does HDLC provides?
3.Mention some of the frames of HDLC?
4.What are the routing protocols used in HDLC
5.Compare HDLC with WSN protocols?

RESULT:

Thus the HDLC is studied and simulated.

EXPT.NO.5	SIMULATION OF DISTANCE VECTOR ROUTING ALGORITHM
DATE:	

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

SOFTWARE REQUIRED:
NS-2

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:

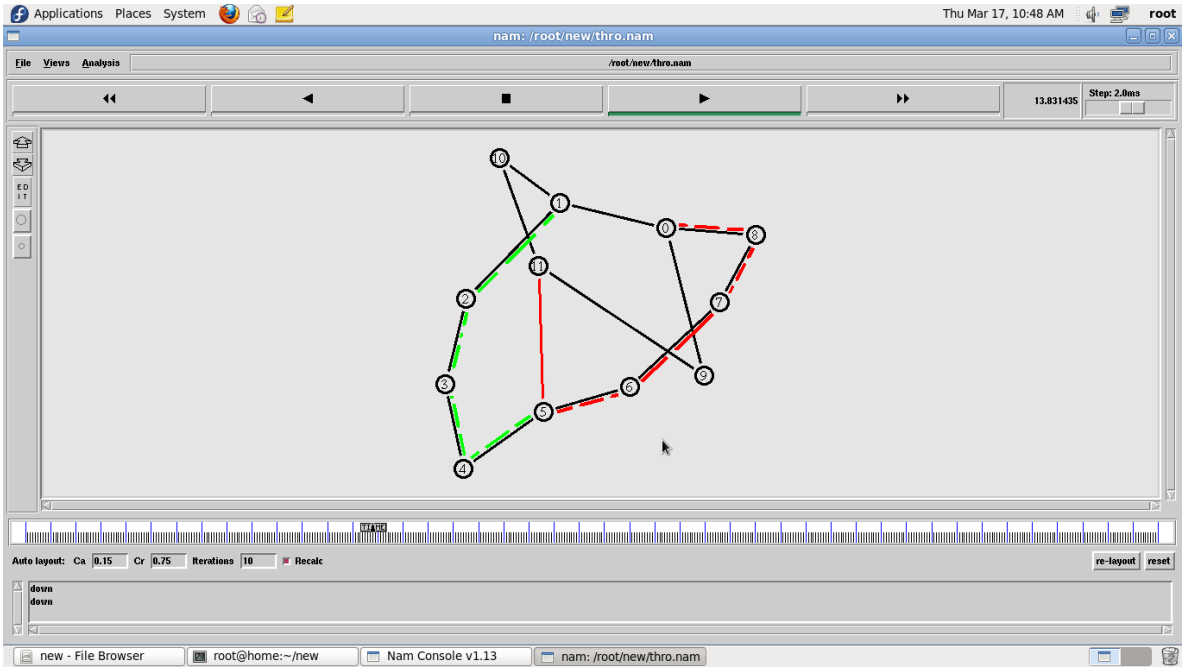
1. Create a simulator object
2. Define different colors for different data flows
3. Open a nam trace file and define finish procedure then close the trace file, and execute nam on trace file.
4. Create n number of nodes using for loop
5. Create duplex links between the nodes
6. Setup UDP Connection between 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:

```
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 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:



VIVA QUESTIONS:

1. Compare connection oriented and connection less protocols.
2.What is MTU?
3.Explain the working of Distance vector routing.
4.Differentiate Proactive and Reactive routing Protocols.
5.What are the different attributes for calculating the cost of a path?

RESULT:

Thus the Distance vector Routing Algorithm was Simulated and studied.

EXPT.NO.6	SIMULATION OF LINK STATE ROUTING ALGORITHM
DATE:	

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

SOFTWARE REQUIRED:
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 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:

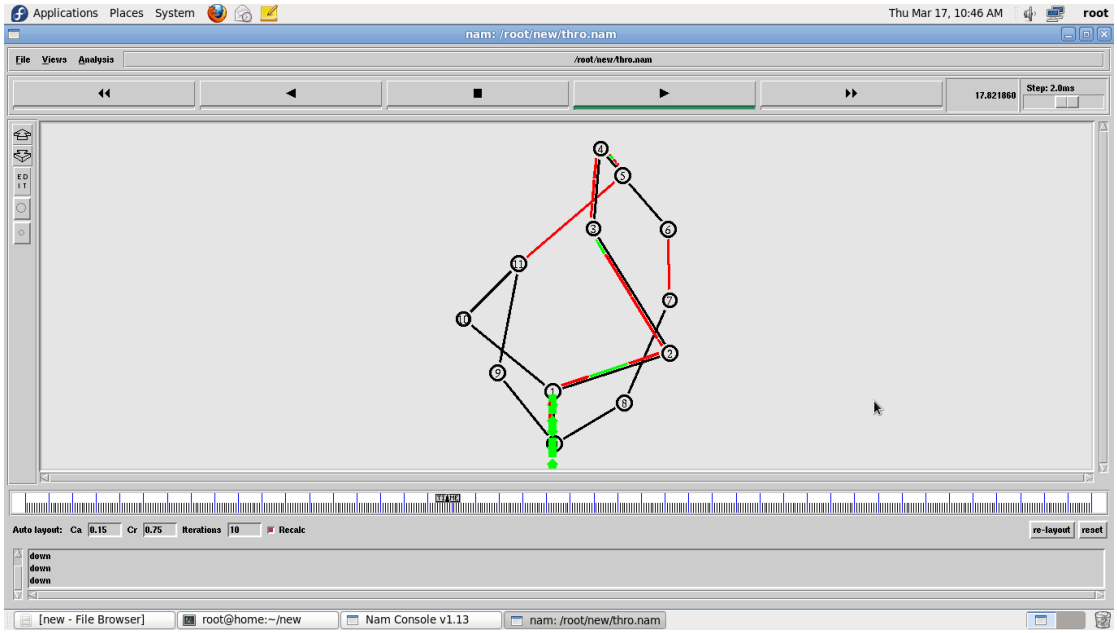
1. Create a simulator object
2. Define different colors for different data flows
3. Open a nam trace file and define finish procedure then close the trace file, and execute nam on trace file.
4. Create n number of nodes using for loop
5. Create duplex links between the nodes
6. Setup UDP Connection between 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:

```
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:



VIVA QUESTIONS:

1. What is Routing?
2. What is Dynamic rouitng?
3. What are the two steps in link state routing?
4. Compare link state and Distance Vector routing
5. What are all the route metric used in Link state routing?

RESULT:

Thus the Link State Routing Algorithm was Simulated and studied.

EXPT.NO.7	DATA ENCRYPTION AND DECRYPTION
DATE:	

AIM:
To implement Data encryption and decryption

SOFTWARE REQUIREMENTS:
Turbo C software

THEORY:
In **encryption,** each letter position in the file text which is given in encrypt mode is changed according to the ascending order of the key text. In **decryption** each letter position in the encrypted file text is changed according to the ascending order of the key text.

ALGORITHM-ENCRYPTION
Get the text to be encrypted (plain text) and key text.

a. Find the length of the plain text.
b. For i=1 to length of plain text
c. Find the binary equivalent of ith character of plain text.
d. Find the binary equivalent of ith character of key text
e. Find the XOR of above two values.

The resulting value will be the encrypted format (cipher text) of the plain text.

ALGORITHM-DECRYPTION
Get the text to be decrypted (cipher text) and key text.
Find the length of the cipher text.
For i=1 to length of cipher text

a. Find the binary equivalent of ith character of cipher text.
b. Find the binary equivalent of ith character of key text
c. Find the XOR of above two values.

The resulting value will be the decrypted format (original plain text) of the cipher plain text.

PROGRAM
```
# include <stdio.h>
#include<conio.h>
void main ( )
{ static int s, i, k,n,c[100];
printf("\n program 1: encryption and 2. decryption");
scanf ("%d", &s);
switch (s)
{ case 1: printf("enter the key value:");
          scanf("%d", &k);
printf("enter the length of text:");
          scanf("%d", &n);
printf("enter the data to be encrypted:");
for (i=0;i<=n;i++)
scanf("%c", &c[i]);
for (i=0;i<=n;i++)
{c[i]=c[i]+k;
if (c[i]>90)
c[i]=c[i]-26;}
printf("encrypted data");
for (i=1;i<=n;i++)
printf("%c", c[i]);
break;
case 2: printf("enter the key value:");
          scanf("%d", &k);
```

```
printf("enter the length of text:");
            scanf("%d", &n);
printf("enter the data to be decrypted:");
for (i=0;i<=n;i++)
scanf("%c", &c[i]);
for (i=0;i<=n;i++)
{c[i]=c[i]-k;
if (c[i]<65)
c[i]=c[i]+26;}
printf("decrypted data");
for (i=1;i<=n;i++)
printf("%c", c[i]);
break;
 case 3: break;
getch ();
}
}
```

OUTPUT:

Program 1: encryption and 2. decryption

1. ENCRYPTION

enter the key value: 1
enter the length of text: 5
enter the data to be encrypted: HELLO
encrypted data : IFMMP

2. DECRYPTION

enter the key value: 1
enter the length of text: 5
enter the data to be decrypted: IFMMP
decrypted data : HELLO

VIVA QUESTIONS:

1.What is meant by Encryption?
2.What is Decryption?
3.Encrypt the word "HELLO" using Caesar cipher.
4.What are the different algorithms available for encryption?
5.What type of information can be secured with Cryptography?

RESULT:

Thus the Data Encryption and Decryption was studied.

EXPT.NO.8	IMPLEMENTATION OF ERROR DETECTION AND ERROR CORRECTION TECHNIQUES
DATE:	

AIM:
To implement error detection and error correction techniques.

SOFTWARE REQUIREMENTS:
Turbo C

THEORY:
The upper layers work on some generalized view of network architecture and are not aware of actual hardware data processing. Hence, the upper layers expect error-free transmission between the systems. Most of the applications would not function expectedly if they receive erroneous data. Applications such as voice and video may not be that affected and with some errors they may still function well. Data-link layer uses some error control mechanism to ensure that frames (data bit streams) are transmitted with certain level of accuracy. But to understand how errors is controlled, it is essential to know what types of errors may occur. CRC is a different approach to detect if the received frame contains valid data. This technique involves binary division of the data bits being sent. The divisor is generated using polynomials. The sender performs a division operation on the bits being sent and calculates the remainder. Before sending the actual bits, the sender adds the remainder at the end of the actual bits. Actual data bits plus the remainder is called a codeword. The sender transmits data bits as codewords.

ALGORITHM:

1. Open Turbo c++ software and type the program for error detection
2. Get the input in the form of bits.
3. Append 16 zeros as redundancy bits.
4. Divide the appended data using a divisor polynomial.
5. The resulting data should be transmitted to the receiver.
6. At the receiver the received data is entered.
7. The same process is repeated at the receiver.
8. If the remainder is zero there is no error otherwise there is some error in the received bits
9. Run the program.

C PROGRAM

```
#include<stdio.h>
char m[50],g[50],r[50],q[50],temp[50];
void caltrans(int);
void crc(int);
void calram();
void shiftl();
int main()
{
int n,i=0;
char ch,flag=0;
printf("Enter the frame bits:");
while((ch=getc(stdin))!='\n')
m[i++]=ch;
n=i;
for(i=0;i<16;i++)
m[n++]='0';
m[n]='\0';
printf("Message after appending 16 zeros:%s",m);
for(i=0;i<=16;i++)
g[i]='0';
g[0]=g[4]=g[11]=g[16]='1';g[17]='\0';
printf("\ngenerator:%s\n",g);
crc(n);
```

```
printf("\n\nquotient:%s",q);
caltrans(n);
printf("\ntransmitted frame:%s",m);
printf("\nEnter transmitted frame:");
scanf("\n%s",m);
printf("CRC checking\n");
crc(n);
printf("\n\nlast remainder:%s",r);
for(i=0;i<16;i++)
if(r[i]!='0')
flag=1;
else
continue;
if(flag==1)
printf("Error during transmission");
else
printf("\n\nReceived freme is correct");
}
void crc(int n)
{
int i,j;
for(i=0;i<n;i++)
temp[i]=m[i];
for(i=0;i<16;i++)
r[i]=m[i];
printf("\nintermediate remainder\n");
for(i=0;i<n-16;i++)
{
if(r[0]=='1')
{
q[i]='1';
calram();
}
else
{
q[i]='0';
shiftl();
}
r[16]=m[17+i];
r[17]='\0';
printf("\nremainder %d:%s",i+1,r);
for(j=0;j<=17;j++)
temp[j]=r[j];
}
q[n-16]='\0';
}
void calram()
{
int i,j;
for(i=1;i<=16;i++)
r[i-1]=((int)temp[i]-48)^((int)g[i]-48)+48;
}
void shiftl()
{
int i;
for(i=1;i<=16;i++)
r[i-1]=r[i];
}
void caltrans(int n)
```

```
{
int i,k=0;
for(i=n-16;i<n;i++)
m[i]=((int)m[i]-48)^((int)r[k++]-48)+48;
m[i]='\0';
}
```

OUTPUT:

Enter the Frame Bits:
1011
The msg after appending 16 zeros:
10110000000000000000
The Transmitted frame is:**10111011000101101011**
Enter the transmitted Frame
10111011000101101011
Received msg:**10111011000101101011**
The Remainder is:**0000000000000000**
Received frame is correct.

VIVA QUESTIONS:

1. What are the types of errors?
2. What is Error Detection? What are its methods?
3. What is Redundancy?
4. What is CRC?
5. What is Checksum?

RESULT:
Thus the error detection and error correction is implemented successfully.

EXPT.NO: 9	PERFORMANCE ANALYSIS OF CSMA/CA AND CSMA/CD PROTOCOLS
DATE:	

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

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:

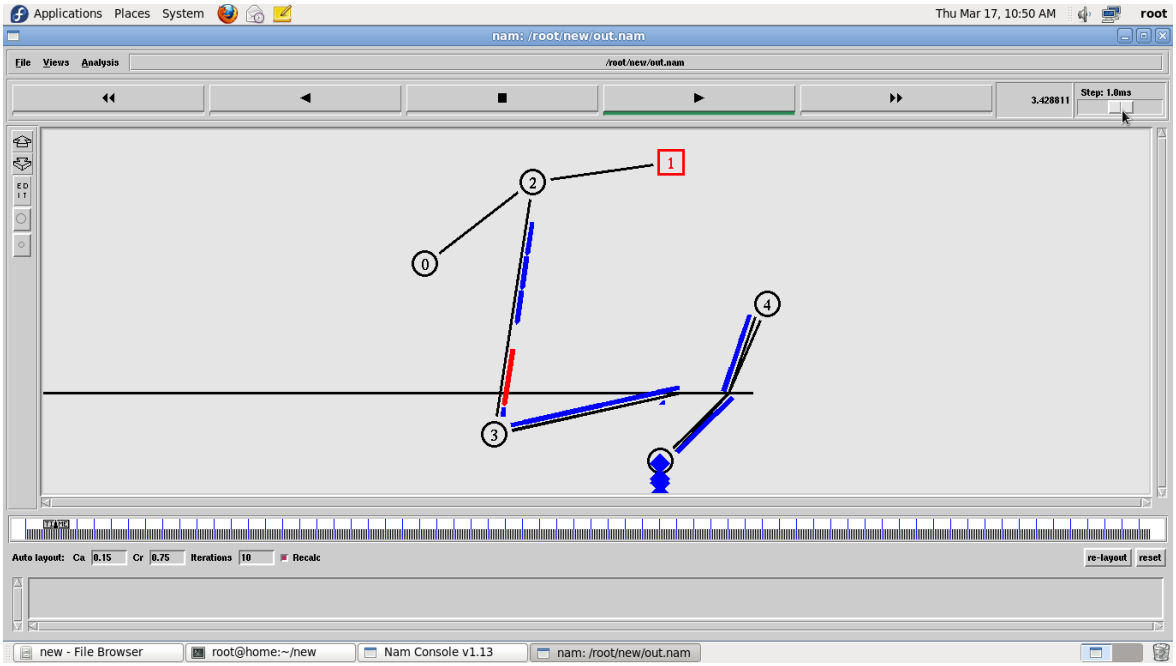
CSMA/CA

```
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/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 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 $now+$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:

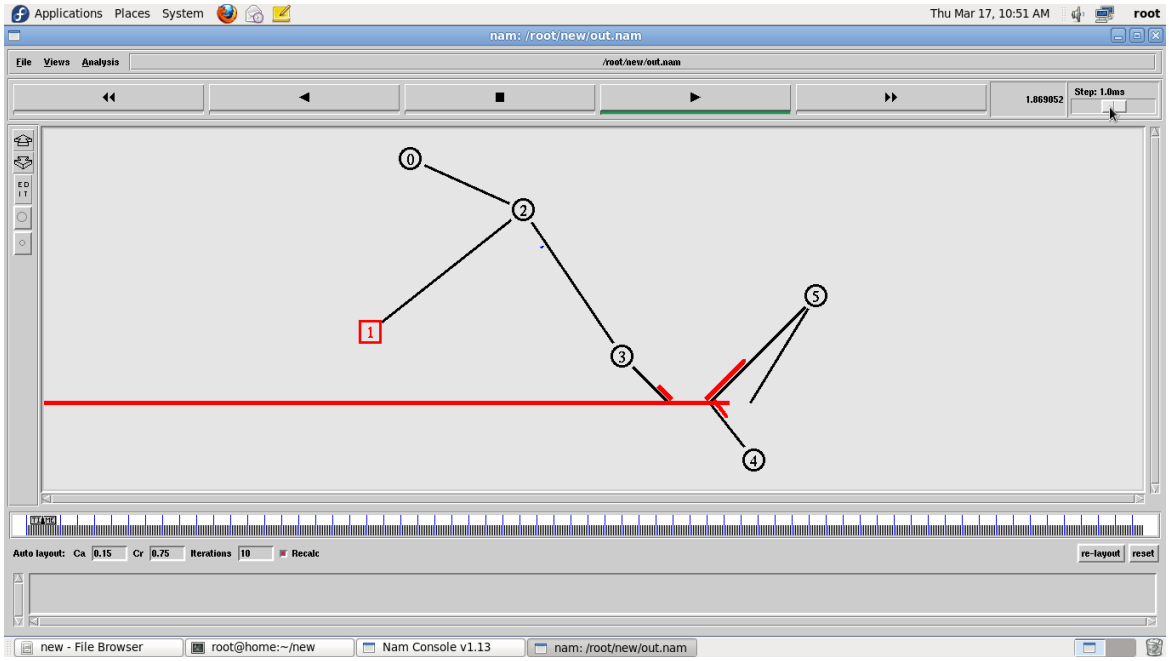


CSMA/CD

```
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
$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 $now+$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:



VIVA QUESTIONS:

1.Explain the concept of CSMA?
2.Compare CSMA/CA and CSMA/CD.
3.What is the function of MAC layer?
4.What is DCF?
5.how does the collision is avoided by CSMA/CD?

RESULT:

Thus, the performance of CSMA / CD protocol was studied through simulation.

EXPT.NO 10	SIMULATION OF GO BACK N PROTOCOL AND SELECTIVE REPEAT PROTOCOLS
DATE:	

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

SOFTWARE REQUIREMENTS:
1. NS-2 Simulator

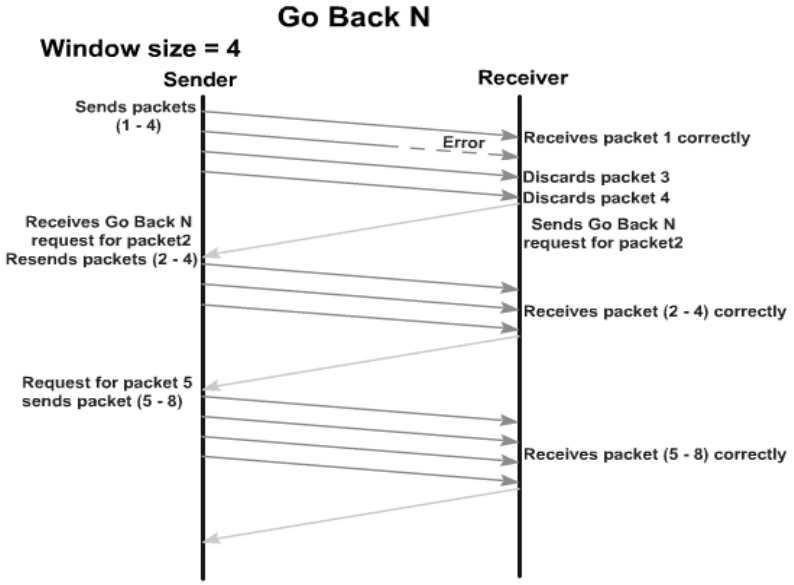
THEORY:

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,... 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.

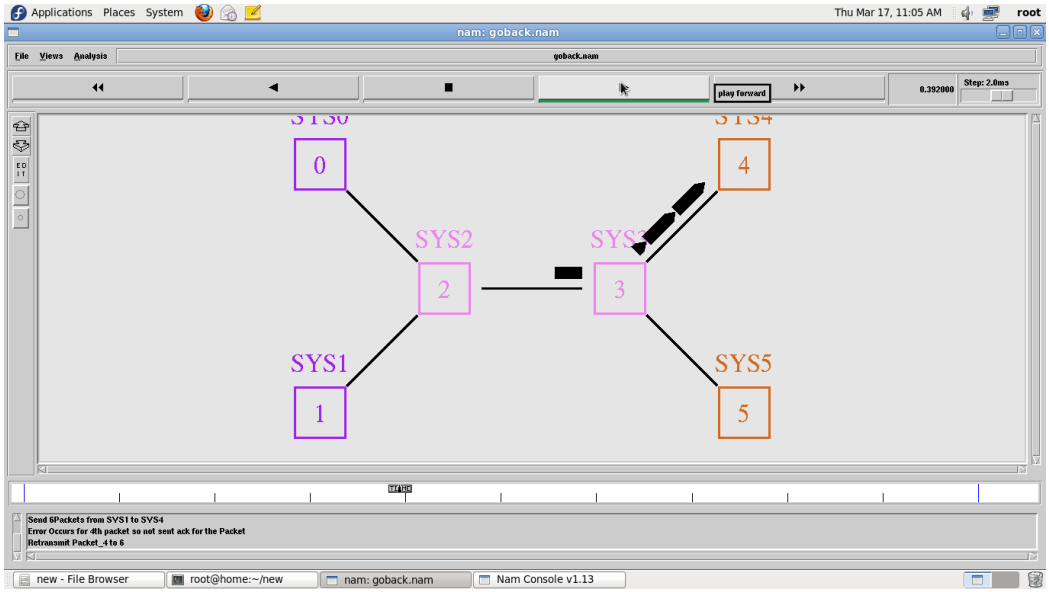
9. If the frames are errorless in the next transmission and if the acknowledgment is error free, the window slides by the number of error-free frames being transmitted.
10. If the acknowledgment is transmitted with error, all the frames of window at source are retransmitted, and window doesn't slide.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

 # PROGRAM FOR GOBACK N:

```
#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:



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:

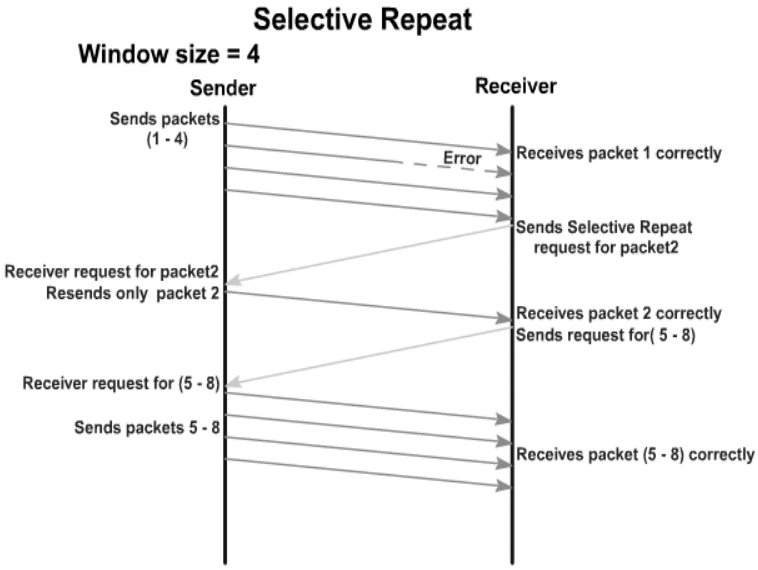
1. Fewer retransmissions.

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 t**ransmission flow control protocol.



#PROGRAM FOR SELECTIVE REPEAT:

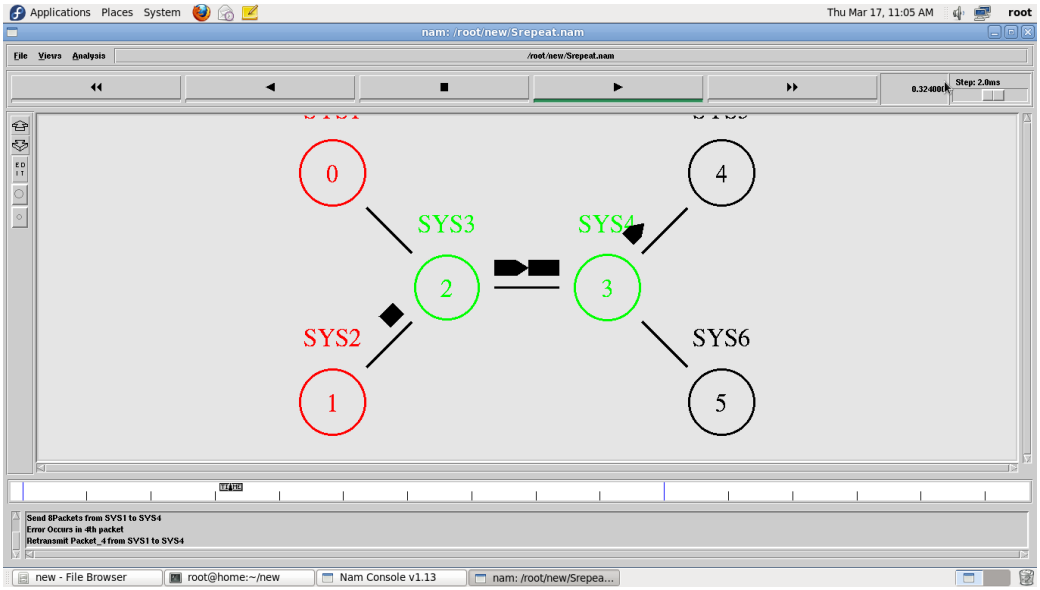
```
#send packets one by one
set ns [new Simulator]
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
$n0 color "red"
$n1 color "red"
$n2 color "green"
$n3 color "green"
$n4 color "black"
$n5 color "black"
$n0 shape circle ;
$n1 shape circle ;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

VIVA QUESTIONS:

1.What is Go back-N protocol?
2.What is Go back-N protocol ARQ?
3.What is flow control?
4.What is fixed size framing?
5.What is pipelining?

OUTPUT:



RESULT:
Thus the Go back N and Selective Repeat protocols were Simulated and studied.

EXPT.NO 11	STUDY OF SOCKET PROGRAMMING AND CLIENT-SERVER MODEL
DATE:	

AIM:
To study the client/server model of socket programming.

SOFTWARE REQUIREMENTS:
Java

THEORY:

Socket creates an endpoint for communication and returns a descriptor.
The domain parameter specifies a common domain this selects the protocol family which will be used for communication.

ALGORITHM:

1. Create a server socket.
2. Then create a client socket.
3. DataInputStream and DataOutputStream is to abstract different ways to input and output whether the stream is a file, a web page.
4. Specify the IP address and port number for socket.
5. Close all streams.
6. Close server and client socket.
7. stop

PROGRAM:

// CLIENT // Client.java

```
import java.io.*;
import java.net.*;
class Client{
	public static void main(String args[]){
		String data;
		Socket socket1;
		DataInputStream key_in=new DataInputStream(System.in);
		DataInputStream s_in;
		PrintStream s_out;

		System.out.println("Client:");
		try{
			socket1=new Socket("127.0.0.1",9000);
			s_in=new DataInputStream(socket1.getInputStream());
			s_out=new PrintStream(socket1.getOutputStream());
			while(true){
				data=key_in.readLine();
				if(data.charAt(0)=='Q') break;
				s_out.println(data);
				System.out.println("Client: "+s_in.readLine());
			}
```

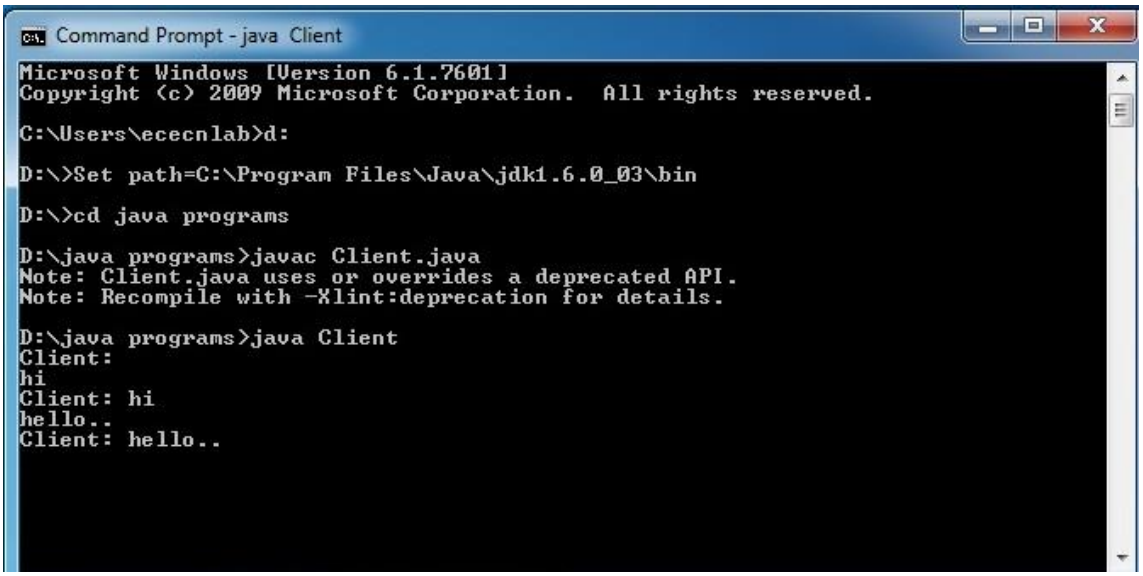
```
                        socket1.close();
                }
                catch(Exception e){}
        }
}
```

// SERVER // Server. Java

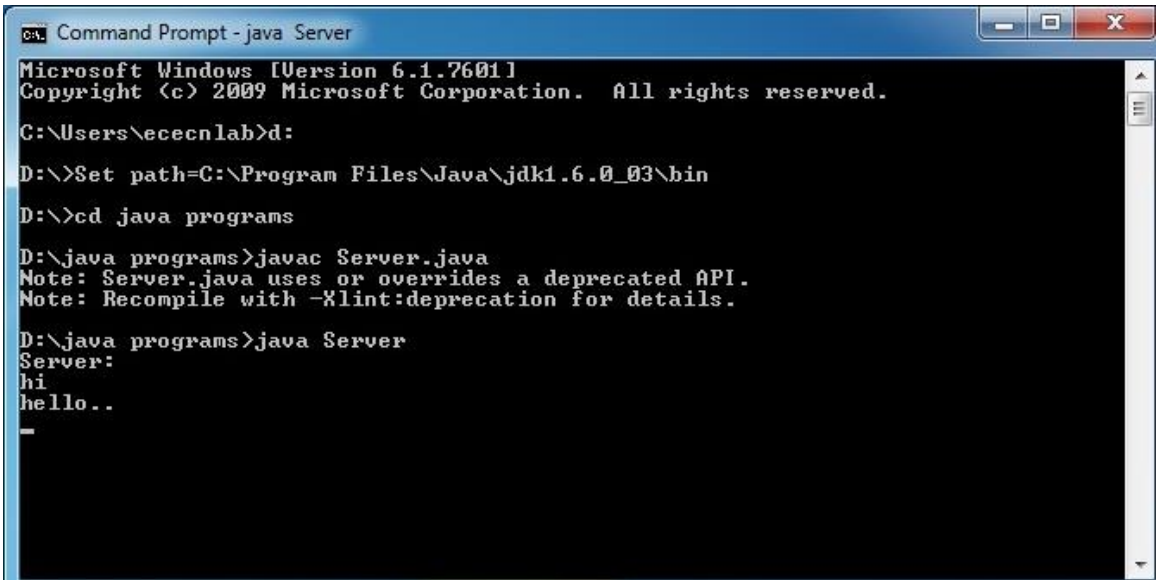
```
import java.io.*;
import java.net.*;

class Server {
        static Socket socket1;
        static DataInputStream s_in;
        static PrintStream s_out;
        public static void main(String args[]) throws IOException{
                String data="";
                ServerSocket sc;
                System.out.println("Server:");
                sc=new ServerSocket(9000);
                while(true){
                        try{

                        socket1=sc.accept();
                        s_in=new DataInputStream(socket1.getInputStream());
                        s_out=new PrintStream(socket1.getOutputStream());
                        }
                        catch(IOException e){}
                        try{
                                while(!(data.equals("exit"))){
                                        data=s_in.readLine();
                                        if(data==null) {
                                                System.out.println("Server: ");
                                                break;
                                        }
                                        s_out.println(data);
                                        System.out.println(data);
                                }
                                socket1.close();
                        }
                        catch(IOException e){}
                }
        }
 }
```

OUTPUT:

CLIENT:



SERVER:



VIVA QUESTIONS:

1.Define Socket
2.What is socket programming?
3.What is the function of command bind?
4.What is the syntax for connecting Client and Server
5.What is the command to assign port number to client and server?.

RESULT:

Thus the Socket Client-Server model is simulated and studied.

EXPT.NO.12	SOCKET PROGRAM FOR ECHO/PING/TALK COMMANDS
DATE	

AIM:
To write a Java program to implement ping command.

SOFTWARE REQUIREMENTS:
Java

THEORY:
ICMP ping flood is a kind of DOS attack that can be performed on remote machines connected via a network. It involves sending a large number of ping echo requests (packets) to the target system such that it is not able to tackle so fast. So the result is that the host either gets too busy into replying these echo requests that it gets no time to serve its original purpose, or it might crash or something similar. So if a machine connected to the internet gets flooded by such a large quantity of echo packets then it wont be able to process other network activities it was intended to, and will keep very busy in replying to the echo requests. Different machines handle this differently depending on the network security and kind of operating system setup etc. Slowly all machines connected to the internet are securing themselves from such dos attacks. The most common technique is the use of a firewall, that will block the sending ip if it receives an unexpected amount of echo request. Other techniques involve not replying to ping packets at all from over the internet. Each of the techniques has its own pros and cons and has limitations.If its a website or online network then using the firewalls or other blocking policies would work well. If its a broadband router of a home user that is connected to internet, then flooding such a device will, depending on the make and model, either crash it or make it so slow that the users would be thrown off. To test this out flood your own broadband router.

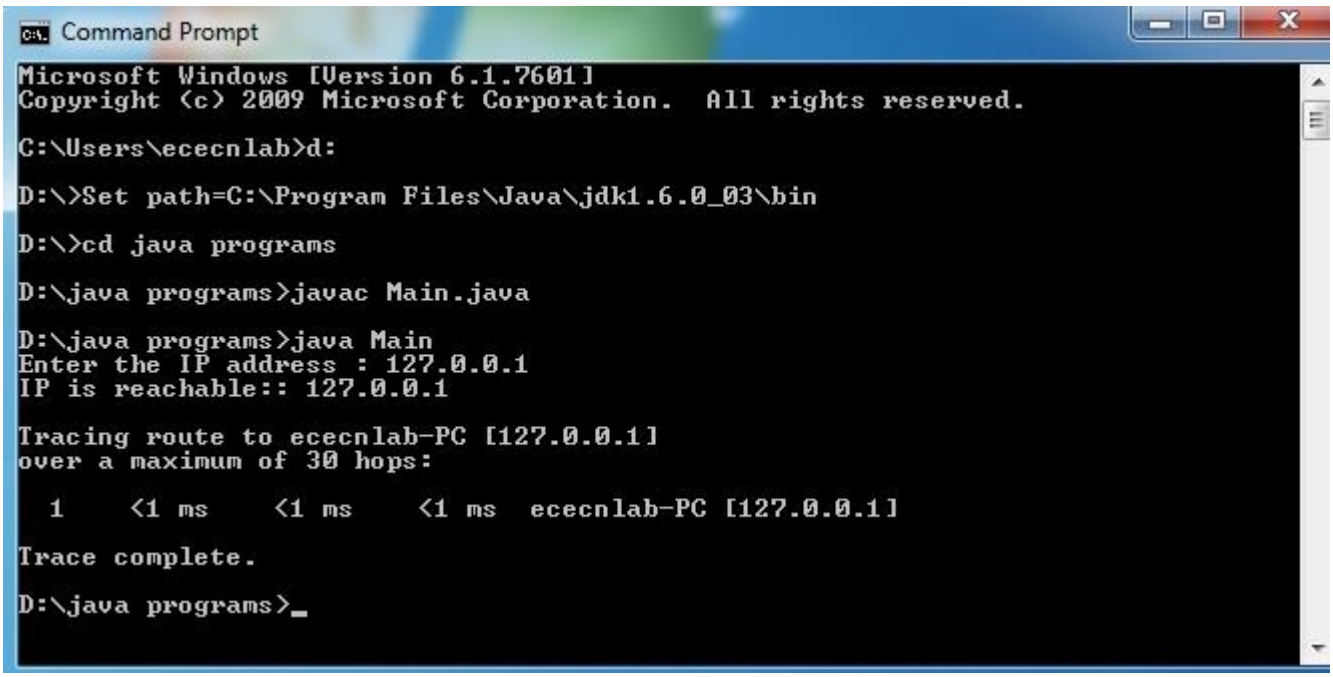
ALGORITHM:

1. Start the program
2. Create process p.
3. Using Runtime.getRuntiume() command get IP address from local Network.
4. Run and execute the program.
5. Stop.

PROGRAM:
```
import java.io.*;
import java.net.*;
import java.util.*;
public class Main
{
public static void main(String[] args)
{
   try
   {
         BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
         String ipaddr;
         System.out.print("Enter the IP address : ");
         ipaddr = br.readLine();
         boolean reachable = (java.lang.Runtime.getRuntime().exec("ping -n 1
"+ipaddr).waitFor()==0);
      if(reachable)
         {
```

```
                System.out.println("IP is reachable:: "+ipaddr);
                    try
                    {
                        Process p = Runtime.getRuntime().exec("Tracert "+ipaddr);
                        Scanner scan = new Scanner(p.getInputStream());
                        while(scan.hasNextLine())
                        {
                            System.out.println(scan.nextLine());
                        }
                    }
                    catch(Exception ex)
                    {
                        System.out.println("Error "+ex);
                    }
            }
        else
            {
            System.out.println("IP is not reachable: "+ipaddr);
        }

    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
}
}
```

OUTPUT:



```
Command Prompt
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\ececnlab>d:

D:\>Set path=C:\Program Files\Java\jdk1.6.0_03\bin

D:\>cd java programs

D:\java programs>javac Main.java

D:\java programs>java Main
Enter the IP address : 127.0.0.1
IP is reachable:: 127.0.0.1

Tracing route to ececnlab-PC [127.0.0.1]
over a maximum of 30 hops:

  1    <1 ms    <1 ms    <1 ms  ececnlab-PC [127.0.0.1]

Trace complete.

D:\java programs>_
```

VIVA QUESTIONS:

1.What does a Socket consists of?
2.What is the concept of Echo?
3.Explain Ping?
4.Mention some advantages of Java Sockets.
5.How do you open a Socket?

RESULT:

Thus the Echo/Ping/Talk concepts are simulated and studied.

EXPT.NO.13	IMPLEMENTING A WIRELESS SENSOR NETWORK
DATE:	

AIM:

To simulate a wireless sensor network using NS2.

SOFTWARE REQUIREMENTS:

NS-2 Simulator

THEORY:

A wireless sensor network (WSN) consists of a large number of small sensor nodes that are deployed in the area in which a factor is to be monitored. In wireless sensor network, energy model is one of the optional attributes of a node. The energy model denotes the level of energy in a mobile node. The components required for designing energy model includes initialEnergy, txPower, rxPower, and idlePower. The "initialEnergy" represents the level of energy the node has at the initial stage of simulation. "txPower" and "rxPower" denotes the energy consumed for transmitting and receiving the packets. If the node is a sensor, the energy model should include a special component called "sensePower". It denotes the energy consumed during the sensing operation. Apart from these components, it is important to specify the communication range (RXThresh_) and sensing range of a node (CSThresh_). The sample 18.tcl designs a WSN in which sensor nodes are configured with different communication and sensing range. Base Station is configured with highest communication range. Data Transmission is established between nodes using UDP agent and CBR traffic.

ALGORITHM:

1. Create a simulator object
2. Define a setting options for wireless channel
3. Create trace file and name file
4. Setup topography object and nodes
5. Provide initial location of mobile nodes
6. Setup a UDP connection between nodes
7. Printing the window size

PROGRAM:

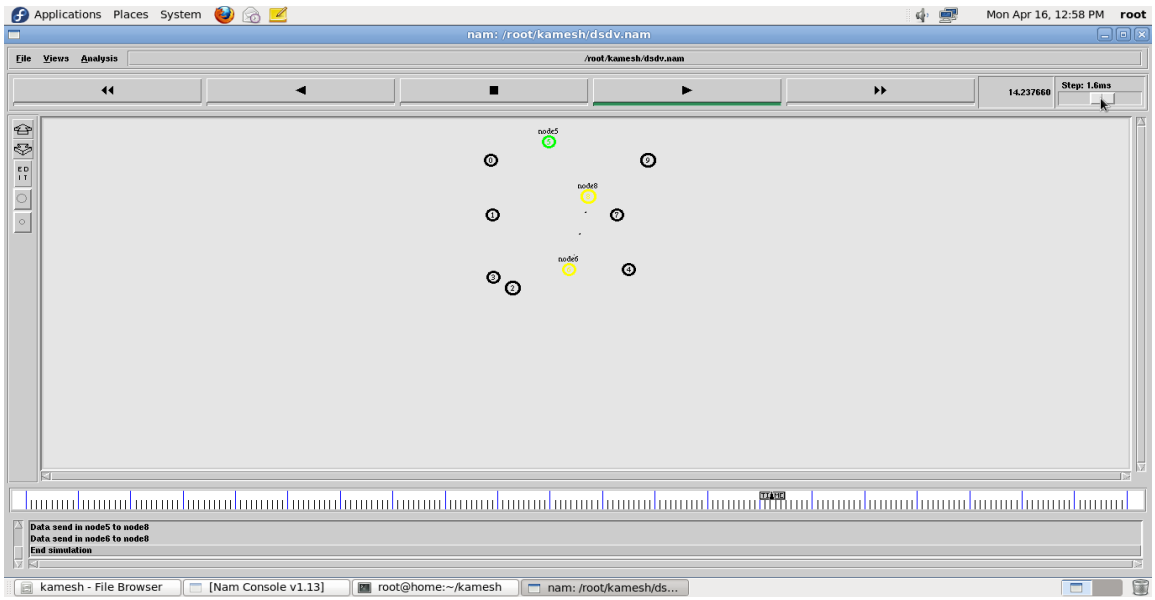
```
# Define setting options
set val(chan)         Channel/WirelessChannel    ;# channel type
set val(prop)         Propagation/TwoRayGround   ;# radio-propagation model
set val(netif)        Phy/WirelessPhy            ;# network interface type
set val(mac)          Mac/802_11                 ;# MAC type
set val(ifq)          Queue/DropTail/PriQueue    ;# interface queue type
set val(ll)           LL                         ;# link layer type
set val(ant)          Antenna/OmniAntenna        ;# antenna model
set val(ifqlen)       50                         ;# max packet in ifq
set val(nn)           10                         ;# number of mobilenodes
set val(rp)           DSDV                       ;# routing protocol
set val(x)            500                        ;# X dimension of topography
set val(y)            400                        ;# Y dimension of topography
set val(stop)         150                        ;# time of simulation end
set ns            [new Simulator]
#Creating trace file and nam file
set tracefd       [open dsdv.tr w]
set windowVsTime2 [open win.tr w]
set namtrace      [open dsdv.nam w]
$ns trace-all $tracefd
$ns namtrace-all-wireless $namtrace $val(x) $val(y)
# set up topography object
set topo       [new Topography]
$topo load_flatgrid $val(x) $val(y)
create-god $val(nn)
# configure the nodes
```

```
        $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 $topo \
                -agentTrace ON \
                -routerTrace ON \
                -macTrace OFF \
                -movementTrace ON
          for {set i 0} {$i < $val(nn) } { incr i } {
           set node_($i) [$ns node]
      }
# Provide initial location of mobilenodes
$node_(0) set X_ 5.0
$node_(0) set Y_ 800.0
$node_(0) set Z_ 0.0
$node_(1) set X_ 8.0
$node_(1) set Y_ 650.0
$node_(1) set Z_ 0.0
$node_(2) set X_ 60.0
$node_(2) set Y_ 450.0
$node_(2) set Z_ 0.0
$node_(3) set X_  10.0
$node_(3) set Y_  480.0
$node_(3) set Z_  0.0
#another four
$node_(4) set X_ 350.0
$node_(4) set Y_ 500.0
$node_(4) set Z_ 0.0
$node_(5) set X_ 150.0
$node_(5) set Y_ 850.0
$node_(5) set Z_ 0.0
$node_(6) set X_ 200.0
$node_(6) set Y_ 500.0
$node_(6) set Z_ 0.0
#$node_(7) set X_  320.0
$node_(7) set X_  320.0
$node_(7) set Y_  650.0
$node_(7) set Z_  0.0
#another four
$node_(8) set X_ 250.0
$node_(8) set Y_ 700.0
$node_(8) set Z_ 0.0
$node_(9) set X_ 400.0
$node_(9) set Y_ 800.0
$node_(9) set Z_ 0.0
##################################
#copy of the data
##################################
#####Attack Node
$node_(5) color Green
$ns at 10.0 "$node_(5) color Green"
$ns at 10.0 "$node_(5) label EndUser"
# Set a udp connection between node_(5) and node_(8)
```

```
set udp [new Agent/UDP]
set sink [new Agent/LossMonitor]
 #$ns attach-agent $node_(5) $udp
 #$ns attach-agent $node_(8) $sink
 $ns attach-agent $node_(8) $udp
 $ns attach-agent $node_(5) $sink
 $ns connect $udp $sink
 set cbr [new Application/Traffic/CBR]
 $cbr attach-agent $udp
 $node_(8) color Green
 $ns at 10.0 "$node_(8) color yellow"
 $ns at 10.0 "$node_(5) label  node5 "
 $ns at 10.0 "$node_(8) label  node8 "
 $ns at 10.0 "$cbr start"
 $ns at 10.45 "$cbr stop"
 # Set a udp connection between node_(8) and node_(6)
 set udp1 [new Agent/UDP]
 set sink1 [new Agent/LossMonitor]
 $ns attach-agent $node_(8) $udp1
 $ns attach-agent $node_(6) $sink1
 $ns connect $udp1 $sink1
 set cbr1 [new Application/Traffic/CBR]
 $cbr1 attach-agent $udp1
 $node_(6) color yellow
 $ns at 10.0 "$node_(6) color yellow"
 $ns at 10.0 "$node_(6) label  node6 "
 $ns at 10.0 "$cbr1 start"
 $ns at 10.45 "$cbr1 stop"
 # Set a udp connection between node_(8) and node_(6)
 set udp2 [new Agent/UDP]
 set sink2 [new Agent/LossMonitor]
 $ns attach-agent $node_(5) $udp2
 $ns attach-agent $node_(8) $sink2
 $ns connect $udp2 $sink2
 set cbr2 [new Application/Traffic/CBR]
 $cbr2 attach-agent $udp2
 $node_(6) color yellow
 $ns at 12.0 "$cbr2 start"
 $ns at 13.45 "$cbr2 stop"
  # Set a udp connection between node_(8) and node_(6)
  set udp3 [new Agent/UDP]
  set sink3 [new Agent/LossMonitor]
  $ns attach-agent $node_(8) $udp3
  $ns attach-agent $node_(6) $sink3
  $ns connect $udp3 $sink3
  set cbr3 [new Application/Traffic/CBR]
  $cbr3 attach-agent $udp1
  $node_(6) color yellow
  $ns at 14.0 "$cbr3 start"
  $ns at 16.45 "$cbr3 stop"
# Printing the window size
proc plotWindow {tcpSource file} {
global nss
set time 0.01
set now [$ns now]
set cwnd [$tcpSource set cwnd_]
puts $file "$now $cwnd"
$ns at [expr $now+$time] "plotWindow $tcpSource $file" }
#$ns at 10.1 "plotWindow $tcp $windowVsTime2"
```

```
        # Define node initial position in nam
        for {set i 0} {$i < $val(nn)} { incr i } {
        # 30 defines the node size for nam
        $ns initial_node_pos $node_($i) 30
        }
        # Telling nodes when the simulation ends
        for {set i 0} {$i < $val(nn) } { incr i } {
           #$ns at $val(stop) "$node_($i) reset";
                #$ns at 200.25 "$node_($i) reset";
                $ns at 21.25 "$node_($i) reset";
        }
        $ns at 0.00 "$ns trace-annotate \"Wireless Mac Ptotocol \""
        $ns at 10.0 "$ns trace-annotate \" Data send in  node5 to node8 \""
        $ns at 10.0 "$ns trace-annotate \" Data send in  node6 to node8 \""
        $ns at 10.45 "$ns trace-annotate \" Data Collision \""
        $ns at 12.00 "$ns trace-annotate \"Data send in  node5 to node8\""
        $ns at 14.00 "$ns trace-annotate \"Data send in  node6 to node8\""
        $ns at 21.25 "$ns trace-annotate \"End simulation\""
        $ns at 21.25 "$ns nam-end-wireless 21.01"
        $ns at 21.25 "stop"
        $ns at 22.01 "puts \"end simulation\" ; $ns halt"
        proc stop {} {
           global ns tracefd namtrace
           $ns flush-trace
           close $tracefd
           close $namtrace
                exec nam dsdv.nam &
        exit 0
        }
$ns run
```

OUTPUT:



VIVA QUESTIONS:

1. What is a Wireless Sensor Network?
2. How is the Network Configured?
3. How long does a node operates in a battery power?
4. What is the range of a wireless sensor node?
5. Can I use the wireless nodes in outside environment?

RESULT:

Thus the wireless sensor network is successfully implemented using NS2 simulator.

EXPT.NO: 14	SIMULATE A MOBILE ADHOC NETWORK
DATE:	

AIM:

To simulate a Mobile Adhoc network (MANET) using NS2.

SOFTWARE REQUIREMENTS:

Network Simulator -2

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:

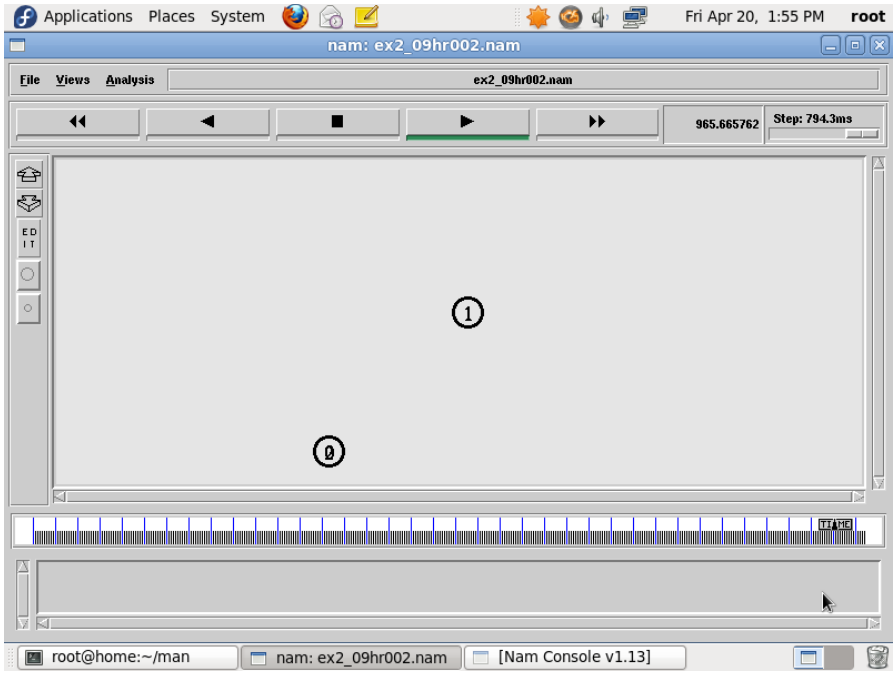
1. Create a simulator object
2. Set the values for the parameter
3. Open a nam trace file and define finish procedure then close the trace file, and execute nam on trace file.
4. Create the nodes that forms a network numbered from 0 to 3
5. Schedule events and run the program.

PROGRAM:

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

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

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

set topo [new Topography]
$topo load_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
exec nam output.nam &
exit 0
}

$ns run
```

OUTPUT:



VIVA QUESTIONS:

1. What is the meaning of the word Adhoc?
2. Mention some applications of Adhoc Networks?
3. Mention some of the routing protocols for Adhoc Network.
4. What are the common issues in Adhoc network?
5. Is Multi hop communication possible in Adhoc network?

RESULT:

Thus the mobile adhoc network is simulated successfully using network simulator2.

EXPT.NO: 15 DATE:	IMPLEMENT TRANSPORT CONTROL PROTOCOL IN SENSOR NETWORK

AIM:

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

SOFTWARE REQUIREMENTS:

Network Simulator -2

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:

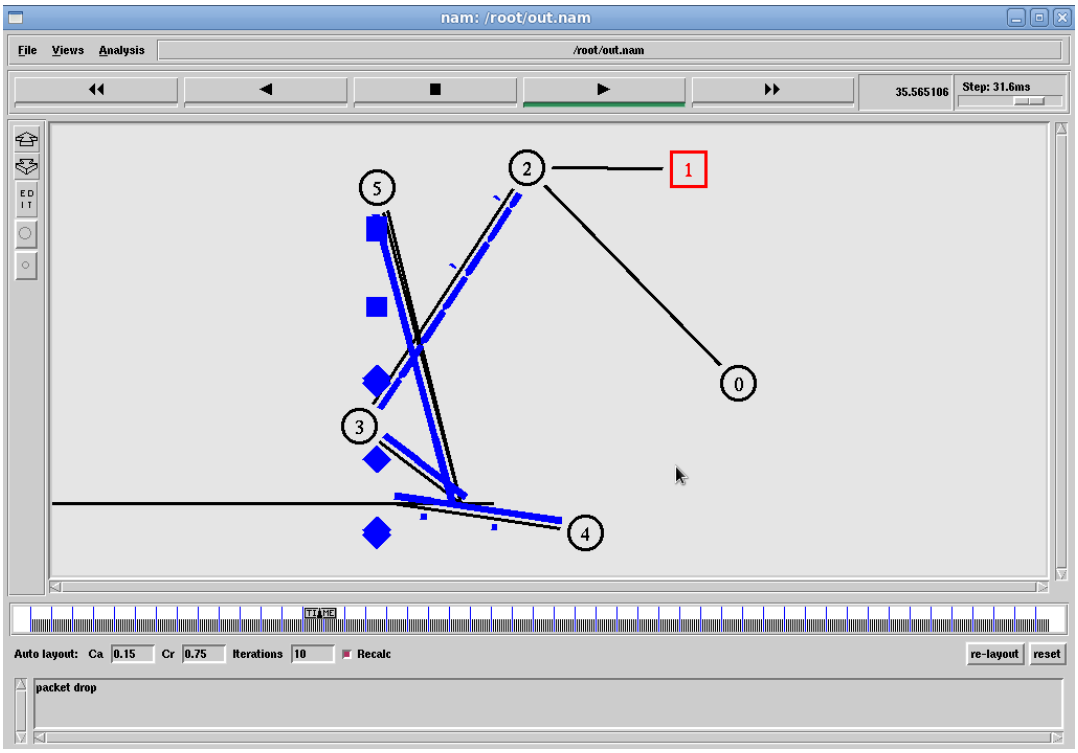
1. Create a simulator object
2. Define different colors for different data flows
3. Open a nam trace file and define finish procedure then close the trace file, and execute nam on trace file.
4. Create six nodes that forms a network numbered from 0 to 5
5. Create duplex links between the nodes and add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


$ns at 1.0 "$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.
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:



VIVA QUESTIONS:

1. What is TCP?
2. Explain the three way Handshake process?
3. Which layer is closer to a user?
4. Explain how TCP avoids a network meltdown?
5. What is the difference between flow control and Error control?

RESULT:

Thus the Transport control protocol in sensor network is simulated successfully using NS2.