EX.NO: 7A **STUDY OF NETWORK SIMULATOR USING 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.

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).

BASIC ARCHITECTURE:

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

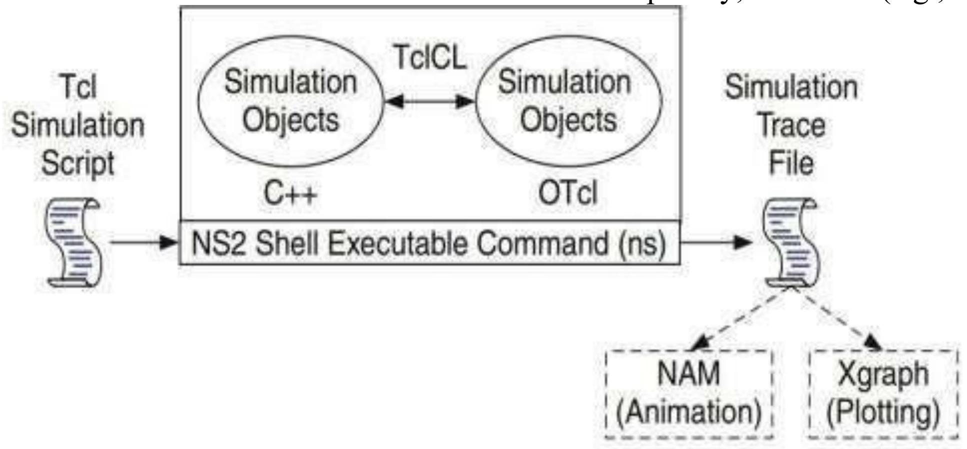


Fig. 2.1. Basic architecture of NS.

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.

1. 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.

2. Basics of TCL

Syntax: command arg1 arg2 arg3

3. 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]

4. 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

5. 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.
6. Initialization and Termination of TCL Script in NS-2

An ns simulation starts with the command

7. 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 $tracefile
#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

8. 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:

9. $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:

10. 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**

11. $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.

RESULT:

Thus the network simulator 2 is studied in detail.

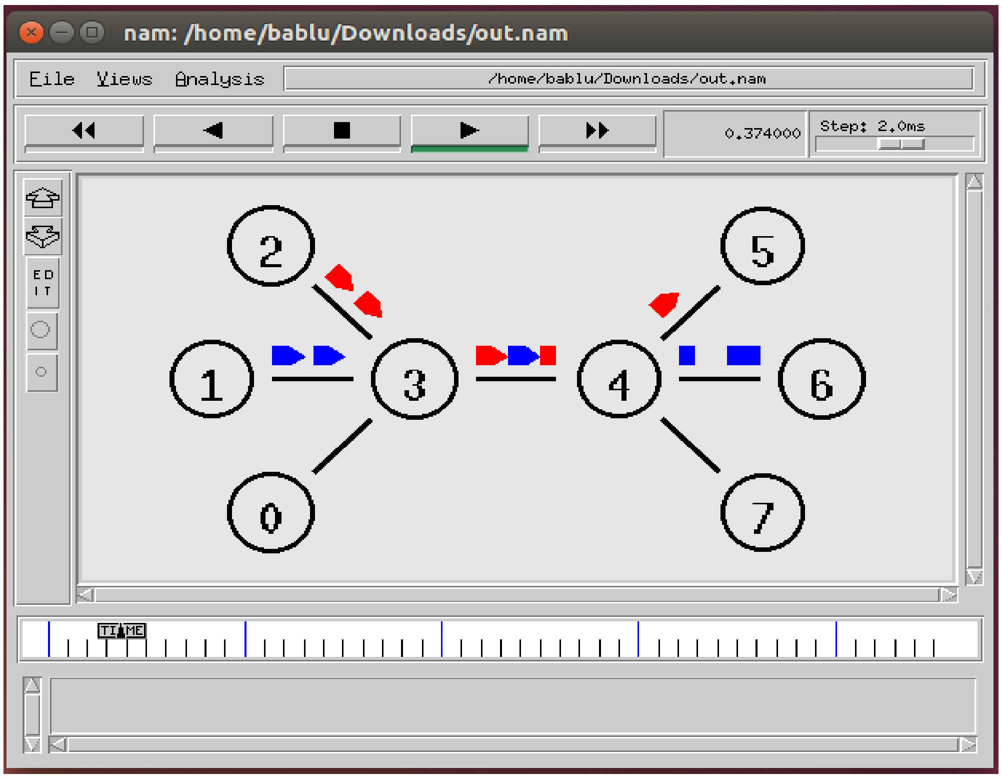
PROGRAM7B:

```
#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 $nf
        #Execute nam on the trace file
        exec nam out.nam &
        exit 0
}
#Create eight nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
set n7 [$ns node]

#Create links between the nodes
$ns duplex-link $n0 $n3 1Mb 10ms RED
$ns duplex-link $n1 $n3 1Mb 10ms RED
$ns duplex-link $n2 $n3 1Mb 10ms RED
$ns duplex-link $n3 $n4 1Mb 10ms RED
$ns duplex-link $n4 $n5 1Mb 10ms RED
$ns duplex-link $n4 $n6 1Mb 10ms RED
$ns duplex-link $n4 $n7 1Mb 10ms RED
$ns duplex-link-op $n0 $n3 orient right-up
$ns duplex-link-op $n3 $n4 orient middle
$ns duplex-link-op $n2 $n3 orient right-down
$ns duplex-link-op $n4 $n5 orient right-up
$ns duplex-link-op $n4 $n7 orient right-down
$ns duplex-link-op $n1 $n3 orient right
$ns duplex-link-op $n6 $n4 orient left
Create a UDP agent and attach it to node n2
set udp0 [new Agent/UDP]
$ns attach-agent $n2 $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
```

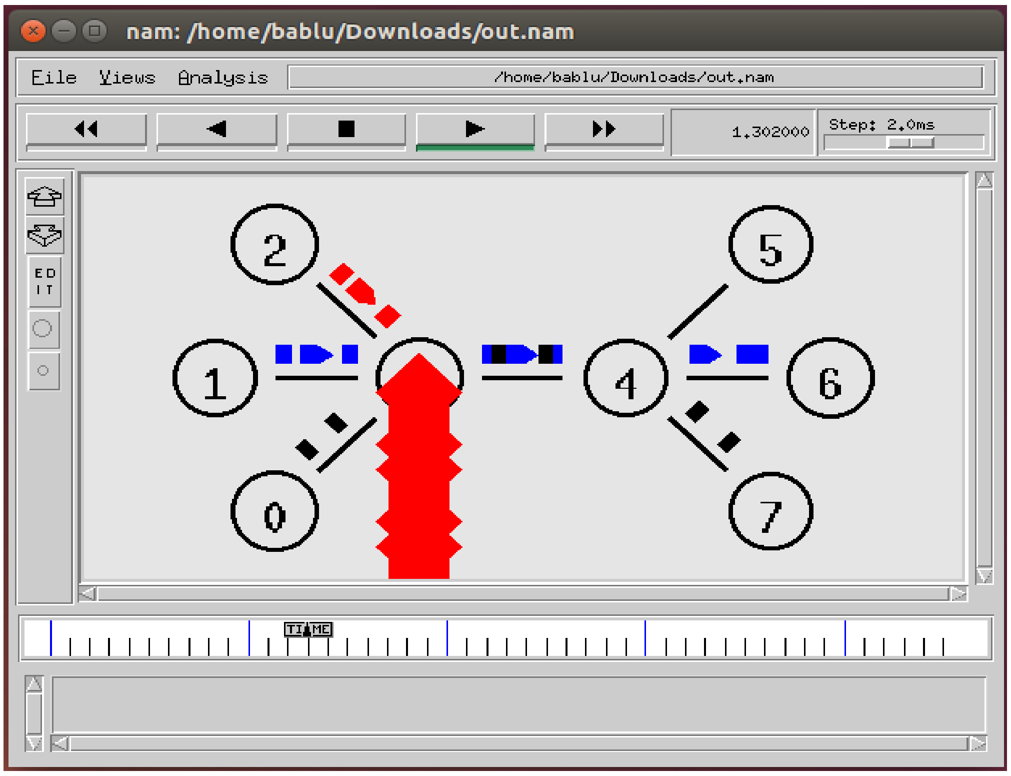
```
$cbr0 attach-agent $udp0



#Create a Null agent (a traffic sink) and attach it to node n5
set null0 [new Agent/Null]
$ns attach-agent $n5 $null0

#Connect the traffic sources with the traffic sink
$ns connect $udp0 $null0
#Create a UDP agent and attach it to node n1
set udp1 [new Agent/UDP]
$ns attach-agent $n1 $udp1

#Create a CBR traffic source and attach it to udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp1

#Create a Null agent (a traffic sink) and attach it to node n6
set null0 [new Agent/Null]
$ns attach-agent $n6 $null0

#Connect the traffic sources with the traffic sink
$ns connect $udp1 $null0

#Create a UDP agent and attach it to node n0
set udp2 [new Agent/UDP]
$ns attach-agent $n0 $udp2

#Create a CBR traffic source and attach it to udp2
set cbr2 [new Application/Traffic/CBR]
$cbr2 set packet size_ 500
$cbr2 set interval_ 0.005
$cbr2 attach-agent $udp2

#Create a Null agent (a traffic sink) and attach it to node n7
set null0 [new Agent/Null]
$ns attach-agent $n7 $null0

#Connect the traffic sources with the traffic sink
$ns connect $udp2 $null0

$udp0 set fid_ 1

$udp1 set fid_ 2

$udp2 set fid_ 3

#Define different colors for data flows
```

```
$ns color 1 Red
$ns color 2 Green
$ns color 2 Blue




#Schedule events for the CBR agents
$ns at 0.1 "$cbr0 start"
$ns at 0.2 "$cbr1 start"
$ns at 0.5 "$cbr2 start"
$ns at 4.0 "$cbr2 stop"
$ns at 4.2 "$cbr1 stop"
$ns at 4.5 "$cbr0 stop"

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

#Run the simulation
$ns run
```

OUTPUT:


nam: /home/bablu/Downloads/out.nam
File
Views
Analysis
/home/bablu/Downloads/out.nam
0.374000
Step: 2.0ms
EDIT
TIME
2
5
1
3
4
6
0
7

2117230020153


nam: /home/bablu/Downloads/out.nam
File Views Analysis
/home/bablu/Downloads/out.nam
1.302000
Step: 2.0ms
EDIT
2
5
1
4
6
0
7
TIME

Introduction :

The transmission Control Protocol (TCP) is one of the most important protocols of Internet Protocols suite. It is most widely used protocol for data transmission in communication network such as internet.

Features

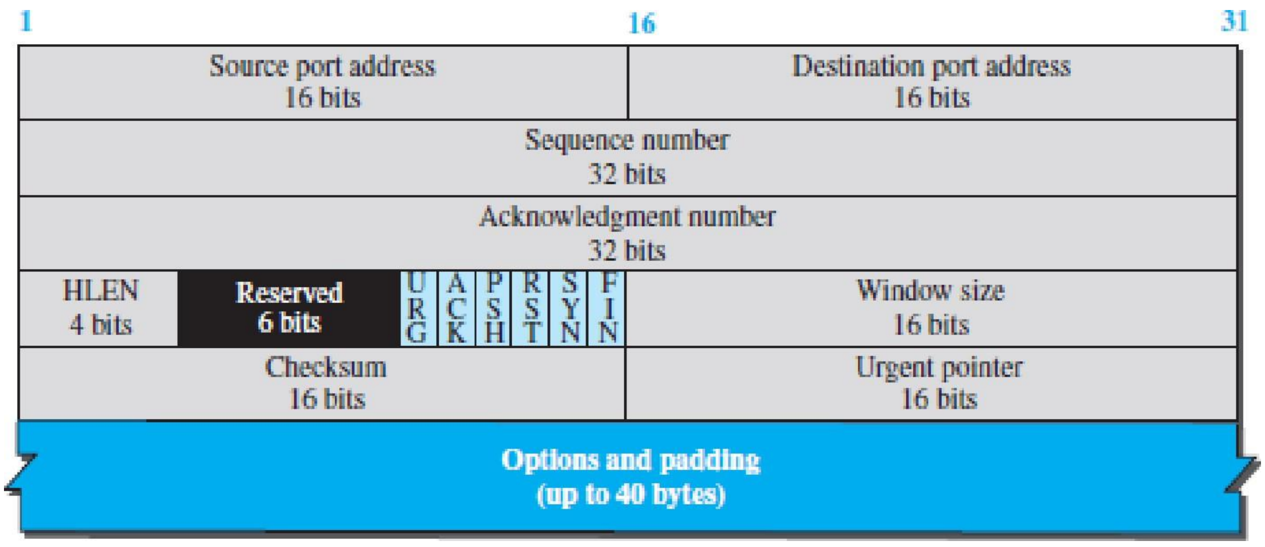
- TCP is reliable protocol. That is, the receiver always sends either positive or negative acknowledgement about the data packet to the sender, so that the sender always has bright clue about whether the data packet is reached the destination or it needs to resend it.
- TCP ensures that the data reaches intended destination in the same order it was sent.
- TCP is connection oriented. TCP requires that connection between two remote points be established before sending actual data.
- TCP provides error-checking and recovery mechanism.
- TCP provides end-to-end communication.
- TCP provides flow control and quality of service.
- TCP operates in Client/Server point-to-point mode.
- TCP provides full duplex server, i.e. it can perform roles of both receiver and sender.

Header

The length of TCP header is minimum 20 bytes long and maximum 60 bytes.

Source port address. This is a 16-bit field that defines the port number of the application program in the host that is sending the segment.

Destination port address. This is a 16-bit field that defines the port number of the application program in the host that is receiving the segment.



Source port address. This is a 16-bit field that defines the port number of the application program in the host that is sending the segment.

Destination port address. This is a 16-bit field that defines the port number of the application program in the host that is receiving the segment.

Sequence number. This 32-bit field defines the number assigned to the first byte of data contained in this segment. TCP is a stream transport protocol. To ensure connectivity, each byte to be transmitted is numbered. The sequence number tells the destination which byte in this sequence is the first byte in the segment. During connection establishment each party uses a random number generator to create an **initial sequence number** (ISN), which is usually different in each direction.

Acknowledgment number. This 32-bit field defines the byte number that the receiver of the segment is expecting to receive from the other party. If the receiver of the segment has successfully received byte number *x* from the other party, it returns *x* + 1 as the acknowledgment number. Acknowledgment and data can be piggybacked together.

Header length. This 4-bit field indicates the number of 4-byte words in the TCP header. The length of the header can be between 20 and 60 bytes. Therefore, the value of this field is always between 5 ($5 \times 4 = 20$) and 15 ($15 \times 4 = 60$).

Control. This field defines 6 different control bits or flags. One or more of these bits can be set at a time. These bits enable flow control, connection establishment and termination, connection abortion, and the mode of data transfer in TCP.

Window size. This field defines the window size of the sending TCP in bytes. Note that the length of this field is 16 bits, which means that the maximum size of the window is 65,535 bytes. This value is normally referred to as the receiving window (*rwnd*) and is determined by the receiver. The sender must obey the dictation of the receiver in this case.

Checksum. This 16-bit field contains the checksum. The calculation of the checksum for TCP follows the same procedure as the one described for UDP. However, the use of the checksum in the UDP datagram is optional, whereas

the use of the checksum for TCP is mandatory. The same pseudoheader, serving the same purpose, is added to the segment. For the TCP pseudoheader, the value for the protocol field is 6.

Urgent pointer. This 16-bit field, which is valid only if the urgent flag is set, is used when the segment contains urgent data. It defines a value that must be added to the sequence number to obtain the number of the last urgent byte in the data section of the segment.

Options. There can be up to 40 bytes of optional information in the TCP header.

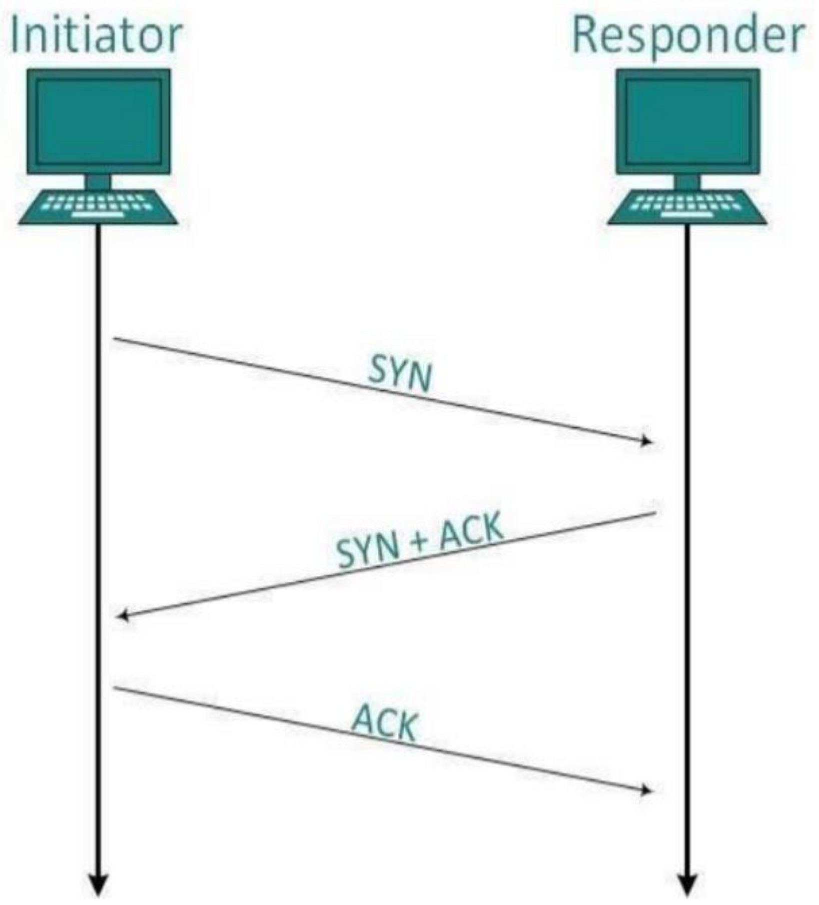
Addressing

TCP communication between two remote hosts is done by means of port numbers (TSAPs). Ports numbers can range from 0 – 65535 which are divided as:

- System Ports (0 – 1023)
- User Ports (1024 – 49151)
- Private/Dynamic Ports (49152 – 65535)

Connection Management

TCP communication works in Server/Client model. The client initiates the connection and the server either accepts or rejects it. Three-way handshaking is used for connection management.



Establishment

Client initiates the connection and sends the segment with a Sequence number. Server acknowledges it back with its own Sequence number and ACK of client’s segment which is one more than client’s Sequence number. Client after receiving ACK of its segment sends an acknowledgement of Server’s response.

Release

Either of server and client can send TCP segment with FIN flag set to 1. When the receiving end responds it back by Acknowledging FIN, that direction of TCP communication is closed and connection is released.

Bandwidth Management

TCP uses the concept of window size to accommodate the need of Bandwidth management. Window size tells the sender at the remote end, the number of data byte segments the receiver at this end can receive. TCP uses slow start phase by using window size 1 and increases the window size exponentially after each successful communication.

For example, the client uses windows size 2 and sends 2 bytes of data. When the acknowledgement of this segment received the windows size is doubled to 4 and next sent the segment sent will be 4 data bytes long. When the acknowledgement of 4-byte data segment is received, the client sets windows size to 8 and so on.

If an acknowledgement is missed, i.e. data lost in transit network or it received NACK, then the window size is reduced to half and slow start phase starts again.

Error Control &and Flow Control

TCP uses port numbers to know what application process it needs to handover the data segment. Along with that, it uses sequence numbers to synchronize itself with the remote host. All data segments are sent and received with sequence numbers. The Sender knows which last data segment was received by the Receiver when it gets ACK. The Receiver knows about the last segment sent by the Sender by referring to the sequence number of recently received packet.

If the sequence number of a segment recently received does not match with the sequence number the receiver was expecting, then it is discarded and NACK is sent back. If two segments arrive with the same sequence number, the TCP timestamp value is compared to make a decision.

Multiplexing

The technique to combine two or more data streams in one session is called Multiplexing. When a TCP client initializes a connection with Server, it always refers to a welldefined port number which indicates the application process. The client itself uses a randomly generated port number from private port number pools.

Using TCP Multiplexing, a client can communicate with a number of different application process in a single session. For example, a client requests a web page which in turn contains different types of data (HTTP, SMTP, FTP etc.) the TCP session timeout is increased and the session is kept open for longer time so that the three-way handshake overhead can be avoided.

This enables the client system to receive multiple connection over single virtual connection. These virtual connections are not good for Servers if the timeout is too long.

Congestion Control

When large amount of data is fed to system which is not capable of handling it, congestion occurs. TCP controls congestion by means of Window mechanism. TCP sets a window size telling the other end how much data segment to send. TCP may use three algorithms for congestion control:

· Additive increase, Multiplicative Decrease
· Slow Start
· Timeout React

Timer Management

TCP uses different types of timer to control and management various tasks:

Keep-alive timer:
· This timer is used to check the integrity and validity of a connection.
· When keep-alive time expires, the host sends a probe to check if the connection still exists.

Retransmission timer:
· This timer maintains stateful session of data sent.
· If the acknowledgement of sent data does not receive within the Retransmission time, the data segment is sent again.

Persist timer:
· TCP session can be paused by either host by sending Window Size 0.
· To resume the session a host needs to send Window Size with some larger value. · If this segment never reaches the other end, both ends may wait for each other for infinite time.
· When the Persist timer expires, the host re-sends its window size to let the other end know.
· Persist Timer helps avoid deadlocks in communication.

Timed-Wait:

· After releasing a connection, either of the hosts waits for a Timed-Wait time to terminate the connection completely.

· This is in order to make sure that the other end has received the acknowledgement of its connection termination request.

· Timed-out can be a maximum of 240 seconds (4 minutes).

Crash Recovery

TCP is very reliable protocol. It provides sequence number to each of byte sent in segment. It provides the feedback mechanism i.e. when a host receives a packet, it is bound to ACK that packet having the next sequence number expected (if it is not the last segment).

When a TCP Server crashes mid-way communication and re-starts its process it sends TPDU broadcast to all its hosts. The hosts can then send the last data segment which was never unacknowledged and carry onwards.

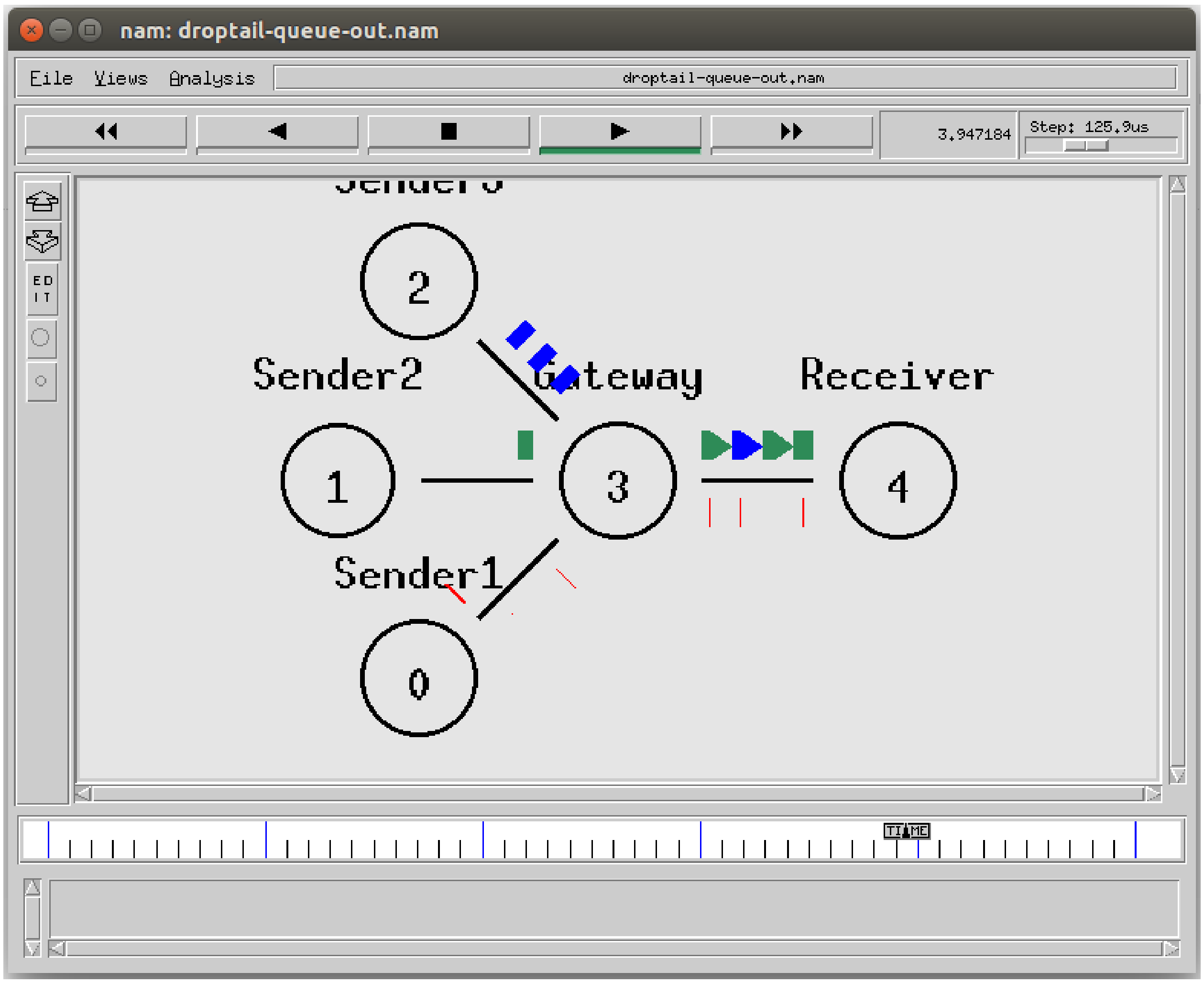
PROGRAM8A: File name - TCP.tcl

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

#Open trace files
set f [open droptail-queue-out.tr w]
$ns trace-all $f

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

#s1, s2 and s3 act as sources.
set s1 [$ns node]
set s2 [$ns node]
set s3 [$ns node]

#G acts as a gateway
set G [$ns node]

#r acts as a receiver
set r [$ns node]

#Define different colors for data flows
$ns color 1 red
$ns color 2 SeaGreen
$ns color 3 blue

#Create links between the nodes
$ns duplex-link $s1 $G 6Mb 10ms DropTail
$ns duplex-link $s2 $G 6Mb 10ms DropTail
$ns duplex-link $s3 $G 6Mb 10ms DropTail
$ns duplex-link $G $r 3Mb 10ms DropTail
```

```
#Define the layout of the nodes
$ns duplex-link-op $s1 $G orient right-up
$ns duplex-link-op $s2 $G orient right
$ns duplex-link-op $s3 $G orient right-down
$ns duplex-link-op $G $r orient right

#Define the queue size for the link between node G and r
$ns queue-limit $G $r 5

#Monitor the queues for links vertically
$ns duplex-link-op $s1 $G queuePos 0.5
$ns duplex-link-op $s2 $G queuePos 0.5
$ns duplex-link-op $s3 $G queuePos 0.5
$ns duplex-link-op $G $r queuePos 0.5

#Create a TCP agent and attach it to node s1
set tcp1 [new Agent/TCP/Reno]
$ns attach-agent $s1 $tcp1
$tcp1 set window_ 8
$tcp1 set fid_ 1

#Create a TCP agent and attach it to node s2
set tcp2 [new Agent/TCP/Reno]
$ns attach-agent $s2 $tcp2
$tcp2 set window_ 8
$tcp2 set fid_ 2

#Create a TCP agent and attach it to node s3
set tcp3 [new Agent/TCP/Reno]
$ns attach-agent $s3 $tcp3
$tcp3 set window_ 4
$tcp3 set fid_ 3

#Create TCP sink agents and attach them to node r
set sink1 [new Agent/TCPSink]
set sink2 [new Agent/TCPSink]
set sink3 [new Agent/TCPSink]
$ns attach-agent $r $sink1
$ns attach-agent $r $sink2
$ns attach-agent $r $sink3

#Connect the traffic sources with the traffic sinks
$ns connect $tcp1 $sink1
$ns connect $tcp2 $sink2
$ns connect $tcp3 $sink3


Create FTP applications and attach them to agents
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2
set ftp3 [new Application/FTP]
```

```
$ftp3 attach-agent $tcp3


#Define a 'finish' procedure
proc finish {} {
 global ns
 $ns flush-trace
 puts "running nam..."
 exec nam -a droptail-queue-out.nam &
 exit 0
}

#Define label for nodes
$ns at 0.0 "$s1 label Sender1"
$ns at 0.0 "$s2 label Sender2"
$ns at 0.0 "$s3 label Sender3"
$ns at 0.0 "$G label Gateway"
$ns at 0.0 "$r label Receiver"

#Schedule ftp events
$ns at 0.1 "$ftp1 start"
$ns at 0.1 "$ftp2 start"
$ns at 0.1 "$ftp3 start"
$ns at 5.0 "$ftp1 stop"
$ns at 5.0 "$ftp2 stop"
$ns at 5.0 "$ftp3 stop"

#Call finish procedure after 5 seconds of simulation time
$ns at 5.25 "finish"

#Run the simulation
$ns run
```

OUTPUT:


nam: droptail-queue-out.nam
File Views Analysis
droptail-queue-out.nam
3.947184
Step: 125.9us
Sender0
2
Sender2
Gateway
Receiver
1
3
4
Sender1
0
TIME

Introduction :

The User Datagram Protocol (UDP) is simplest Transport Layer communication protocol available of the TCP/IP protocol suite. It involves minimum amount of communication mechanism. UDP is said to be an unreliable transport protocol but it uses IP services which provides best effort delivery mechanism.

In UDP, the receiver does not generate an acknowledgement of packet received and in turn, the sender does not wait for any acknowledgement of packet sent. This shortcoming makes this protocol unreliable as well as easier on processing.

Requirement of UDP

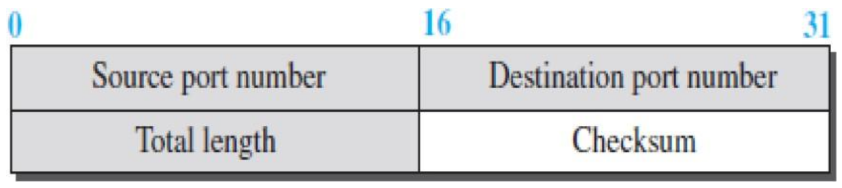
A question may arise, why do we need an unreliable protocol to transport the data? We deploy UDP where the acknowledgement packets share significant amount of bandwidth along with the actual data. For example, in case of video streaming, thousands of packets are forwarded towards its users. Acknowledging all the packets is troublesome and may contain huge amount of bandwidth wastage. The best delivery mechanism of underlying IP protocol ensures best efforts to deliver its packets, but even if some packets in video streaming get lost, the impact is not calamitous and can be ignored easily. Loss of few packets in video and voice traffic sometimes goes unnoticed.

Features

· UDP is used when acknowledgement of data does not hold any significance.
· UDP is good protocol for data flowing in one direction.
· UDP is simple and suitable for query based communications.
· UDP is not connection oriented.
· UDP does not provide congestion control mechanism.
· UDP does not guarantee ordered delivery of data.
· UDP is stateless.
· UDP is suitable protocol for streaming applications such as VoIP, multimedia streaming.

UDP Header

UDP header is as simple as its function.



UDP header contains four main parameters:
· **Source Port** - This 16 bits information is used to identify the source port of the packet.
· **Destination Port** - This 16 bits information, is used identify application level service on destination machine.
· **Length -** Length field specifies the entire length of UDP packet (including header). It is 16-bits field and minimum value is 8-byte, i.e. the size of UDP header itself.
· **Checksum** - This field stores the checksum value generated by the sender before sending. IPv4 has this field as optional so when checksum field does not contain any value it is made 0 and all its bits are set to zero.

UDP application

Here are few applications where UDP is used to transmit data:

· main Name Services
· Simple Network Management Protocol
· Trivial File Transfer Protocol

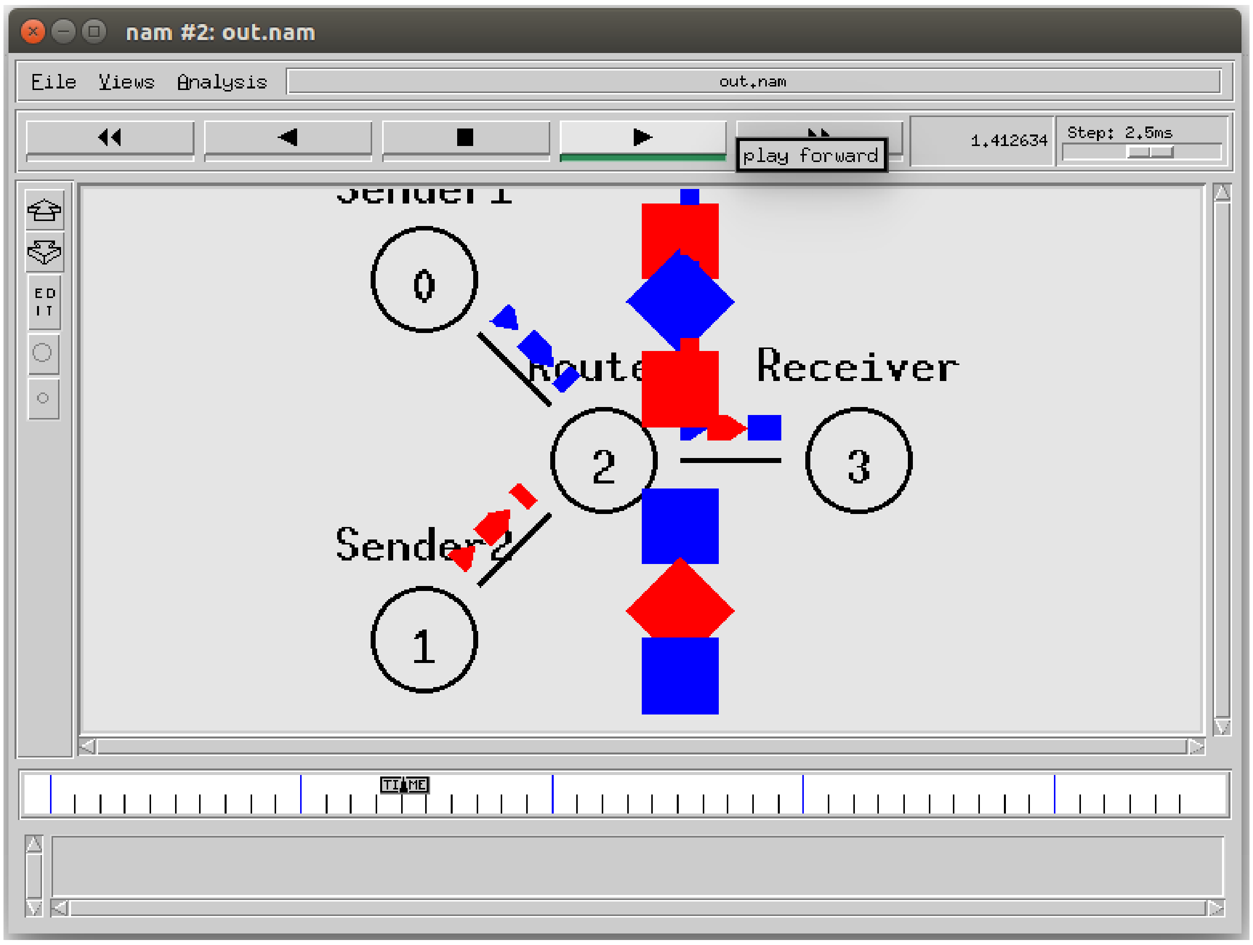
· Routing Information Protocol
· Kerberos

PROGRAM 8B: File name - UDP.tcl

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

#Define different colors for data flows
$ns color 1 Blue
$ns color 2 Red

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

#Create four nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

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

#Specify layout of nodes
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right

#Monitor the queue for the link 2ņ3 vertically
$ns duplex-link-op $n2 $n3 queuePos 0.5

#Create a UDP agent and attach it to node n0
set udp0 [new Agent/UDP]
$udp0 set class_ 1
$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 UDP agent and attach it to node n1
set udp1 [new Agent/UDP]
$udp1 set class_ 2
$ns attach-agent $n1 $udp1
```

```
# Create a CBR traffic source and attach it to udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp1

#Create a Null agent (a traffic sink) and attach it to node n3
set null0 [new Agent/Null]
$ns attach-agent $n3 $null0

#Connect traffic sources with the traffic sink
$ns connect $udp0 $null0
$ns connect $udp1 $null0

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

#Define label for nodes
$ns at 0.0 "$n0 label Sender1"
$ns at 0.0 "$n1 label Sender2"
$ns at 0.0 "$n2 label Router"
$ns at 0.0 "$n3 label Receiver"

#Schedule events for the CBR agents
$ns at 0.5 "$cbr0 start"
$ns at 1.0 "$cbr1 start"
$ns at 4.0 "$cbr1 stop"
$ns at 4.5 "$cbr0 stop"

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

#Run the simulation
$ns run
```

OUTPUT:

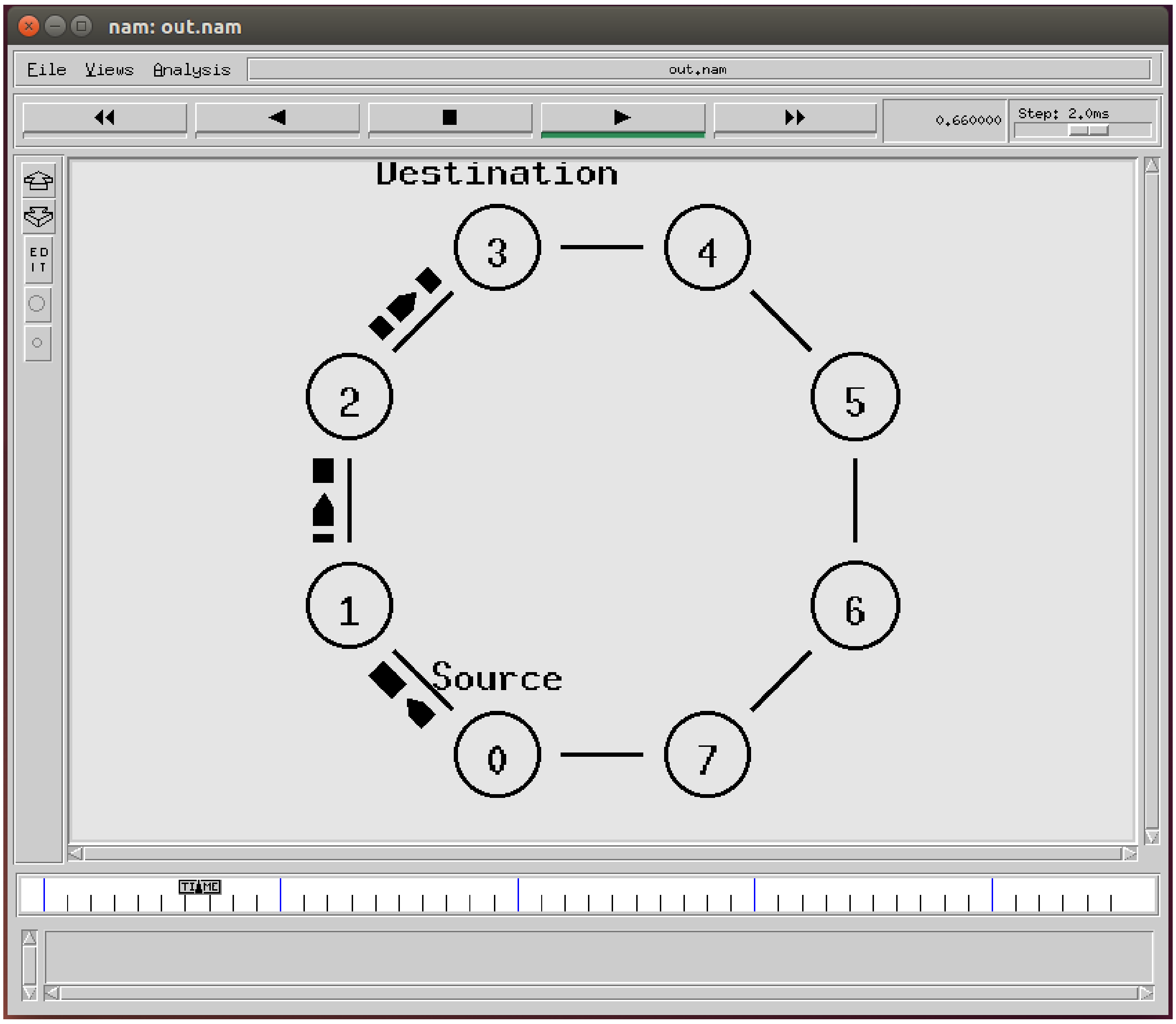

nam #2: out.nam
File
Views
Analysis
out.nam
play forward
1.412634
Step: 2.5ms
Sender 1
0
Router
Receiver
2
3
Sender 2
1
TIME

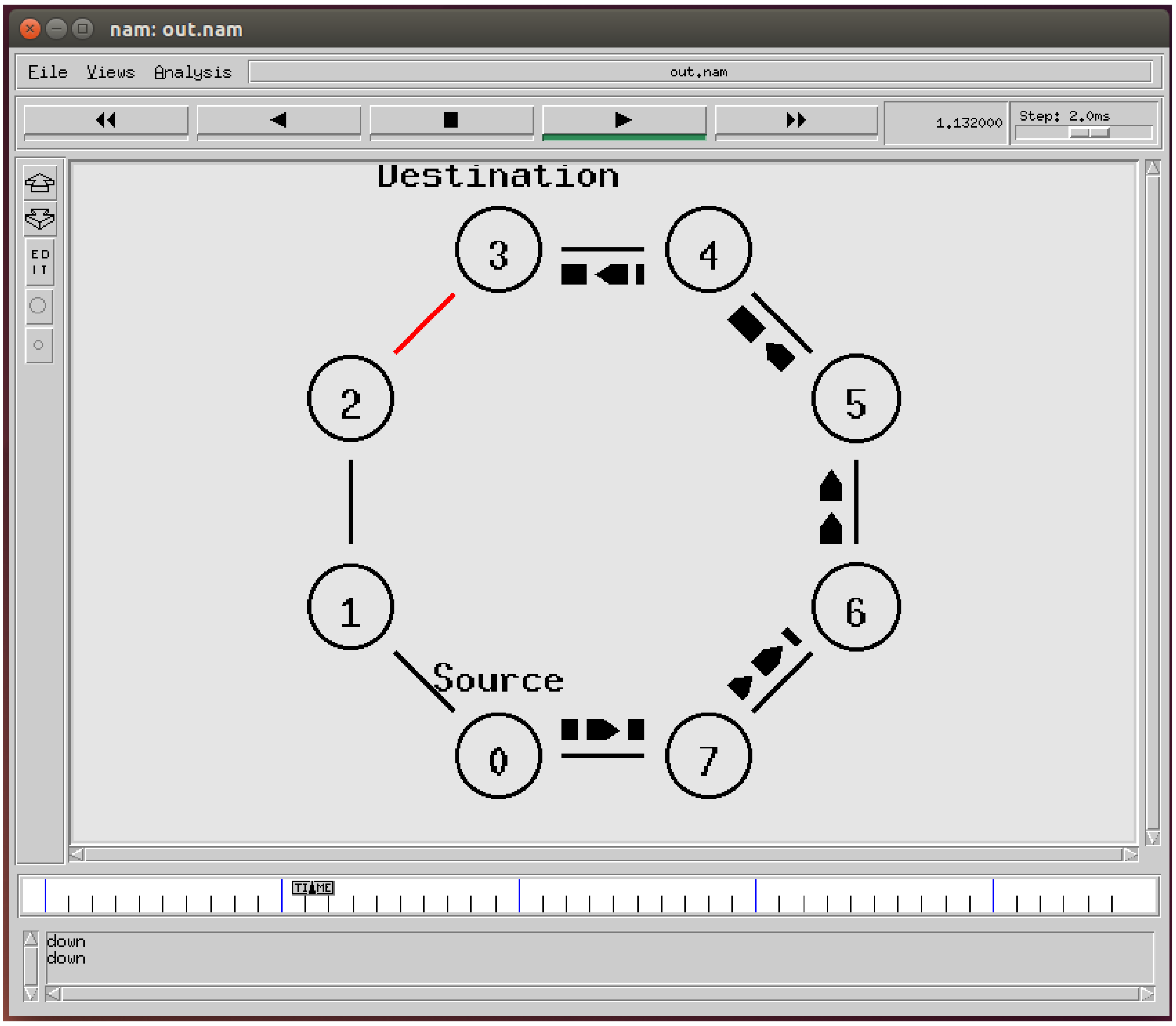
PROGRAM9A:

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

#Use distance vector routing
$ns rtproto DV

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

# Open tracefile
set nt [open trace.tr w]
$ns trace-all $nt

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

# Create 8 nodes
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
set n7 [$ns node]
set n8 [$ns node]

# Specify link characterestics
$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 $n6 1Mb 10ms DropTail
$ns duplex-link $n6 $n7 1Mb 10ms DropTail
$ns duplex-link $n7 $n8 1Mb 10ms DropTail
$ns duplex-link $n8 $n1 1Mb 10ms DropTail

# specify layout as a octagon
$ns duplex-link-op $n1 $n2 orient left-up
$ns duplex-link-op $n2 $n3 orient up
$ns duplex-link-op $n3 $n4 orient right-up
$ns duplex-link-op $n4 $n5 orient right
```

```
$ns duplex-link-op $n5 $n6 orient right-down
$ns duplex-link-op $n6 $n7 orient down
$ns duplex-link-op $n7 $n8 orient left-down
$ns duplex-link-op $n8 $n1 orient left

#Create a UDP agent and attach it to node n1
set udp0 [new Agent/UDP]
$ns attach-agent $n1 $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 n4
set null0 [new Agent/Null]
$ns attach-agent $n4 $null0

#Connect the traffic source with the traffic sink
$ns connect $udp0 $null0

#Schedule events for the CBR agent and the network dynamics
$ns at 0.0 "$n1 label Source"
$ns at 0.0 "$n4 label Destination"
$ns at 0.5 "$cbr0 start"
$ns rtmodel-at 1.0 down $n3 $n4
$ns rtmodel-at 2.0 up $n3 $n4
$ns at 4.5 "$cbr0 stop"

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

#Run the simulation
$ns run
```

OUTPUT:


nam: out.nam
File Views Analysis
out.nam
0.660000
Step: 2.0ms
Destination
Source
0
1
2
3
4
5
6
7
TIME


nam: out.nam
File Views Analysis
out.nam
1.132000
Step: 2.0ms
Destination
3
4
2
5
1
6
Source
0
7
TIME

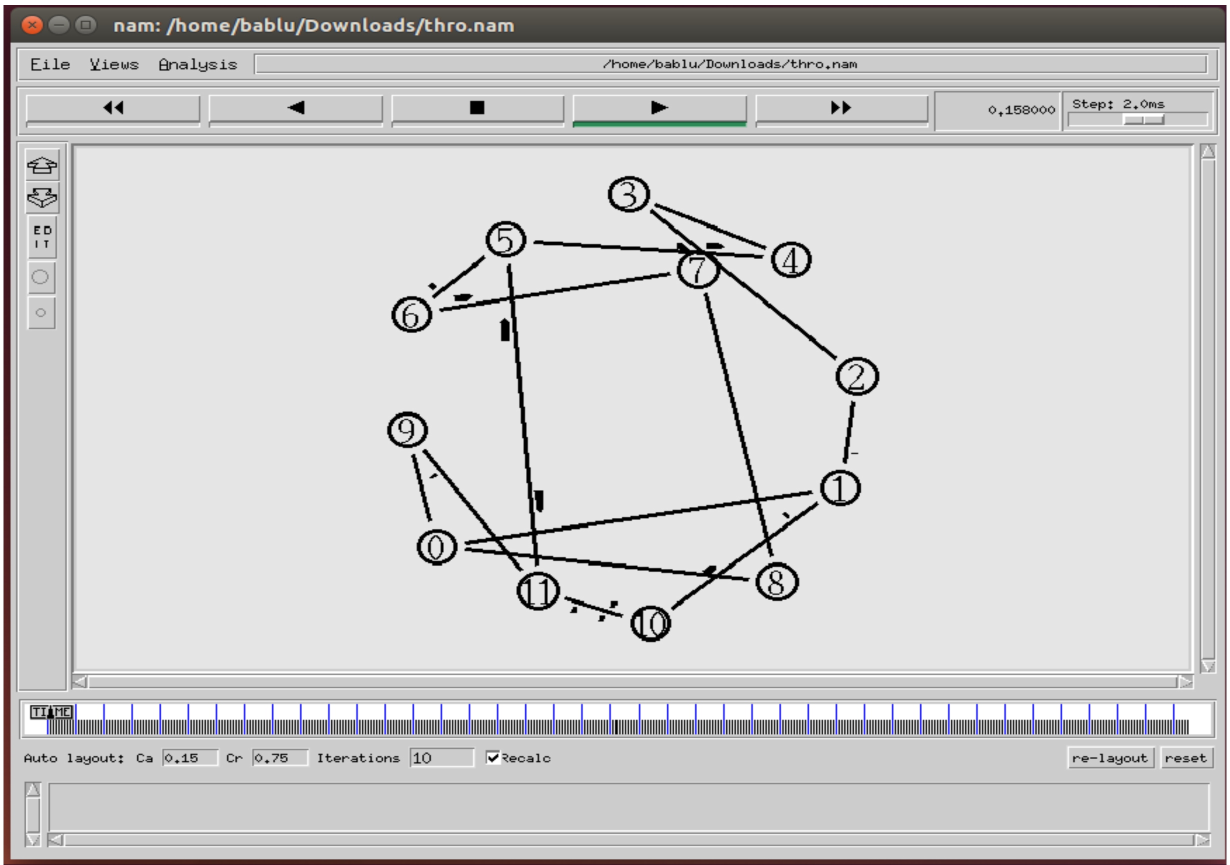
```
down
down
```

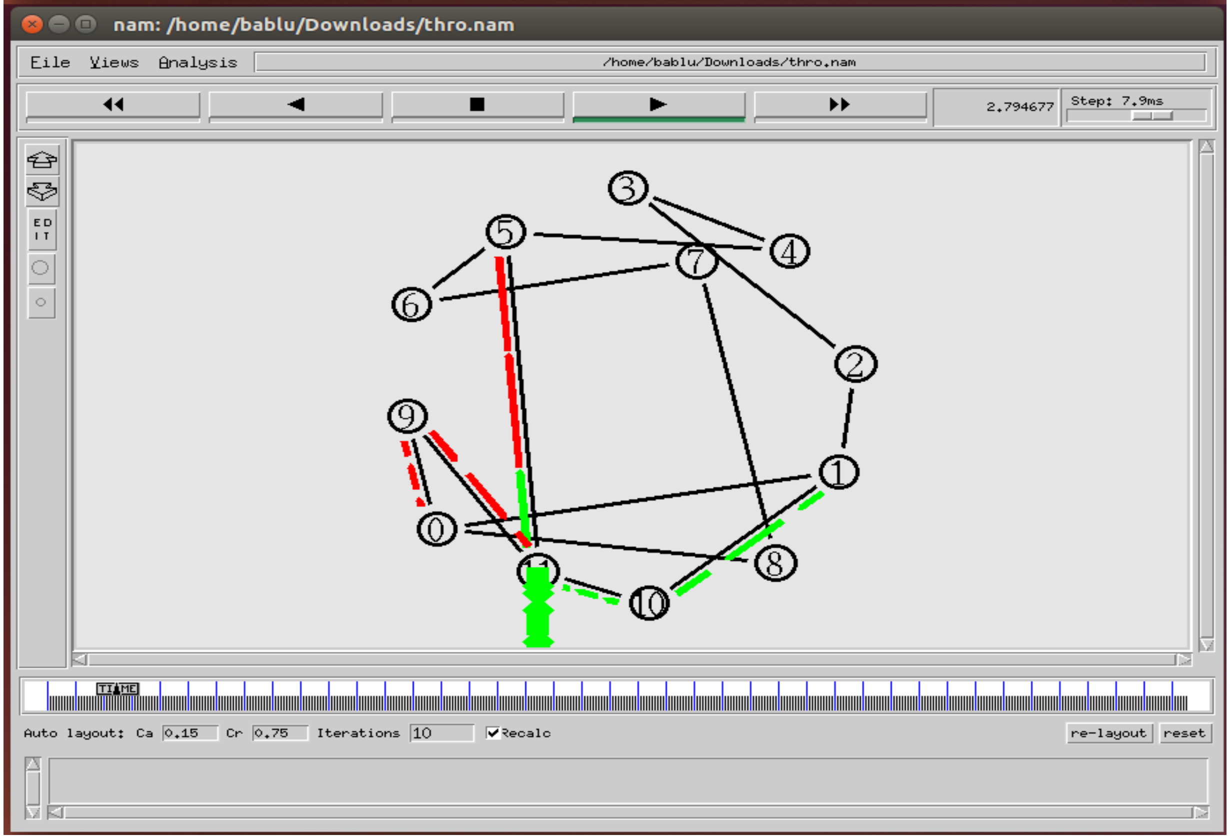
PROGRAM9B:

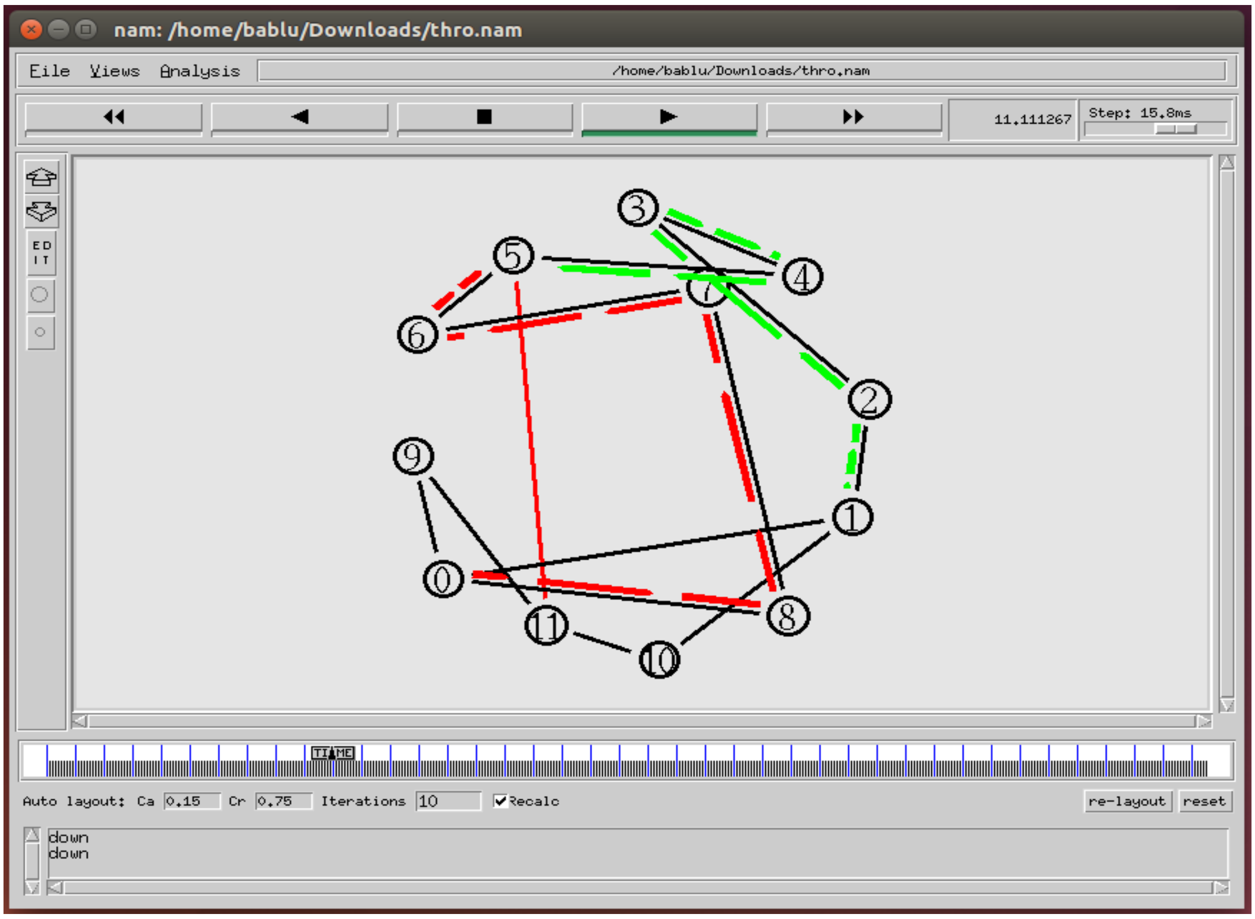
```
set ns [new Simulator]
set nr [open thro.tr w]
$ns trace-all $nr
set nf [open thro.nam w]

$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:


nam: /home/bablu/Downloads/thro.nam
File Views Analysis
/home/bablu/Downloads/thro.nam
0.158000
Step: 2.0ms
3
5
4
7
6
2
9
1
0
11
8
10
TIME
Auto layout: Ca 0.15 Cr 0.75 Iterations 10 Recalc
re-layout
reset


nam: /home/bablu/Downloads/thro.nam
File Views Analysis
/home/bablu/Downloads/thro.nam
2.794677
Step: 7.9ms
3
5
4
7
6
2
9
1
0
8
10
TIME
Auto layout: Ca 0.15 Cr 0.75 Iterations 10 Recalc
re-layout
reset

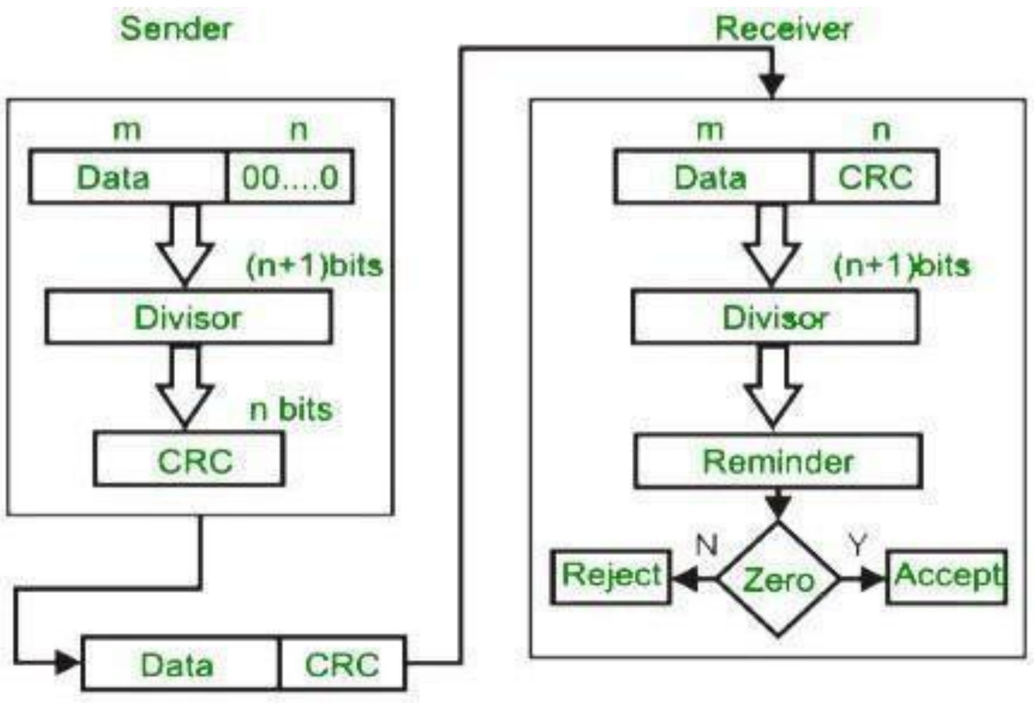


Auto layout: Ca 0.15 Cr 0.75 Iterations 10 ✓Recalc

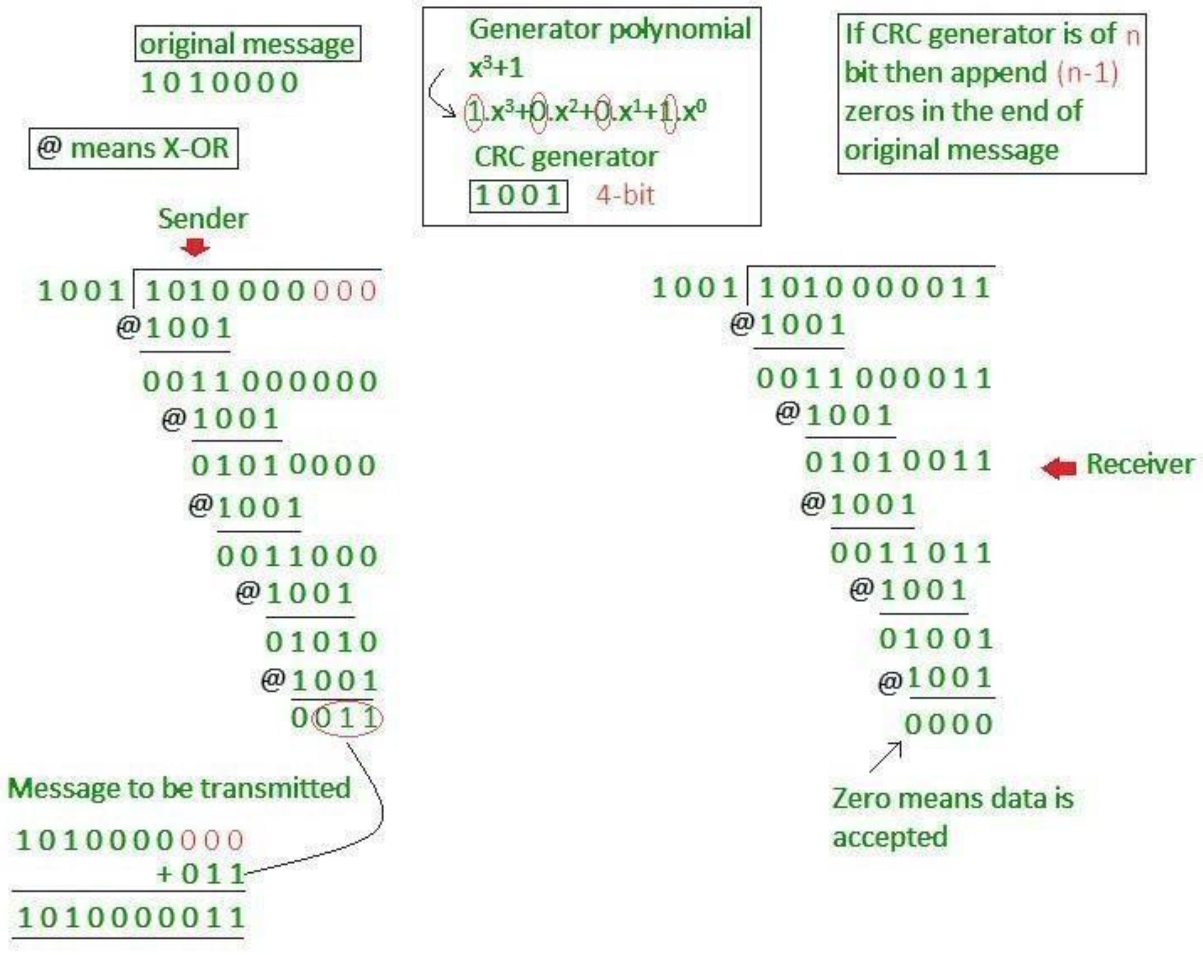
```
down
down
```

Cyclic redundancy check (CRC)

- Unlike checksum scheme, which is based on addition, CRC is based on binary division.
- In CRC, a sequence of redundant bits, called cyclic redundancy check bits, are appended to the end of data unit so that the resulting data unit becomes exactly divisible by a second, predetermined binary number.
- At the destination, the incoming data unit is divided by the same number. If at this step there is no remainder, the data unit is assumed to be correct and is therefore accepted.
- A remainder indicates that the data unit has been damaged in transit and therefore must be rejected.



EXAMPLE:


original message
1010000
@ means X-OR
Generator polynomial
x³+1
1.x³+0.x²+0.x¹+1.x⁰
CRC generator
1001 4-bit
If CRC generator is of n bit then append (n-1) zeros in the end of original message
Sender
1001 | 1010000000
@1001
0011000000
@1001
01010000
@1001
0011000
@1001
01010
@1001
0011
Message to be transmitted
1010000000
+011
1010000011
1001 | 1010000011
@1001
0011000011
@1001
01010011
@1001
0011011
@1001
01001
@1001
0000
Receiver
Zero means data is accepted

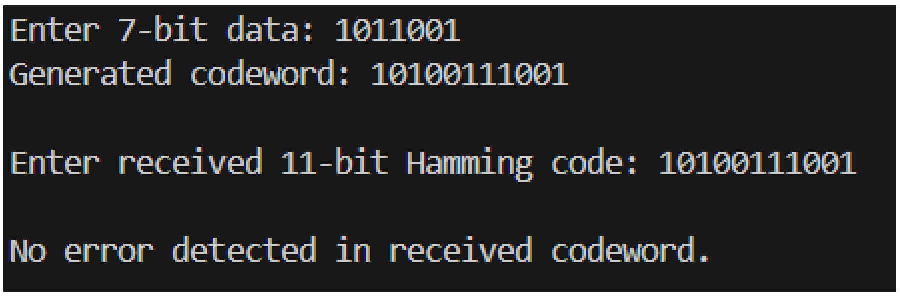
PROGRAM10:

```
import java.util.Scanner;

public class HammingCode {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        char[] data = new char[8];  // 1-based indexing

        char[] codeword = new char[13]; // 1 to 11 used

        char[] received = new char[13]; // for received codeword

        System.out.print("Enter 7-bit data: ");

        String input = sc.next();

        if (input.length() != 7) {

            System.out.println("Error: You must enter exactly 7 bits.");

            return;

        }

        for (int i = 0; i < 7; i++) {

            data[i + 1] = input.charAt(i);

        }

        int j = 0, k = 1;

        for (int i = 1; i <= 11; i++) {

            if (i == Math.pow(2, j)) {

                codeword[i] = '0'; // parity placeholder

                j++;

            } else {

                codeword[i] = data[k++];

            }

        }

        for (int i = 0; i < 4; i++) {
```

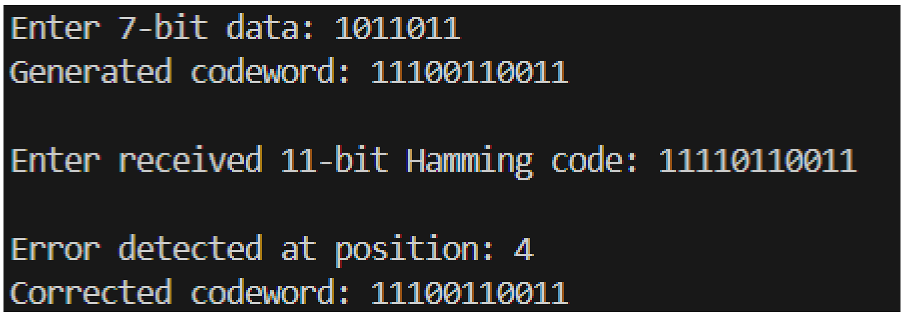
```
    int parityPos = (int) Math.pow(2, i);
    int count = 0;
    for (int p = parityPos; p <= 11; p += 2 * parityPos) {
        for (int q = 0; q < parityPos && (p + q) <= 11; q++) {
            if (codeword[p + q] == '1') {
                count++;
            }
        }
    }
    codeword[parityPos] = (count % 2 == 0) ? '0' : '1';
}
System.out.print("Generated codeword: ");
for (int i = 1; i <= 11; i++) {
    System.out.print(codeword[i]);
}
System.out.print("\n\nEnter received 11-bit Hamming code: ");
String recv = sc.next();
if (recv.length() != 11) {
    System.out.println("Error: You must enter exactly 11 bits.");
    return;
}
for (int i = 1; i <= 11; i++) {
    received[i] = recv.charAt(i - 1);
}
int errorPos = 0;
for (int i = 0; i < 4; i++) {
    int parityPos = (int) Math.pow(2, i);
```

```
            int count = 0;
            for (int p = parityPos; p <= 11; p += 2 * parityPos) {
                for (int q = 0; q < parityPos && (p + q) <= 11; q++) {
                    if (received[p + q] == '1') {
                        count++;
                    }
                }
            }
            if (count % 2 != 0) {
                errorPos += parityPos;
            }
        }
        if (errorPos == 0) {
            System.out.println("\nNo error detected in received codeword.");
        } else {
            System.out.println("\nError detected at position: " + errorPos);
            // Correct the error
            received[errorPos] = (received[errorPos] == '1') ? '0' : '1';
            System.out.print("Corrected codeword: ");
            for (int i = 1; i <= 11; i++) {
                System.out.print(received[i]);
            }
        }
        System.out.println();
        sc.close();
    }
}
```

OUTPUT:



```
Enter 7-bit data: 1011001
Generated codeword: 10100111001

Enter received 11-bit Hamming code: 10100111001

No error detected in received codeword.
```



```
Enter 7-bit data: 1011011
Generated codeword: 11100110011

Enter received 11-bit Hamming code: 11110110011

Error detected at position: 4
Corrected codeword: 11100110011
```