DAYANANDA SAGAR COLLEGE OF ENGINEERING

DEPARTMENT ELECTRONICS & COMMUNICATION ENGINEERING

CCN & IOT LAB MANUAL
VI Semester (18EC6DLIOT)
Autonomous Course

Dr. ANITHA V

Prof. KRUTTHIKA H K

Dr. MAHESH KUMAR N

Dr. SHASHI RAJ K

Mr. NUTHESH KUMAR A

Mrs. GAYATHRI

Name of the Student	:	
Semester /Section	:	
USN	:	
Batch	:	

DAYANANDA SAGAR COLLEGE OF ENGINEERING

SHAVIGE MALLESHWARA HILLS, KUMARSWAMY LAYOUT, BANGALORE – 78

AN AUTONOMOUS INSTITUTE AFFILIATED TO VTU, APPROVED BY AICTE & UGC

ACCREDITED BY NBA & NAAC WITH 'A' GRADE

DAYANANDA SAGAR COLLEGE OF ENGINEERING
DEPARTMENT ELECTRONICS & COMMUNICATION ENGINEERING

Name of the Laboratory	**:**	CCN & IOT Lab
Semester/Year	**:**	VI / 2021 (Autonomous)
No. of Students/Batch	**:**	20
No. of Equipment's	**:**	20
Area in square meters	**:**	109 Sq. Mtrs.
Location	**:**	Level – 2
Total Cost of Lab	**:**	Rs. 37, 77,535/-
Lab In charge/s	**:**	Prof. KRUTTHIKA H.K.
Instructors	**:**	Mr. NUTHESH KUMAR A Mrs. GAYATHRI
HOD	**:**	Dr. T.C. Manjunath, Ph.D. (IIT Bombay)

ABOUT THE COLLEGE & THE DEPARTMENT

The Dayananda Sagar College of Engineering was established in 1979, was founded by Sri R. Dayananda Sagar and is run by the Mahatma Gandhi Vidya Peetha Trust (MGVP). The college offers undergraduate, post-graduates and doctoral programmes under Visvesvaraya Technological University & is currently autonomous institution. MGVP Trust is an educational trust and was promoted by Late. Shri. R. Dayananda Sagar in 1960. The Trust manages 28 educational institutions in the name of “Dayananda Sagar Institutions” (DSI) and multi – Specialty hospitals in the name of Sagar Hospitals - Bangalore, India. Dayananda Sagar College of Engineering is approved by All India Council for Technical Education (AICTE), Govt. of India and affiliated to Visvesvaraya Technological University. It has widest choice of engineering branches having 16 Under Graduate courses & 17 Post Graduate courses. In addition, it has 21 Research Centers in different branches of Engineering catering to research scholars for obtaining Ph.D under VTU. Various courses are accredited by NBA & the college has a NAAC with ISO certification. One of the vibrant & oldest dept is the ECE dept. & is the biggest in the DSI group with 70 staffs & 1200+ students with 10 Ph.D.’s & 30+ staffs pursuing their research in various universities. At present, the department runs a UG course (BE) with an intake of 240 & 2 PG courses (M.Tech.), viz., VLSI Design Embedded Systems & Digital Electronics & Communications with an intake of 18 students each. The department has got an excellent infrastructure of 10 sophisticated labs & dozen class room, R & D centre, etc…

VISION OF THE DEPARTMENT

To achieve continuous improvement in quality technical education for global competence with focus on industry, societal needs, research and professional ethics.

MISSION OF THE DEPARTMENT

- Offering quality education in Electronics and Communication Engineering with effective teaching and learning process in multidisciplinary environment.

- Training the students to take-up the projects in emerging technologies and work with the team spirit.

- To imbibe the professional ethics, development of skills and research culture for better placement opportunities.

PROGRAM EDUCATIONAL OBJECTIVES [PEOs]

After 4 years, the students will be,

PEO-1: Ready to apply the state-of-art technology in industry and meeting the societal needs with knowledge of Electronics and Communication Engineering due to strong academic culture.

PEO-2: Competent in technical and soft skills to be employed with capability of working in multidisciplinary domains.

PEO-3: Professionals, capable of pursuing higher studies in technical, research or management programs.

PROGRAM SPECIFIC OUTCOMES [PSOs]

Students will be able to,

PSO-1: Design, develop and integrate electronics circuits and systems using current practices and standards.

PSO-2: Apply knowledge of hardware and software in designing embedded and communication systems.

INSTRUCTIONS TO THE CANDIDATES

1. Students should come with thorough preparation for the experiment to be conducted.

2. Students will not be permitted to attend the laboratory unless they bring the practical record fully completed in all respects pertaining to the experiment conducted in the previous class.

3. Practical record should be neatly maintained.

4. They should obtain the signature of the staff-in-charge in the observation book after completing each experiment.

5. Theory regarding each experiment should be written in the practical record before procedure in your own words.

6. Ask lab technician for assistance if you have any problem.

7. Save your class work, assignments in system.

8. Do not download or install software without the assistance of the laboratory technician.

9. Do not alter the configuration of the system.

10. Turnoff the systems after use.

DAYANANDA SAGAR COLLEGE OF ENGINEERING

LAB MANUAL FOR
CCN & IOT LABORATORY

Course code: 18EC6DLIOT	Credits : 2
L : P : T : S : 0 : 2 : 0 : 0	CIE Marks : 50
Exam Hours : 3	SEE Marks : 50
Total Hours : 26	Total Marks : 100

COURSE OBJECTIVES:

1. To build the knowledge of networking protocols and interconnections.
2. To impart the programming skill sets to implement the functionalities and responsibilities of Data link and networking layer

COURSE OUTCOMES

CO 1	Apply the difference between wired and wireless network
CO 2	Evaluate the performance parameters of wired and wireless networks
CO 3	Create different wired and wireless networks for data communication
CO 4	
CO 5	
CO 6	

Mapping of Course Outcomes to Program Outcomes:

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1												
CO2												
CO3												
CO4												
CO5												
CO6												

SL NO	COURSE CONTENTS	COs
	PART-A **SIMULATION EXPERIMENTS USING NS2 / NS3 / NCTUNS**	
1	Implement a point to point network with four nodes and duplex links between them. Analyze the network performance by setting the queue size and varying the bandwidth.	CO1,CO2,CO3
2	Implement a four node point to point network with links n0-n2, n1-n2 and n2-n3. Apply TCP agent between n0-n3 and UDP between n1-n3. Apply relevant applications over TCP and UDP agents changing the parameter and determine the number of packets sent by TCP/UDP.	CO1,CO2,CO3
3	Implement Ethernet LAN using n (6-10) nodes. Compare the throughput by changing the error rate and data rate.	CO1,CO2,CO3
4	Implement Ethernet LAN using n nodes and assign multiple traffic to the nodes and obtain congestion window for different sources/ destinations.	CO1,CO2,CO3
5	Implement ESS with transmission nodes in Wireless LAN and obtain the performance parameters.	CO1,CO2,CO3
6	Implementation of any routing algorithm	CO1,CO2,CO3
	PART-B **IOT**	
7		
8		
9		
10		
11		
12		

EXTRA PROGRAMS:

1. Implementation of spanning tree algorithm
2. Implementation of Stop and wait protocol
3. Implement a program to communicate between mobile nodes and suspicious nodes using NS2.
4. Implement UDP wireless communication using NS2
5. Implement TCP communication using NS2

HARDWARE AND SOFTWARE REQUIREMENTS:

Hardware Requirements:

- Processor : Pentium 3 or higher
- RAM : 512MB or more
- Hard Disk : 16GB or more (there should be enough space to hold both Linux and Windows)

SOFTWARE REQUIREMENTS:

- Operating System : Windows, Linux
- Simulation Software: NS2/ NCTUns

TEXT BOOKS:

1. **Introduction to Network Simulator NS2,** Issariyakul, Teerawat, Hossain, Ekram, , Springer US, 2012.

ASSESSMENT PATTERN:

CIE –Continuous Internal Evaluation Lab (50 Marks)

SEE –Semester End Examination Lab (50 Marks)

Bloom's Category	Performance (Day To Day)	Internal Test
Marks (Out of 50)	**25**	**25**
Remember		
Understand		
Apply	05	05
Analyze	10	10
Evaluate	05	05
Create	05	05

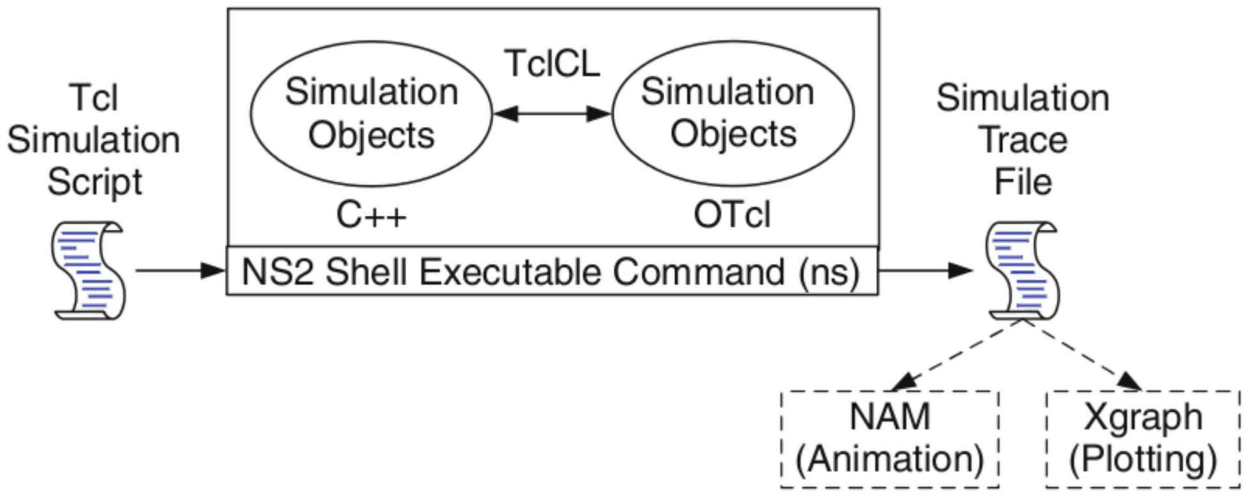
Bloom's Category	Marks Theory(50)
Remember	
Understand	
Apply	15
Analyze	15
Evaluate	10
Create	10

PART -A

INTRODUCTION TO NS-2

- Widely known as NS2, is simply an event driven simulation tool.
- 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.

BASIC ARCHITECTURE OF NS2:



(TOOL COMMAND LANGUAGE) 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.

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]

This is thus the first line in the TCL script. This line declares a new variable using the set command, you can call this variable as you wish, In general, it is declared 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 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 begin 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.

The last line tells the simulator to record all simulation traces in NAM input format. It also gives the file name that the trace will be written to later by the command $ns flush-trace. In our case, this will be the file pointed at by the pointer —$namfile ,i.e the file —out.tr.
The termination of the program is done using a —finish procedure.

#Define a „finish" procedure

Proc finish { } {
global ns tracefile1 namfile
$ns flush-trace

Close $tracefile1
Close $namfile
Exec nam out.nam &
Exit 0

The word proc declares a procedure in this case called **finish** and without arguments. The word **global** is used to tell that we are using variables declared outside the procedure. The simulator method **—flush-trace"** will dump the traces on the respective files. The TCL command **—close"** closes the trace files defined before and **exec** executes the nam program for visualization. The command **exit** will ends the application and return the number 0 as status to the system. Zero is the default for a clean exit. Other values can be used to say that exit because something fails.
At the end of ns program we should call the procedure —finish‖ and specify at what time the termination should occur. For example,

$ns at 125.0 "finish"

will be used to call **—finish** at time 125sec.Indeed,the **at** method of the simulator allows us to schedule events explicitly.

The simulation can then begin using the command,

$ns run

Definition of a network of links and nodes
The way to define a node is,

set n0 [$ns node]

The node is created which is printed by the variable n0. When we shall refer to that node in the script we shall thus write $n0.

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. The definition of the link then includes the way to handle overflow at that queue. In our case, if the buffer capacity of the output queue is exceeded then the last packet to arrive is dropped. Many alternative options exist, such as the RED (Random Early Detectiom) mechanism, the FQ (Fair Queuing), the DRR (Deficit Round Robin), the Stochastic Fair Queuing (SFQ) and the CBQ (which including a priority and a round-robin scheduler).

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

Agents and Applications
We need to define routing (sources, destinations) the agents (protocols) the application that use them.

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

Above 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.

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.

CBR over UDP

A UDP source and destination is defined in a similar way as in the case of TCP.
Instead of defining the rate in the command $cbr set rate_ 0.01Mb, one can define the time interval between transmission of packets using the command.
$cbr set interval_ 0.005

The packet size can be set to some value using
$cbr set packetSize_ <packet size>

SCHEDULING EVENTS

NS is a discrete event based simulation. The tcp script defines when event should occur. The initializing command set ns [new Simulator] creates an event scheduler, and events are then scheduled using the format:
$ns at <time> <event>

The scheduler is started when running ns that is through the command $ns run.
The beginning and end of the FTP and CBR application can be done through the following command
$ns at 0.1 "$cbr start"
$ns at 1.0 " $ftp start"
$ns at 124.0 "$ftp stop"
$ns at 124.5 "$cbr stop"

STRUCTURE OF TRACE FILES:

When tracing into an output ASCII file, the trace is organized in 12 fields as follows in fig shown below, The meaning of the fields are:

1	2	3	4	5	6	7	8	9	10	11	12
Event	**Time**	**From node**	**To node**	**PKT type**	**PKT Size**	**Flags**	**FID**	**Src Addr**	**Dest Addr**	**Seq No.**	**PKT ID**

1. EVENT OR TYPE IDENTIFIER **IPV4-32bit 192.168.2.3 -**

+ :a packet enque event
- :a packet deque event
r :a packet reception event
d :a packet drop (e.g., sent to dropHead_) event
c :a packet collision at the MAC level

2.TIME: Time at which the packet tracing string is created.

3-4. SOURCE AND DESTINATION NODE: Gives the input node and output node of the link at which the event occurs.

5. PACKET NAME: Gives the packet type (eg: Constant Bit Rate (CBR) or (Transmission Control Protocol) TCP).

6. PACKET SIZE: Size of packet in bytes.

7. FLAGS: digit flag string.
"-": disable
1st = "E": ECN (Explicit Congestion Notification) echo is enabled.
2nd = "P": the priority in the IP header is enabled.
3rd : Not in use
4th = "A": Congestion action
5th = "E": Congestion has occurred.
6th = "F": The TCP fast start is used.
7th = "N": Explicit Congestion Notification (ECN) is on.

8. FLOW ID: This is the flow id (fid) of IPv6 that a user can set for each flow at the input OTcl script one can further use this field for analysis purposes; it is also used when specifying stream color for the NAM display.

9-10. SOURCE AND DESTINATION ADDRESS: The format of these two fields is "a.b", where "a" is the address and "b" is the port.

11. SEQUENCE NUMBER: This is the network layer protocol's packet sequence number. Even though UDP implementations in a real network do not use sequence number, ns keeps track of UDP packet sequence number for analysis purposes

12. PACKET UNIQUE ID: The last field shows the unique ID of the packet.

XGRAPH

The xgraph program draws a graph on an x-display given data read from either data file or from standard input if no files are specified. It can display upto 64 independent data sets using different colors and line styles for each set. It annotates the graph with a title, axis labels, grid lines or tick marks, grid labels and a legend.

SYNTAX:
Xgraph [options] file-name

Options are listed here
/-bd <color> (Border)
This specifies the border color of the xgraph window.
/-bg <color> (Background)
This specifies the background color of the xgraph window.

/-fg<color> (Foreground)
This specifies the foreground color of the xgraph window.
/-lf <fontname> (LabelFont)
All axis labels and grid labels are drawn using this font.
/-t<string> (Title Text)
This string is centered at the top of the graph.
/-x <unit name> (XunitText)
This is the unit name for the x-axis. Its default is ―X‖.
/-y <unit name> (YunitText)
This is the unit name for the y-axis. Its default is ―Y‖.

AHO, WEINBERGER AND KERNIGHAN (AWK) SCRIPT:

AWK is a programmable, pattern-matching, and processing tool available in UNIX. It works equally well with text and numbers. AWK is not just a command, but a programming language too. In other words, AWK utility is a pattern scanning and processing language. It searches one or more files to see if they contain lines that match specified patterns and then perform associated actions, such as writing the line to the standard output or incrementing a counter each time it finds a match.

SYNTAX:
awk option 'selection_criteria {action}' file(s)

Here, selection_criteria filters input and select lines for the action component to act upon. The selection_criteria is enclosed within single quotes and the action within the curly braces. Both the selection_criteria and action forms an AWK program.

Example: $ awk „/manager/ {print}" emp.lst
Variables
Awk allows the user to use variables of there choice. You can now print a serial number, using the variable kount, and apply it those directors drawing a salary exceeding 6700:
$ awk –F"|" „$3 == "director" && $6 > 6700 {
kount =kount+1
printf " %3f %20s %-12s %d\n", kount,$2,$3,$6 }" empn.lst

THE –f OPTION: STORING awk PROGRAMS IN A FILE
You should holds large awk programs in separate file and provide them with the awk extension for easier identification. Let's first store the previous program in the file empawk.awk:
$ cat empawk.awk

Observe that this time we haven't used quotes to enclose the awk program. You can now use awk with the –f *filename* option to obtain the same output:
Awk –F"|" –f empawk.awk empn.lst

THE BEGIN AND END SECTIONS:

AWK statements are usually applied to all lines selected by the address, and if there are no addresses, then they are applied to every line of input. But, if you have to print something before processing the first line, for example, a heading, then the BEGIN section can be used gainfully. Similarly, the end section useful in printing some totals after processing is over.

The BEGIN and END sections are optional and take the form

BEGIN {action}

END {action}

These two sections, when present, are delimited by the body of the awk program. You can use them to print a suitable heading at the beginning and the average salary at the end.

BUILT-IN VARIABLES:

Awk has several built-in variables. They are all assigned automatically, though it is also possible for a user to reassign some of them. You have already used NR, which signifies the record number of the current line. We'll now have a brief look at some of the other variable.

The FS Variable: as stated elsewhere, awk uses a contiguous string of spaces as the default field delimiter. FS redefines this field separator, which in the sample database happens to be the |. When used at all, it must occur in the BEGIN section so that the body of the program knows its value before it starts processing:

BEGIN {FS="|"}

This is an alternative to the –F option which does the same thing.

The OFS Variable: when you used the print statement with comma-separated arguments, each argument was separated from the other by a space. This is awk's default output field separator, and can reassigned using the variable OFS in the BEGIN section:

BEGIN { OFS="~" }

When you reassign this variable with a ~ (tilde), awk will use this character for delimiting the print arguments. This is a useful variable for creating lines with delimited fields.

The NF variable: NF comes in quite handy for cleaning up a database of lines that don't contain the right number of fields. By using it on a file, say emp.lst, you can locate those lines not having 6 fields, and which have crept in due to faulty data entry:

$awk „BEGIN {FS = "|"}

NF! =6 { Print "Record No ", NR, "has", "fields"}‟ empx.lst

STEPS FOR EXECUTION

- Create a folder in **roots** folder. Open the terminal and type

  [root@localhost ~]#**cd** ***foldername***

- To open editor and type program.

  [root@localhost ~]# **gedit** ***programname*****.tcl**

  *Program name should have the extension " **.tcl** "
  *Type the program in the editor window and then save it.

- To open editor and type **awk** program.

  [root@localhost ~]# **gedit** ***programname*****.awk**

  *Program name should have the extension "**.awk** "
  * Type the program in the editor window and then save it.

- To run the simulation program,

  [root@localhost~]# ns programname.tcl

Here **"ns"** indicates network simulator. We get the topology shown in the snapshot. Now press the play button in the simulation window and the simulation will begins.

- To see the trace file contents open the file as ,

[root@localhost~]# **gedit programname.tr**

- After simulation is completed run **awk file** to see the output,

  [root@localhost~]# **awk –f** ***programname*****.awk programname.tr**

PROGRAM 1:

TO IMPLEMENT A POINT-TO-POINT NETWORK WITH FOUR NODES AND DUPLEX LINKS BETWEEN THEM. ANALYZE THE NETWORK PERFORMANCE BY SETTING QUEUE SIZE AND VARYING THE BANDWIDTH.

ALGORITHM:

STEP 1: Create the simulator object ns for designing the given simulation.
STEP 2: Open the trace file and nam file in the write mode.
STEP 3: Create the 4 nodes of the simulation using the 'set' command and duplex link between them.
STEP 4: Create a queue size between nodes.
STEP 5: Create UDP agent for the nodes and attach these agents to the nodes.
STEP 6: The traffic generator used is UDP and measured in terms of cbr0 and cbr1.
STEP 7: Configure node2 and node 3 as the null and attach it.
STEP 8: Connect source and destination nodes using 'connect' command.
STEP 9: Create the Packet size and time interval between each packet coming using the cbr object instance created.
STEP 10: Schedule the start and stop of events for UDP agent with 0.00msec and 10msec for cbr0 agent.
STEP 11: Schedule the start and stop of events for UDP agent with 2.00msec and 12 msec for cbr1 agent.
STEP 12: Schedule the simulation for 13 msec.

PROGRAM:	**COMMENTS**
set ns [new Simulator]	;#creating a new variable ns
set tf [open p1.tr w]	;#open trace file p1.tr in writable mode
$ns trace-all $tf	;#main class object ns linked with trace file object tf
set nf [open p1.nam w]	;#open nam file p1.nam in writable mode
$ns namtrace-all $nf	;#main class object ns linked with nam file object nf
set n0 [$ns node]	;#creation of nodes
set n1 [$ns node]	
set n2 [$ns node]	
set n3 [$ns node]	

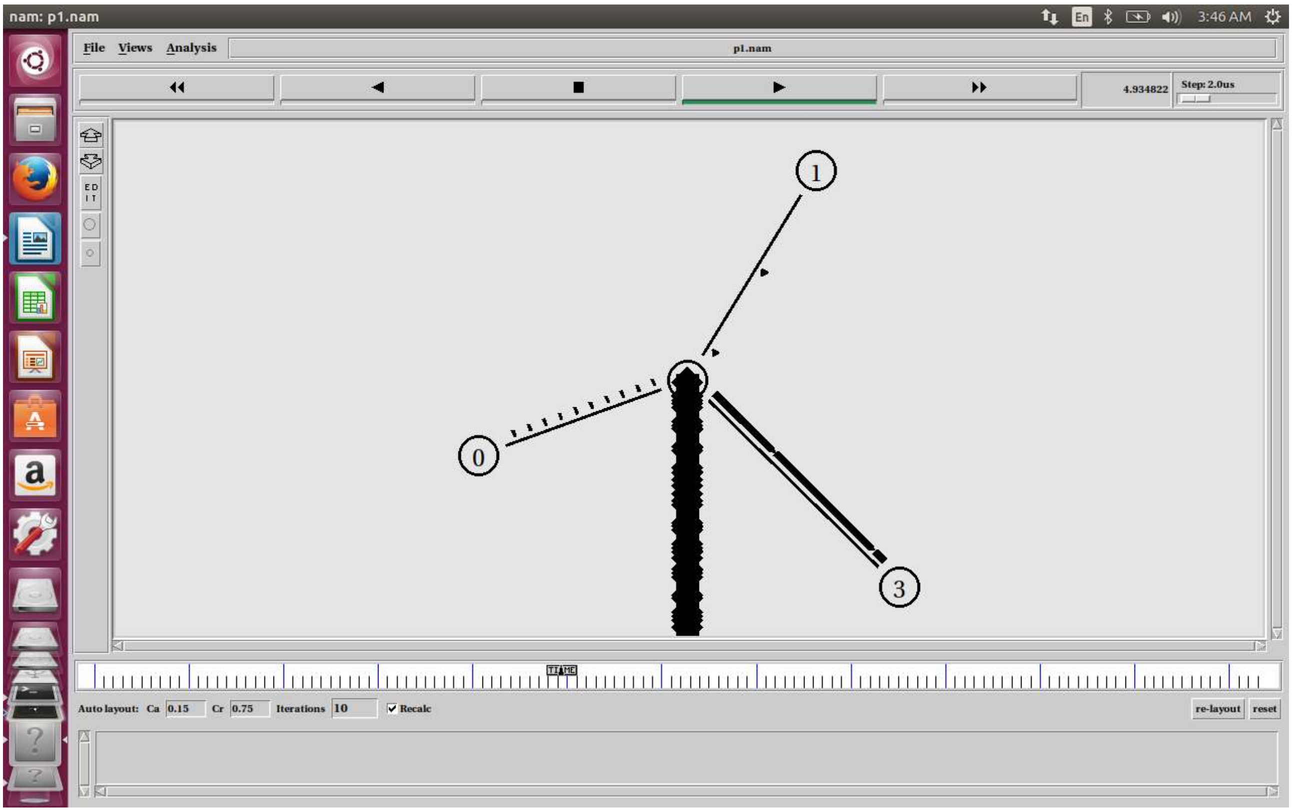
```
$ns duplex-link $n0 $n2 20Mb 10ms DropTail         ;#forming duplex connection between the nodes
$ns duplex-link $n1 $n2 10Mb 10ms DropTail        ;#and vary the bandwidth in this link
$ns duplex-link $n2 $n3 0.7Mb 10ms DropTail

$ns set queue-limit $n0 $n2 10                    ;#Assigning Queuesize between the nodes
$ns set queue-limit $n1 $n2 10
$ns set queue-limit $n2 $n3 5

set udp0 [new Agent/UDP]                          ;#creating object(agent) for the class  Agent/UDP
set udp1 [new Agent/UDP]                          ;#creating object(agent) for the class  Agent/UDP
set null [new Agent/Null]                 ;#creating  Destination object(agent) for the class   Agent/Null

set cbr0 [new Application/Traffic/CBR]       ;#creating object(agent) for the class
                                              ;#Application/Traffic/CBR
set cbr1 [new Application/Traffic/CBR]

$ns attach-agent $n0 $udp0                        ;#Attaching agents
$ns attach-agent $n1 $udp1
$ns attach-agent $n2 $null
$ns attach-agent $n3 $null

$cbr0 attach-agent $udp0
$cbr1 attach-agent $udp1

$ns connect $udp0 $null                          ;#connecting source and destination node agents
$ns connect $udp1 $null

$cbr0 set packetSize_ 512                         ;#assigning values of packet size and time interval
$cbr0 set interval_ 0.001

$cbr1 set packetSize_ 512
$cbr1 set interval_ 0.005
```

```
proc finish { } {                                   ;#terminate Simulation
global ns nf tf
$ns flush-trace
close $tf
close $nf
exec nam p1.nam &
exit 0
}

$ns at 0.0 "$cbr0 start"                            ;#Setting time for start and stop the packet transmission.
$ns at 10.0 "$cbr0 stop"
$ns at 2.0 "$cbr1 start"
$ns at 12.0 "$cbr1 stop"

$ns at 13.0  "finish"                               ;#Setting time to stop the simulation
$ns run
```

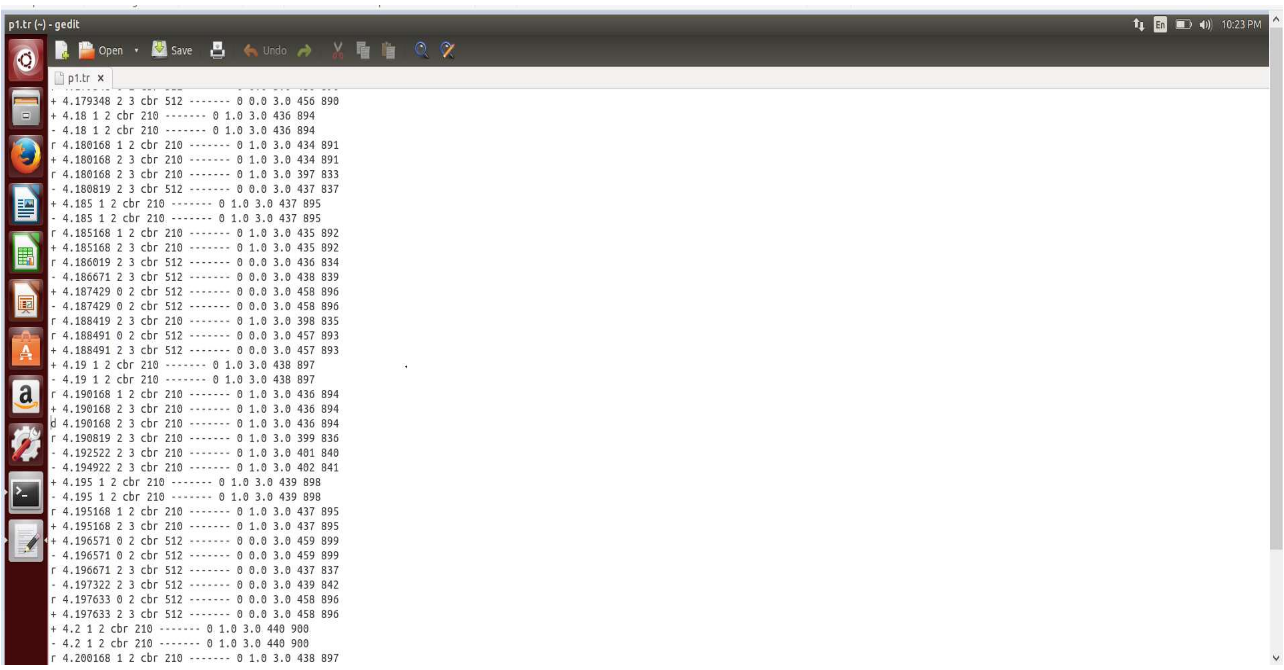
(AWK) SCRIPT:

```
BEGIN{
	count=0
}

{
	if($1=="d")
		count++;
}

END {
	printf("Number of packets dropped: %d\n",count);
}
```

TOPOLOGY AND NAM RESULT:



TRACE FILE OUTPUT:


p1.tr (~) - gedit

```
+ 4.179348 2 3 cbr 512 ------- 0 0.0 3.0 456 890
+ 4.18 1 2 cbr 210 ------- 0 1.0 3.0 436 894
- 4.18 1 2 cbr 210 ------- 0 1.0 3.0 436 894
r 4.180168 1 2 cbr 210 ------- 0 1.0 3.0 434 891
+ 4.180168 2 3 cbr 210 ------- 0 1.0 3.0 434 891
r 4.180168 2 3 cbr 210 ------- 0 1.0 3.0 397 833
- 4.180819 2 3 cbr 512 ------- 0 0.0 3.0 437 837
+ 4.185 1 2 cbr 210 ------- 0 1.0 3.0 437 895
- 4.185 1 2 cbr 210 ------- 0 1.0 3.0 437 895
r 4.185168 1 2 cbr 210 ------- 0 1.0 3.0 435 892
+ 4.185168 2 3 cbr 210 ------- 0 1.0 3.0 435 892
r 4.186019 2 3 cbr 512 ------- 0 0.0 3.0 436 834
- 4.186671 2 3 cbr 512 ------- 0 0.0 3.0 438 839
+ 4.187429 0 2 cbr 512 ------- 0 0.0 3.0 458 896
- 4.187429 0 2 cbr 512 ------- 0 0.0 3.0 458 896
r 4.188419 2 3 cbr 210 ------- 0 1.0 3.0 398 835
r 4.188491 0 2 cbr 512 ------- 0 0.0 3.0 457 893
+ 4.188491 2 3 cbr 512 ------- 0 0.0 3.0 457 893
+ 4.19 1 2 cbr 210 ------- 0 1.0 3.0 438 897
- 4.19 1 2 cbr 210 ------- 0 1.0 3.0 438 897
r 4.190168 1 2 cbr 210 ------- 0 1.0 3.0 436 894
+ 4.190168 2 3 cbr 210 ------- 0 1.0 3.0 436 894
d 4.190168 2 3 cbr 210 ------- 0 1.0 3.0 436 894
r 4.190819 2 3 cbr 210 ------- 0 1.0 3.0 399 836
- 4.192522 2 3 cbr 210 ------- 0 1.0 3.0 401 840
- 4.194922 2 3 cbr 210 ------- 0 1.0 3.0 402 841
+ 4.195 1 2 cbr 210 ------- 0 1.0 3.0 439 898
- 4.195 1 2 cbr 210 ------- 0 1.0 3.0 439 898
r 4.195168 1 2 cbr 210 ------- 0 1.0 3.0 437 895
+ 4.195168 2 3 cbr 210 ------- 0 1.0 3.0 437 895
+ 4.196571 0 2 cbr 512 ------- 0 0.0 3.0 459 899
- 4.196571 0 2 cbr 512 ------- 0 0.0 3.0 459 899
r 4.196671 2 3 cbr 512 ------- 0 0.0 3.0 437 837
- 4.197322 2 3 cbr 512 ------- 0 0.0 3.0 439 842
r 4.197633 0 2 cbr 512 ------- 0 0.0 3.0 458 896
+ 4.197633 2 3 cbr 512 ------- 0 0.0 3.0 458 896
+ 4.2 1 2 cbr 210 ------- 0 1.0 3.0 440 900
- 4.2 1 2 cbr 210 ------- 0 1.0 3.0 440 900
r 4.200168 1 2 cbr 210 ------- 0 1.0 3.0 438 897
```

OUTPUT:

Case 1:

Source Node	Destination Node	Bandwidth	Delay	Queue Limit	No. of packets Dropped
n0	n2	20Mb	10ms	10	9902
n1	n2	10Mb	10ms	10	
n2	n3	0.7Mb	10ms	5	

Case 2:

Source Node	Destination Node	Bandwidth	Delay	Queue Limit	No. of packets Dropped
n0	n2	200Mb	10ms	100	0
n1	n2	100Mb	10ms	100	
n2	n3	70Mb	10ms	50	

RESULT: Network Performance analyzed by setting queue size and varying bandwidth using NS-2.

PROGRAM 2:

TO IMPLEMENT FOUR NODE POINT TO POINT NETWORK WITH LINKS N0-N1,N1-N2 AND N2-N3. APPLY TCP AGENT BETWEEN N0-N3 AND UDP BETWEEN N1-N3. APPLY RELEVANT APPLICATION OVER TCP AND UDP AGENTS CHANGING THE PARAMETER AND DETERMINE THE NUMBER OF PACKETS SENT BY TCP/UDP.

ALGORITHM:

STEP 1: Create the simulator object ns for designing the given simulation.
STEP 2: Open the trace file and nam file in the write mode.
STEP 3: Create the 4 nodes of the simulation using the 'set' command and duplex link between them.
STEP 4: Create a queue size between nodes.
STEP 5: Create UDP agent between the nodes n0 & n3 and attach these agents to the nodes.
STEP 6: Create TCP agent between the nodes n1 & n3 and attach these agents to the nodes.
STEP 7: The traffic generator used between n0 & n3 is UDP and measured in terms of cbr.
STEP 8: The traffic generator used between n0 & n3 is FTP and measured in terms of ftp.
STEP 9: Configure node 3 as the null and sink in UDP and TCP protocol respectively and attach it.
STEP 10: Connect source and destination nodes using 'connect' command.
STEP 11: Create the Packet size and time interval between each packet coming using the cbr & ftp object instance created.
STEP 12: Schedule the start and stop of events for FTP agent with 0.00msec and 10msec for ftp agent.
STEP 13: Schedule the start and stop of events for UDP agent with 2.00msec and 12 msec for cbr agent.
STEP 14: Schedule the simulation for 13 msec.

PROGRAM:

```
set ns [new Simulator]

set tf [open p2.tr w]
$ns trace-all $tf

set nf [open p2.nam w]
$ns namtrace-all $nf


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

$ns duplex-link $n0 $n2 20Mb 10ms DropTail
$ns duplex-link $n1 $n2 10Mb 10ms DropTail
$ns duplex-link $n2 $n3 0.7Mb 10ms DropTail
```

```
$ns set queue-limit $n0 $n2 10
$ns set queue-limit $n1 $n2 10
$ns set queue-limit $n2 $n3 5

set tcp [new Agent/TCP]
set udp [new Agent/UDP]

set tcpsink [new Agent/TCPSink]
set null [new Agent/Null]

set cbr [new Application/Traffic/CBR]
set ftp [new Application/FTP]


$ns attach-agent $n0 $tcp
$ns attach-agent $n1 $udp
$ns attach-agent $n3 $tcpsink
$ns attach-agent $n3 $null

$ftp attach-agent $tcp
$cbr attach-agent $udp


$ns connect $udp $null
$ns connect $tcp $tcpsink

$cbr set packetSize_  512
$cbr set interval_  0.005

$ftp set packetSize_  512
$ftp set interval_  0.005


proc finish {} {
global ns nf tf
$ns flush-trace
close $tf
close $nf
exec nam p2.nam &
exit 0
}


$ns at 0.0 "$ftp start"
$ns at 10.0 "$ftp stop"
$ns at 2.0 "$cbr start"
$ns at 12.0 "$cbr stop"

$ns at 13.0 "finish"
$ns run
```

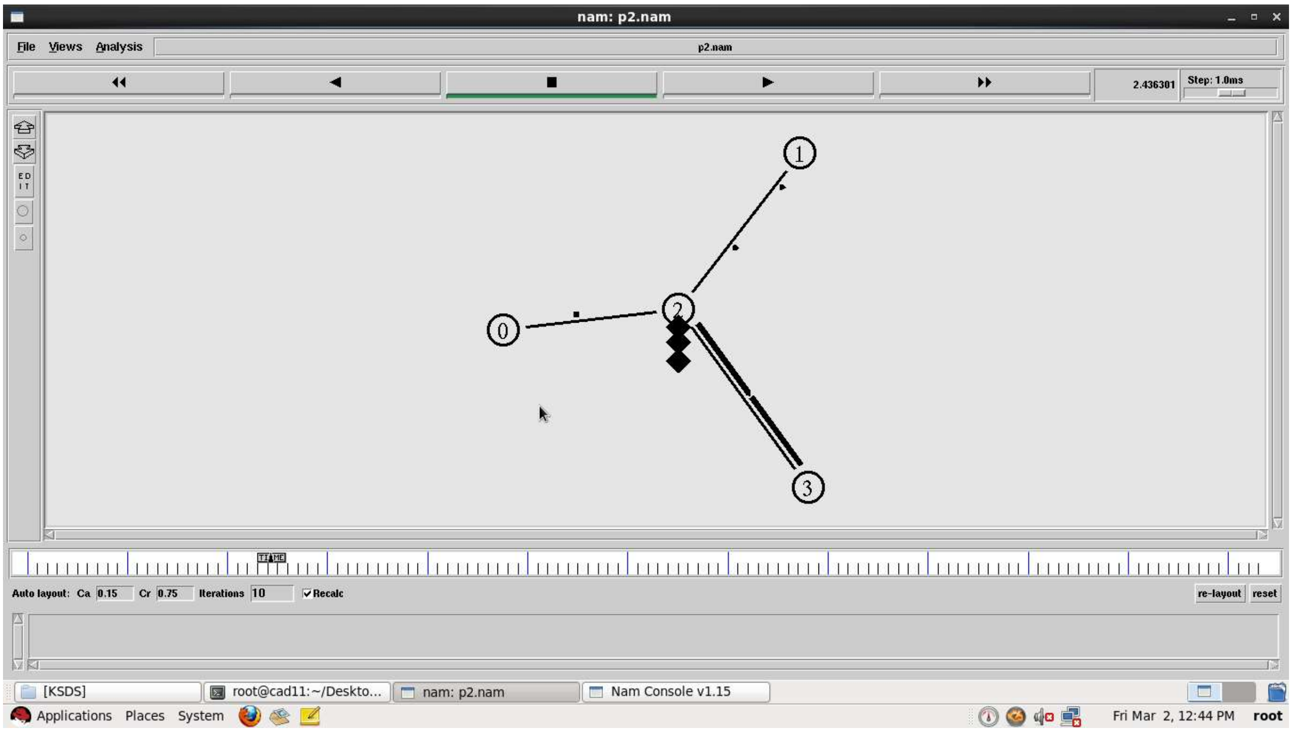
AWK SCRIPT:

```
BEGIN{
tcp=0;
cbr=0;
}
{
if($1=="-" && $5=="tcp")
tcp++
if($1=="-" && $5=="cbr")
cbr++
}
END{
printf("\n no.of tcp packets sent= %d \n",tcp);
printf("\n no.of cbr packets sent= %d \n",cbr);
}
```

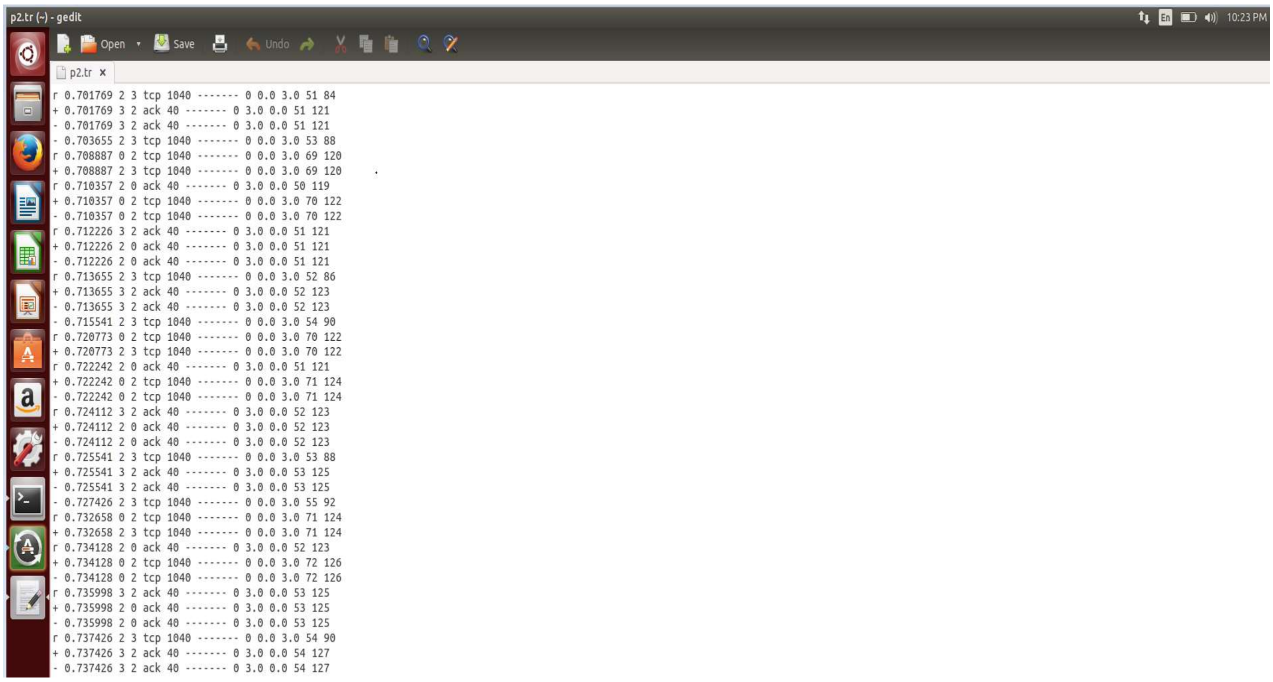
TO RUN:

```
ns filename.tcl
awk -f awkfilename.awk tracefilename.tr
```

TOPOLOGY AND NAM RESULT:



TRACE FILE OUTPUT:



```
r 0.701769 2 3 tcp 1040 ------- 0 0.0 3.0 51 84
+ 0.701769 3 2 ack 40 ------- 0 3.0 0.0 51 121
- 0.701769 3 2 ack 40 ------- 0 3.0 0.0 51 121
- 0.703655 2 3 tcp 1040 ------- 0 0.0 3.0 53 88
r 0.708887 0 2 tcp 1040 ------- 0 0.0 3.0 69 120
+ 0.708887 2 3 tcp 1040 ------- 0 0.0 3.0 69 120
r 0.710357 2 0 ack 40 ------- 0 3.0 0.0 50 119
+ 0.710357 0 2 tcp 1040 ------- 0 0.0 3.0 70 122
- 0.710357 0 2 tcp 1040 ------- 0 0.0 3.0 70 122
r 0.712226 3 2 ack 40 ------- 0 3.0 0.0 51 121
+ 0.712226 2 0 ack 40 ------- 0 3.0 0.0 51 121
- 0.712226 2 0 ack 40 ------- 0 3.0 0.0 51 121
r 0.713655 2 3 tcp 1040 ------- 0 0.0 3.0 52 86
+ 0.713655 3 2 ack 40 ------- 0 3.0 0.0 52 123
- 0.713655 3 2 ack 40 ------- 0 3.0 0.0 52 123
- 0.715541 2 3 tcp 1040 ------- 0 0.0 3.0 54 90
r 0.720773 0 2 tcp 1040 ------- 0 0.0 3.0 70 122
+ 0.720773 2 3 tcp 1040 ------- 0 0.0 3.0 70 122
r 0.722242 2 0 ack 40 ------- 0 3.0 0.0 51 121
+ 0.722242 0 2 tcp 1040 ------- 0 0.0 3.0 71 124
- 0.722242 0 2 tcp 1040 ------- 0 0.0 3.0 71 124
r 0.724112 3 2 ack 40 ------- 0 3.0 0.0 52 123
+ 0.724112 2 0 ack 40 ------- 0 3.0 0.0 52 123
- 0.724112 2 0 ack 40 ------- 0 3.0 0.0 52 123
r 0.725541 2 3 tcp 1040 ------- 0 0.0 3.0 53 88
+ 0.725541 3 2 ack 40 ------- 0 3.0 0.0 53 125
- 0.725541 3 2 ack 40 ------- 0 3.0 0.0 53 125
- 0.727426 2 3 tcp 1040 ------- 0 0.0 3.0 55 92
r 0.732658 0 2 tcp 1040 ------- 0 0.0 3.0 71 124
+ 0.732658 2 3 tcp 1040 ------- 0 0.0 3.0 71 124
r 0.734128 2 0 ack 40 ------- 0 3.0 0.0 52 123
+ 0.734128 0 2 tcp 1040 ------- 0 0.0 3.0 72 126
- 0.734128 0 2 tcp 1040 ------- 0 0.0 3.0 72 126
r 0.735998 3 2 ack 40 ------- 0 3.0 0.0 53 125
+ 0.735998 2 0 ack 40 ------- 0 3.0 0.0 53 125
- 0.735998 2 0 ack 40 ------- 0 3.0 0.0 53 125
r 0.737426 2 3 tcp 1040 ------- 0 0.0 3.0 54 90
+ 0.737426 3 2 ack 40 ------- 0 3.0 0.0 54 127
- 0.737426 3 2 ack 40 ------- 0 3.0 0.0 54 127
```

PROGRAM 3:

SIMULATE AN ETHERNET LAN USING N NODES (0-6), CHANGE THE ERROR RATE AND DATA RATE AND COMPARE THE THROUGHPUT.

ALGORITHM:

STEP 1: Create the simulator object ns for designing the given simulation.
STEP 2: Open the trace file and nam file in the write mode.
STEP 3: Create the nodes of the simulation using the 'set' command.
STEP 4: Create Ethernet LAN using make-LAN command and connect the nodes to the LAN.
STEP 5: Create TCP agent for the nodes and attach these agents to the nodes.
STEP 6: The traffic generator used is FTP for both node1 and node5.
STEP 7: Add error node between the nodes 3 and 6.
STEP 8: Configure node5 as the sink and attach it.
STEP 9: Connect node1 and node5 agents using 'connect' command.
STEP 10: Setting colour for the TCP packets.
STEP 11: Schedule the events for FTP agent 0.1msec.
STEP 12: Schedule the simulation for 5 msec.

```
set ns [new Simulator]

set tf [open p3.tr w]
$ns trace-all $tf

set nf [open p3.nam w]
$ns namtrace-all $nf

$ns color 1 "blue"

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]

$n1 label "Source/UDP"
$n3 label "Error Node"
$n5 label "Destination"


;#The below code is used to create a two Lans (Lan1 and #Lan2).
```

```
$ns make-lan "$n0 $n1 $n2 $n3" 100Mb 10ms LL Queue/DropTail Mac/802_3
$ns make-lan "$n4 $n5 $n6 " 100Mb 10ms LL Queue/DropTail Mac/802_3

$ns duplex-link $n3 $n6 100Mb 10ms DropTail

set udp1 [new Agent/UDP]
set cbr1 [ new Application/Traffic/CBR]
set null5 [new Agent/Null]

$ns attach-agent $n1 $udp1
$cbr1 attach-agent $udp1
$ns attach-agent $n5 $null5

$ns connect $udp1 $null5

$cbr1 set packetSize_ 1000
$cbr1 set interval_ 0.0001      ;# This is the data rate.Change
                                ;# this to vary the throughput.

set err [new ErrorModel]        ;# The code is used to add an error model
$ns lossmodel $err $n3 $n6      ;#between the nodes n3 and n6.
$err set rate_ 0.1              ;# This is the error rate. Change this
                                ;# rate to add errors between n3 and n6.

$udp1 set class_ 1

proc finish { } {
global nf ns tf
close $nf
close $tf
exec nam p3.nam &
exit 0
}

$ns at 0.1 "$cbr1 start"
$ns at 5.0 "finish"
$ns run
```

AWK SCRIPT:

```
BEGIN {
  Rpacketsize=0;
  Rtimeinterval=0;
}
{
if ($1=="r" && $3=="3" && $4=="6")
Rpacketsize = Rpacketsize+$6;
Rtimeinterval=$2;
}
END {
printf ("throughput:%f Mbps\n",(Rpacketsize/Rtimeinterval)*(8/1000000));
```

}

OUTPUT:

The throughput can be analysed by changing the data rate and error rate as shown below.

Case 1: Fix the data rate to 0.0001 and vary the error rate then throughput decreases as shown below.

Error Rate
Data Rate
Throughput
0.1
0.0001

0.2
0.0001

0.3
0.0001

0.4
0.0001

Case 2: Fix the error rate to 0.1 and vary the data rate then throughput increases as shown below.

Error Rate
Data Rate
Throughput
0.1
0.1

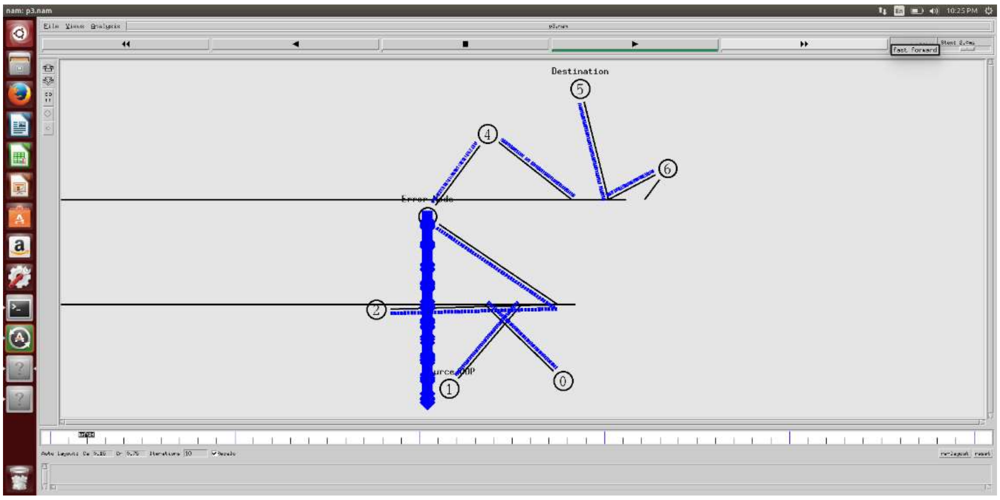
0.1
0.01

0.1
0.001

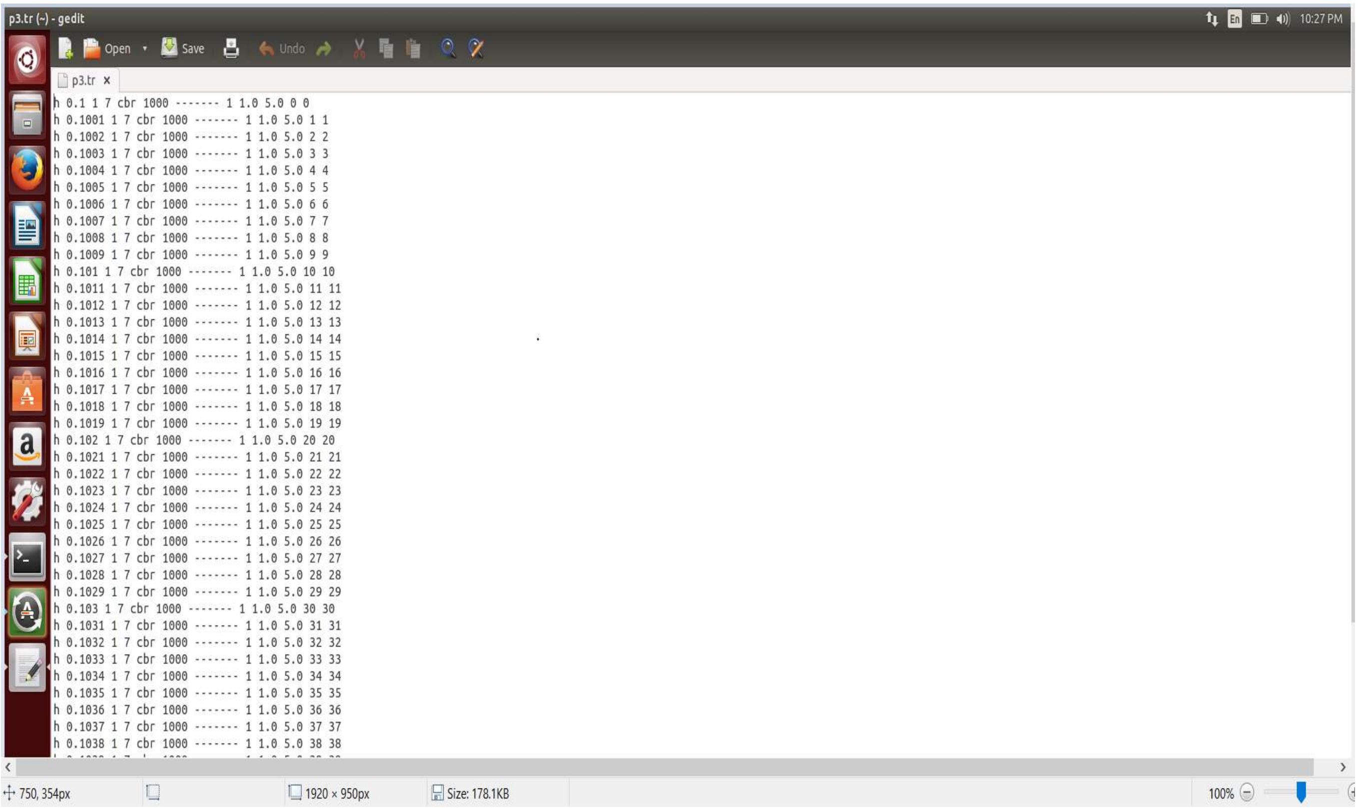
0.1
0.0001

RESULT: Simulation of an Ethernet lan using 7 nodes, the Network throughput calculated by varying the error rate and data rate using NS-2.

NAM Window output:



TRACE FILE:




```
h 0.1 1 7 cbr 1000 ------- 1 1.0 5.0 0 0
h 0.1001 1 7 cbr 1000 ------- 1 1.0 5.0 1 1
h 0.1002 1 7 cbr 1000 ------- 1 1.0 5.0 2 2
h 0.1003 1 7 cbr 1000 ------- 1 1.0 5.0 3 3
h 0.1004 1 7 cbr 1000 ------- 1 1.0 5.0 4 4
h 0.1005 1 7 cbr 1000 ------- 1 1.0 5.0 5 5
h 0.1006 1 7 cbr 1000 ------- 1 1.0 5.0 6 6
h 0.1007 1 7 cbr 1000 ------- 1 1.0 5.0 7 7
h 0.1008 1 7 cbr 1000 ------- 1 1.0 5.0 8 8
h 0.1009 1 7 cbr 1000 ------- 1 1.0 5.0 9 9
h 0.101 1 7 cbr 1000 ------- 1 1.0 5.0 10 10
h 0.1011 1 7 cbr 1000 ------- 1 1.0 5.0 11 11
h 0.1012 1 7 cbr 1000 ------- 1 1.0 5.0 12 12
h 0.1013 1 7 cbr 1000 ------- 1 1.0 5.0 13 13
h 0.1014 1 7 cbr 1000 ------- 1 1.0 5.0 14 14
h 0.1015 1 7 cbr 1000 ------- 1 1.0 5.0 15 15
h 0.1016 1 7 cbr 1000 ------- 1 1.0 5.0 16 16
h 0.1017 1 7 cbr 1000 ------- 1 1.0 5.0 17 17
h 0.1018 1 7 cbr 1000 ------- 1 1.0 5.0 18 18
h 0.1019 1 7 cbr 1000 ------- 1 1.0 5.0 19 19
h 0.102 1 7 cbr 1000 ------- 1 1.0 5.0 20 20
h 0.1021 1 7 cbr 1000 ------- 1 1.0 5.0 21 21
h 0.1022 1 7 cbr 1000 ------- 1 1.0 5.0 22 22
h 0.1023 1 7 cbr 1000 ------- 1 1.0 5.0 23 23
h 0.1024 1 7 cbr 1000 ------- 1 1.0 5.0 24 24
h 0.1025 1 7 cbr 1000 ------- 1 1.0 5.0 25 25
h 0.1026 1 7 cbr 1000 ------- 1 1.0 5.0 26 26
h 0.1027 1 7 cbr 1000 ------- 1 1.0 5.0 27 27
h 0.1028 1 7 cbr 1000 ------- 1 1.0 5.0 28 28
h 0.1029 1 7 cbr 1000 ------- 1 1.0 5.0 29 29
h 0.103 1 7 cbr 1000 ------- 1 1.0 5.0 30 30
h 0.1031 1 7 cbr 1000 ------- 1 1.0 5.0 31 31
h 0.1032 1 7 cbr 1000 ------- 1 1.0 5.0 32 32
h 0.1033 1 7 cbr 1000 ------- 1 1.0 5.0 33 33
h 0.1034 1 7 cbr 1000 ------- 1 1.0 5.0 34 34
h 0.1035 1 7 cbr 1000 ------- 1 1.0 5.0 35 35
h 0.1036 1 7 cbr 1000 ------- 1 1.0 5.0 36 36
h 0.1037 1 7 cbr 1000 ------- 1 1.0 5.0 37 37
h 0.1038 1 7 cbr 1000 ------- 1 1.0 5.0 38 38
```

PROGRAM 4:

IMPLEMENT ETHERNET LAN USING N NODES AND ASSIGN MULTIPLE TRAFFIC TO THE NODES AND OBTAIN CONGESTION WINDOW FOR DIFFERENT SOURCES AND DESTINATIONS.

ALGORITHM:

STEP 1: Create the simulator object ns for designing the given simulation.

STEP 2: Open the trace file and nam file in the write mode.

STEP 3: Create the nodes of the simulation using the 'set' command.

STEP 4: Create Ethernet LAN using make-LAN command and connect the nodes to the LAN.

STEP 5: Create TCP agent for the nodes and attach these agents to the nodes.

STEP 6: The traffic generator used is FTP for both node0 and node2.

STEP 7: Configure node3 and node1 as the sink and attach it.

STEP 8: Connect node0 and node3, node2 and node1 agents using 'connect' command.

STEP 9: Set color for the TCP packets.

STEP10: Define congestion window maximum value.

STEP 11: Schedule the events for FTP agent.

STEP 12: Schedule the simulation for 5 msec.

PROGRAM:

```
set ns [new Simulator]

set tf [open p4.tr w]
$ns trace-all $tf

set nf [open p4.nam w]
$ns namtrace-all $nf

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

$ns make-lan "$n0 $n1 $n2 $n3" 10Mb 10ms LL Queue/DropTail Mac/802_3

set tcp1 [new Agent/TCP]
set ftp1 [new Application/FTP]
set sink1 [new Agent/TCPSink]
set tcp2 [new Agent/TCP]
```

```
set ftp2 [new Application/FTP]
set sink2 [new Agent/TCPSink]

$ns attach-agent $n0 $tcp1
$ftp1 attach-agent $tcp1
$ns attach-agent $n3 $sink1
$ns connect $tcp1 $sink1

$ns attach-agent $n2 $tcp2
$ftp2 attach-agent $tcp2
$ns attach-agent $n1 $sink2
$ns connect $tcp2 $sink2

set file1 [open file1.tr w]
$tcp1 attach $file1
$tcp1 trace cwnd_
$tcp1 set maxcwnd_ 20

set file2 [open file2.tr w]
$tcp2 attach $file2
$tcp2 trace cwnd_
$tcp2 set maxcwnd_ 30

$ns color 1 "red"
$ns color 2 "blue"

$tcp1 set class_ 1
$tcp2 set class_ 2

proc finish { } {
global nf tf ns
$ns flush-trace
exec nam p4.nam &
close $nf
close $tf
exit 0
}

$ns at 0.1 "$ftp1 start"
$ns at 1.5 "$ftp1 stop"

$ns at 2 "$ftp1 start"
$ns at 3 "$ftp1 stop"

$ns at 0.2 "$ftp2 start"
$ns at 2 "$ftp2 stop"

$ns at 2.5 "$ftp2 start"
$ns at 4 "$ftp2 stop"

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

AWK SCRIPT:

```
BEGIN{
}
{
if($6=="cwnd_")
printf("%f\t%f\t\n",$1,$7);
}
END{
}
```

TO RUN:

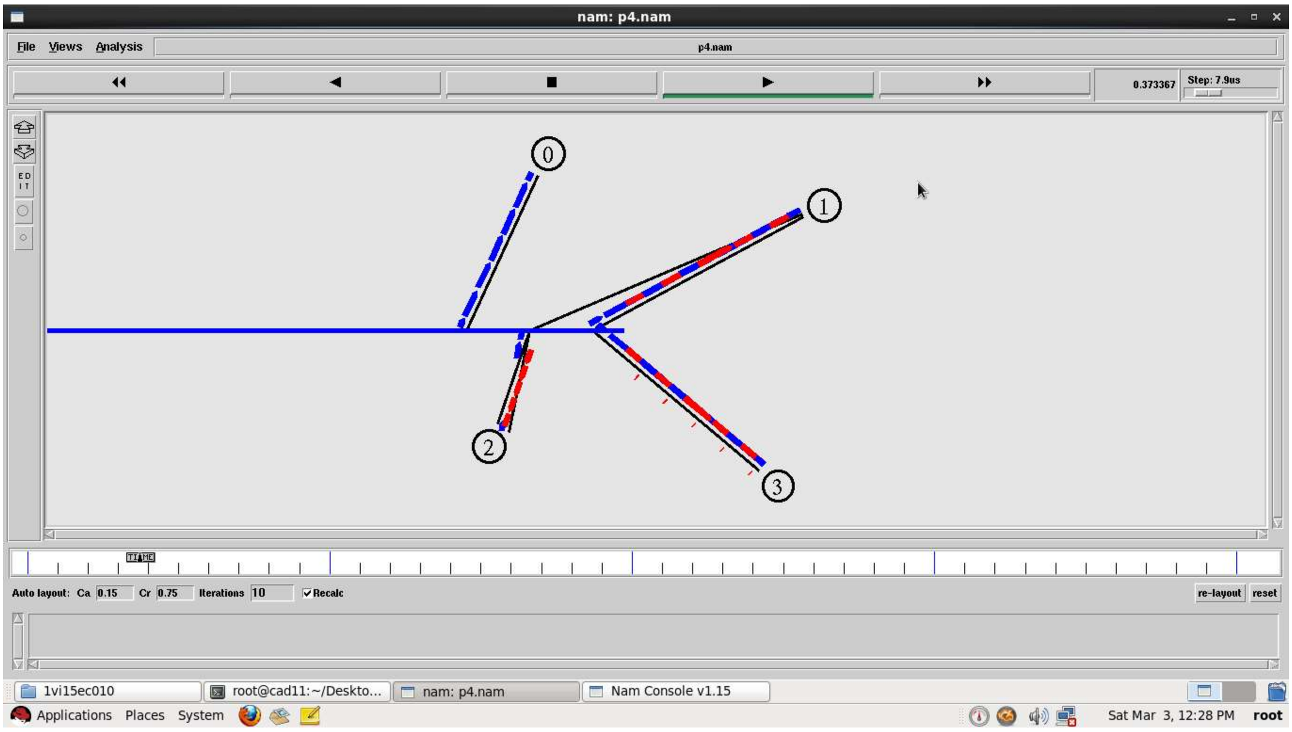
To see the output in the xgraph use these commands:

```
awk -f p4.awk file1.tr  > file1
awk -f p4.awk file2.tr  > file2
xgraph -x “time” -y “convalue” file1 file2
```

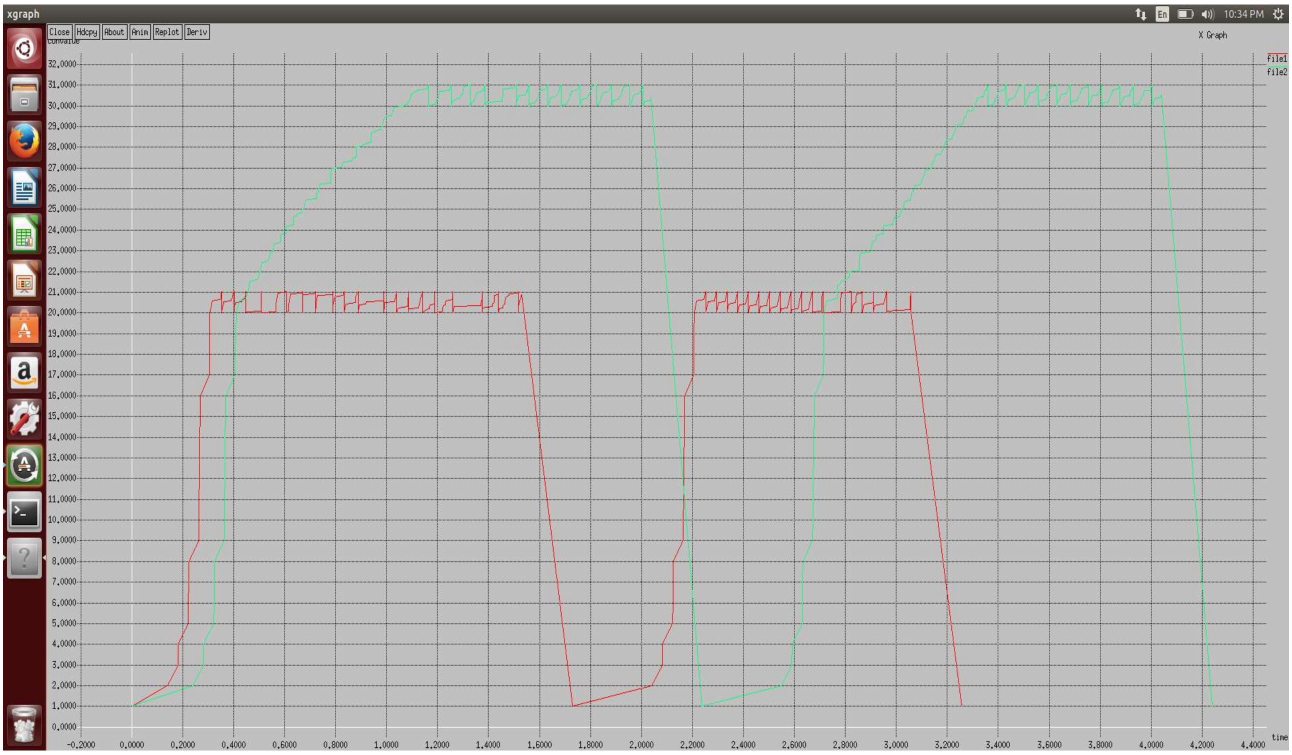
To know the contents of file1 and file2 use these commands:

```
awk -f p4.awk file1.tr
awk -f p4.awk file2.tr
```

TOPOLOGY AND NAM RESULT:



<u>Plot of congestion window for different sources and destination (tcp1 and tcp2)</u>



<u>RESULT:</u> Ethernet LAN using n(four) nodes implemented by creating multiple traffic to the nodes and observed congestion window for different sources and destinations.

PROGRAM 5:

TO IMPLEMENT ESS WITH TRANSMISSION NODES IN WIRELESS LAN AND OBTAIN THE PERFORMANCE PARAMETERS.

ALGORITHM:

STEP 1: Create GOD for wireless nodes.

STEP 2: Create the simulator object ns for designing the given simulation.

STEP 3: Open the trace file and nam file in the write mode.

STEP 4: Create the 3 nodes of the simulation using the 'set' command.

STEP 5: Create a queue size between nodes.

STEP 6: Create TCP agent for the nodes and attach these agents to the nodes.

STEP 7: The traffic generator used is TCP and measured in terms of FTP0 and FTP1.

STEP 8: Create data transmission and node movements.

STEP 9: Schedule the simulation for 250 milisec.

PROGRAM:

```
set ns [new Simulator]

set tf [open p5.tr w]
$ns trace-all $tf

set nf [open p5.nam w]
$ns namtrace-all-wireless $nf 1000 1000

set topo [new Topography]
$topo load_flatgrid 1000 1000

$ns node-config -adhocRouting DSDV \
-llType LL \
-macType Mac/802_11 \
-ifqType Queue/DropTail \
-ifqLen 50 \
-phyType Phy/WirelessPhy \
-channelType Channel/WirelessChannel \
-propType Propagation/TwoRayGround \
-antType Antenna/OmniAntenna \
-topoInstance $topo \
-agentTrace ON \
-routerTrace ON

create-god 3
set n0 [$ns node]
set n1 [$ns node]
```

```
set n2 [$ns node]

$n0 label "tcp0"
$n1 label "sink1/tcp1"
$n2 label "sink2"

$ns initial_node_pos $n0 50
$ns initial_node_pos $n1 50
$ns initial_node_pos $n2 50

#The below code is used to give the initial node positions.
$n0 set X_ 50
$n0 set Y_ 50

$n1 set X_ 200
$n1 set Y_ 200

$n2 set X_ 700
$n2 set Y_ 700

$ns at 0.1 "$n0 setdest 50 50 15"
$ns at 0.1 "$n1 setdest 200 200 25"
$ns at 0.1 "$n2 setdest 700 700 25"

set tcp0 [new Agent/TCP]
set ftp0 [new Application/FTP]
set sink1 [new Agent/TCPSink]

set tcp1 [new Agent/TCP]
set ftp1 [new Application/FTP]
set sink2 [new Agent/TCPSink]

$ns attach-agent $n0 $tcp0
$ftp0 attach-agent $tcp0
$ns attach-agent $n1 $sink1
$ns connect $tcp0 $sink1
$ns attach-agent $n1 $tcp1
$ftp1 attach-agent $tcp1
$ns attach-agent $n2 $sink2
$ns connect $tcp1 $sink2

$ns at 0.1 "$ftp0 start"
$ns at 0.1 "$ftp1 start"

#The below code is used to provide the node movements.
$ns at 100 "$n1 setdest 550 550 15"
$ns at 190 "$n1 setdest 100 100 15"

proc finish {} {
global ns nf tf
$ns flush-trace
exec nam p5.nam &
```

```
close $tf
exit 0
}

$ns at 250 "finish"
$ns run
```

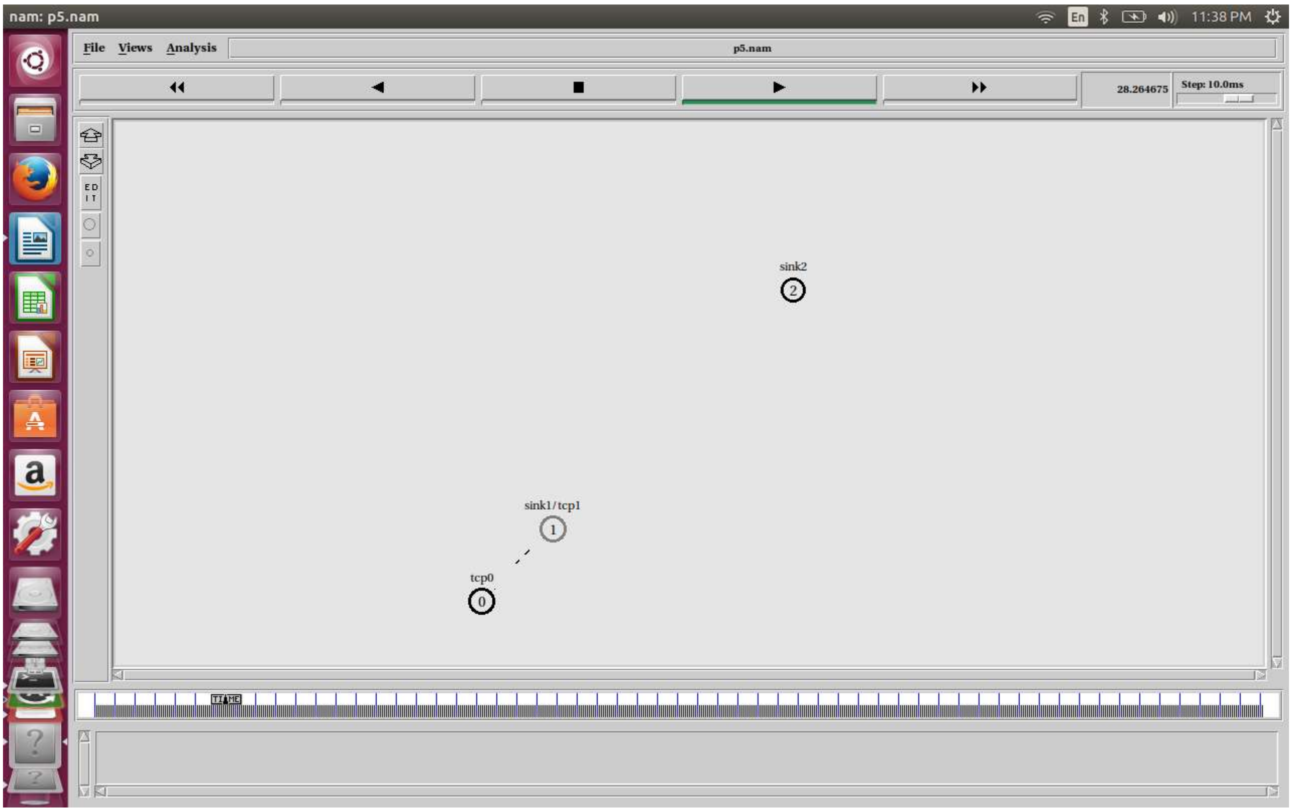
AWK SCRIPT:

```
BEGIN{
count1=0;
count2=0;
pack1=0;
pack2=0;
time1=0;
time2=0;
}
{
if($1=="r" && $3=="_1_" && $4=="AGT")
{
count1++;
pack1=pack1+$8;
time1=$2;
}
if($1=="r" && $3=="_2_" && $4=="RTR")
{
count2++;
pack2=pack2+$8;
time2=$2;
}
}
END{
printf("The Throughput from n0 to n1: %fMbps\n",((count1*pack1*8)/(time1*1000000)));
printf("The Throughput from n1 to n2: %fMbps\n",((count2*pack2*8)/(time2*1000000)));
}
```

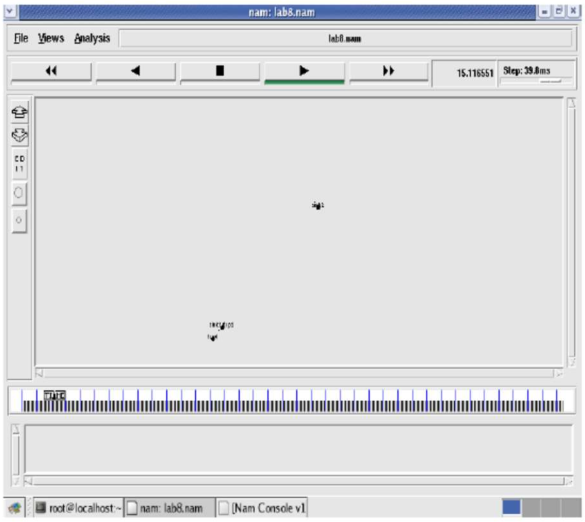
OUTPUT:

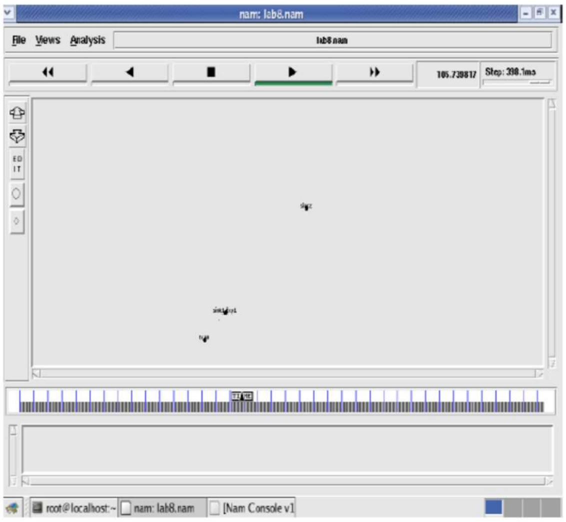
The Throughput from n0 to n1:_______Mbps
The Throughput from n1 to n2:_______Mbps

TOPOLOGY AND NAM RESULT:



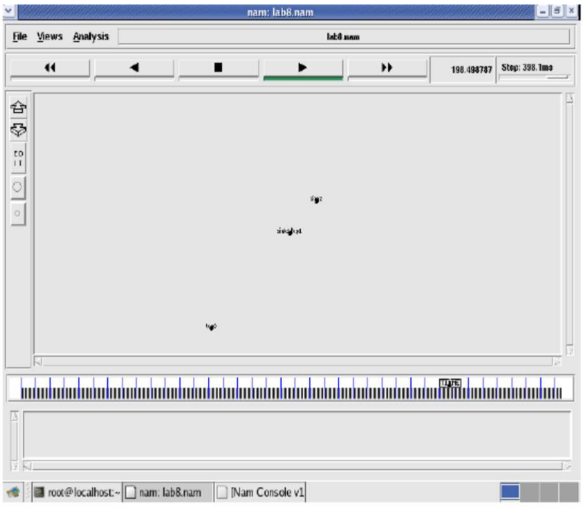
Packet transmission by tcp from n0 to n1

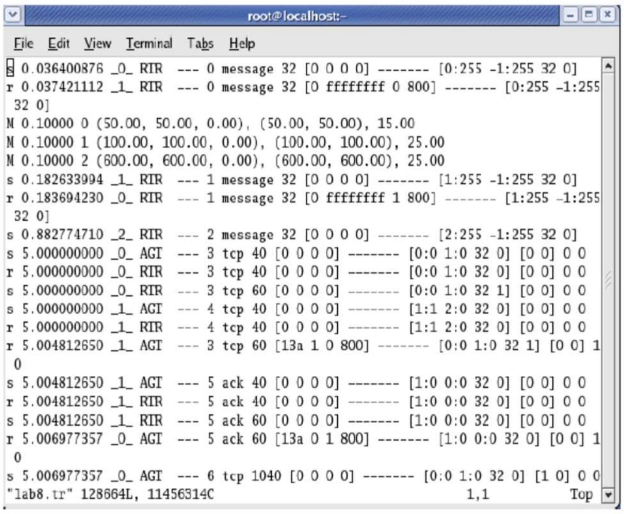




Node 1 and 2 are communicating

Node 2 is moving towards node 3





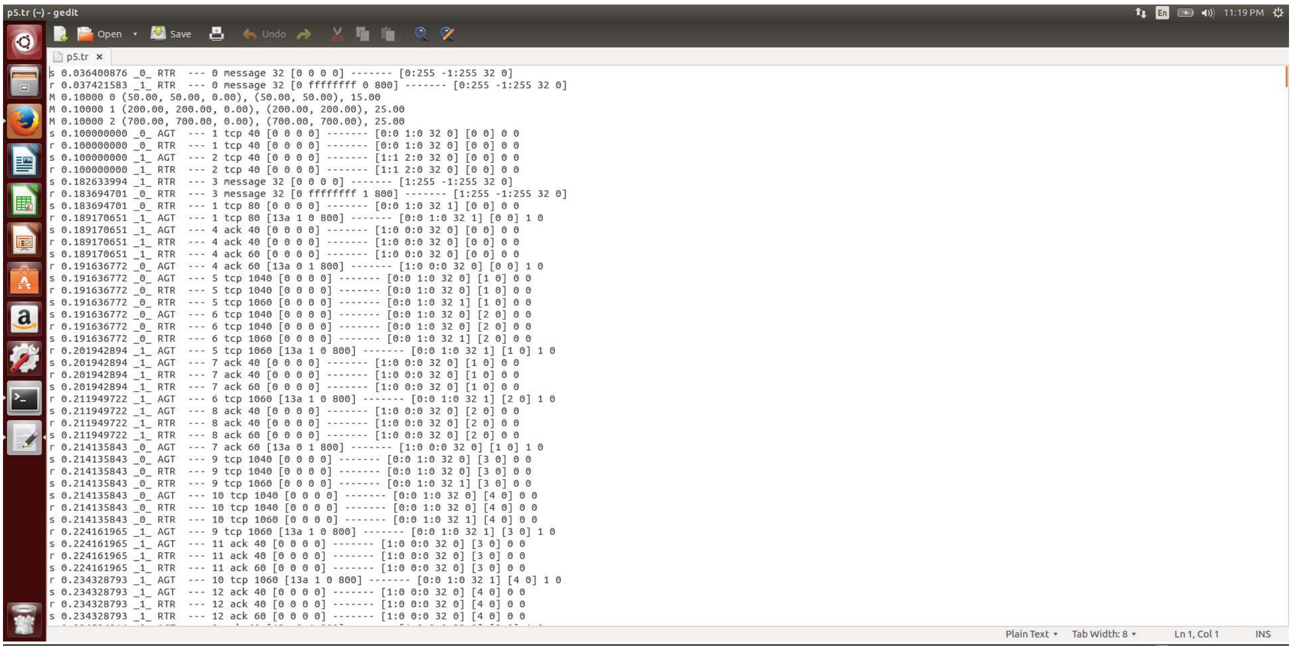
```
s 0.036400876 _0_ RTR  --- 0 message 32 [0 0 0 0] ------- [0:255 -1:255 32 0]
r 0.037421112 _1_ RTR  --- 0 message 32 [0 ffffffff 0 800] ------- [0:255 -1:255
 32 0]
M 0.10000 0 (50.00, 50.00, 0.00), (50.00, 50.00), 15.00
M 0.10000 1 (100.00, 100.00, 0.00), (100.00, 100.00), 25.00
M 0.10000 2 (600.00, 600.00, 0.00), (600.00, 600.00), 25.00
s 0.182633994 _1_ RTR  --- 1 message 32 [0 0 0 0] ------- [1:255 -1:255 32 0]
r 0.183694230 _0_ RTR  --- 1 message 32 [0 ffffffff 1 800] ------- [1:255 -1:255
 32 0]
s 0.882774710 _2_ RTR  --- 2 message 32 [0 0 0 0] ------- [2:255 -1:255 32 0]
s 5.000000000 _0_ AGT  --- 3 tcp 40 [0 0 0 0] ------- [0:0 1:0 32 0] [0 0] 0 0
r 5.000000000 _0_ RTR  --- 3 tcp 40 [0 0 0 0] ------- [0:0 1:0 32 0] [0 0] 0 0
s 5.000000000 _0_ RTR  --- 3 tcp 60 [0 0 0 0] ------- [0:0 1:0 32 1] [0 0] 0 0
s 5.000000000 _1_ AGT  --- 4 tcp 40 [0 0 0 0] ------- [1:1 2:0 32 0] [0 0] 0 0
r 5.000000000 _1_ RTR  --- 4 tcp 40 [0 0 0 0] ------- [1:1 2:0 32 0] [0 0] 0 0
r 5.004812650 _1_ AGT  --- 3 tcp 60 [13a 1 0 800] ------- [0:0 1:0 32 1] [0 0] 1
 0
s 5.004812650 _1_ AGT  --- 5 ack 40 [0 0 0 0] ------- [1:0 0:0 32 0] [0 0] 0 0
r 5.004812650 _1_ RTR  --- 5 ack 40 [0 0 0 0] ------- [1:0 0:0 32 0] [0 0] 0 0
s 5.004812650 _1_ RTR  --- 5 ack 60 [0 0 0 0] ------- [1:0 0:0 32 0] [0 0] 0 0
r 5.006977357 _0_ AGT  --- 5 ack 60 [13a 0 1 800] ------- [1:0 0:0 32 0] [0 0] 1
 0
s 5.006977357 _0_ AGT  --- 6 tcp 1040 [0 0 0 0] ------- [0:0 1:0 32 0] [1 0] 0 0
"lab8.tr" 128664L, 11456314C                          1,1           Top
```

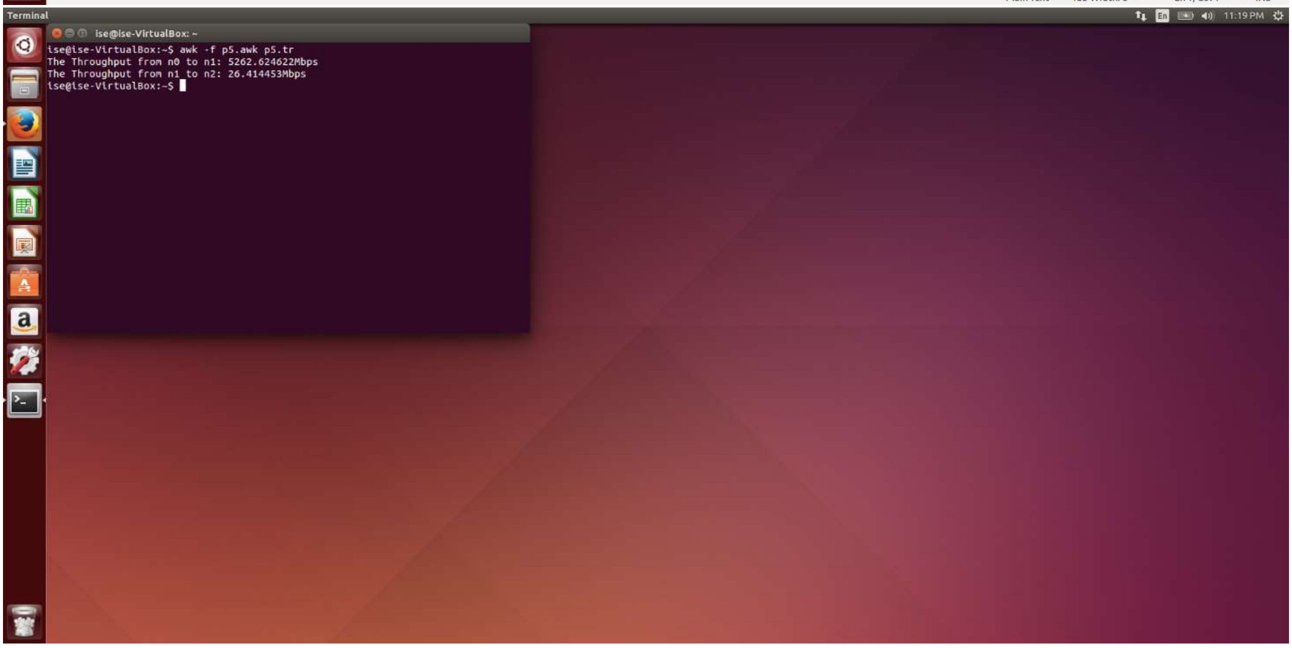
Node 2 is coming back from node 3 towards node1

Trace File

Here **“M”** indicates mobile nodes, **“AGT”** indicates Agent Trace, **“RTR”** indicates Route Trace

TRACE FILE AND AWK OUTPUTS:





PROGRAM 6:

IMPLEMENTATION OF LINK STATE ROUTING ALGORITHM

ALGORITHM:

STEP 1: Create the simulator object ns for designing the given simulation

STEP 2: Open the trace file and nam file in the write mode

STEP 3: Create the 12 nodes of the simulation using the 'set' command and duplex link between them

STEP 4: Create a queue size between nodes

STEP 5: Create UDP agent for the nodes and attach these agents to the nodes

STEP 6: The traffic generator used is UDP and measured in terms of cbr0 and cbr1

STEP 7: Configure node2 and node 3 as the null and attach it

STEP 8: Connect duplex links between the nodes

STEP 9: Create rtmodel and link between 5 and 11 is disconnected

STEP 10: Schedule the simulation for 50 milisec

#TO CREATE SIMULATOR OBJECT, TRACE FILE AND NAM FILE

```
set ns [new Simulator]
set nr [open p6.tr w]
$ns trace-all $nr
set nf [open p6.nam w]
$ns namtrace-all $nf

#Using routing protocol
$ns rtproto LS


#Creation of Nodes
for {set i 0} {$i < 12} {incr i} {
   set n($i) [$ns node]
   }

#Creation of links
$ns duplex-link $n(0) $n(1) 1Mb 10ms DropTail
$ns duplex-link $n(1) $n(2) 1Mb 10ms DropTail
$ns duplex-link $n(2) $n(3) 1Mb 10ms DropTail
$ns duplex-link $n(3) $n(4) 1Mb 10ms DropTail
$ns duplex-link $n(4) $n(5) 1Mb 10ms DropTail
$ns duplex-link $n(5) $n(6) 1Mb 10ms DropTail
$ns duplex-link $n(6) $n(7) 1Mb 10ms DropTail
```

```
$ns duplex-link $n(7) $n(8) 1Mb 10ms DropTail
$ns duplex-link $n(8) $n(0) 1Mb 10ms DropTail
$ns duplex-link $n(0) $n(9) 1Mb 10ms DropTail
$ns duplex-link $n(1) $n(10) 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 up UDP connection
set udp0 [new Agent/UDP]
set cbr0 [new Application/Traffic/CBR]
set null0 [new Agent/Null]
set udp1 [new Agent/UDP]
set cbr1 [new Application/Traffic/CBR]
set null1 [new Agent/Null]

$ns attach-agent $n(0) $udp0
$cbr0 attach-agent $udp0
$ns attach-agent $n(5) $null0
$ns connect $udp0 $null0


$ns attach-agent $n(1) $udp1
$cbr1 attach-agent $udp1
$ns attach-agent $n(5) $null1
$ns connect $udp1 $null1

#Set up CBR over UDP connection


$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005


$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005


#Schedule events for CBR and FTP agents

$ns at 0.1 "$cbr0 start"
$ns at 0.1 "$cbr1 start"

$ns rtmodel-at 10.0 down $n(11) $n(5)     ;# communication link between node 11
                                ;#to 5 disconnected
$ns rtmodel-at 30.0 up $n(11) $n(5)       ;# communication link between node 11
```

;#to 5 reconnected

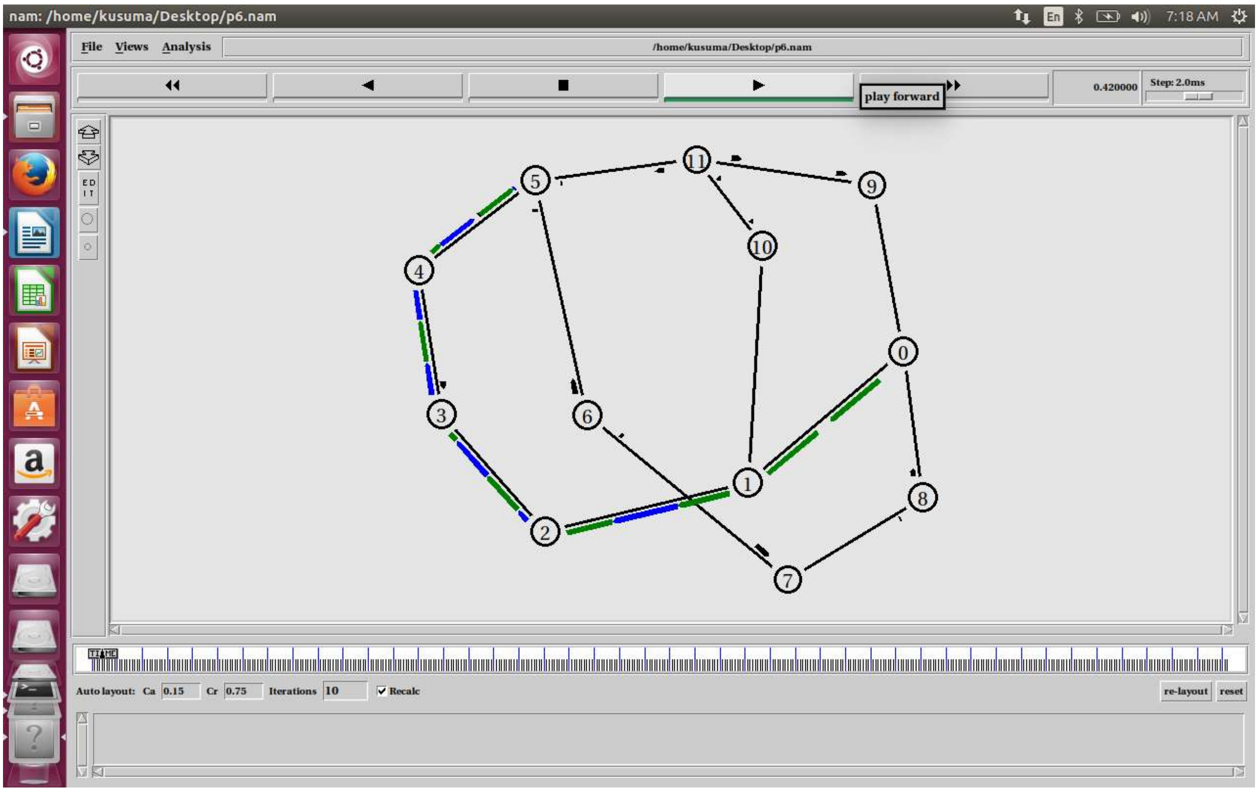
```
$ns rtmodel-at 15.0 down $n(7) $n(6)
$ns rtmodel-at 20.0 up $n(7) $n(6)

$ns color 1 "green"
$ns color 2 "blue"

$udp0 set class_ 1
$udp1  set class_ 2

#Define procedure to clean up
proc finish { } {
    global ns nr nf
    $ns flush-trace
    close $nf
    close $nr
    exec nam p6.nam &
    exit 0
}

$ns at 45.0 "$cbr0 stop"
$ns at 45.0 "$cbr1 stop"
$ns at 50.0 "finish"

#Run the simulation
$ns run
```

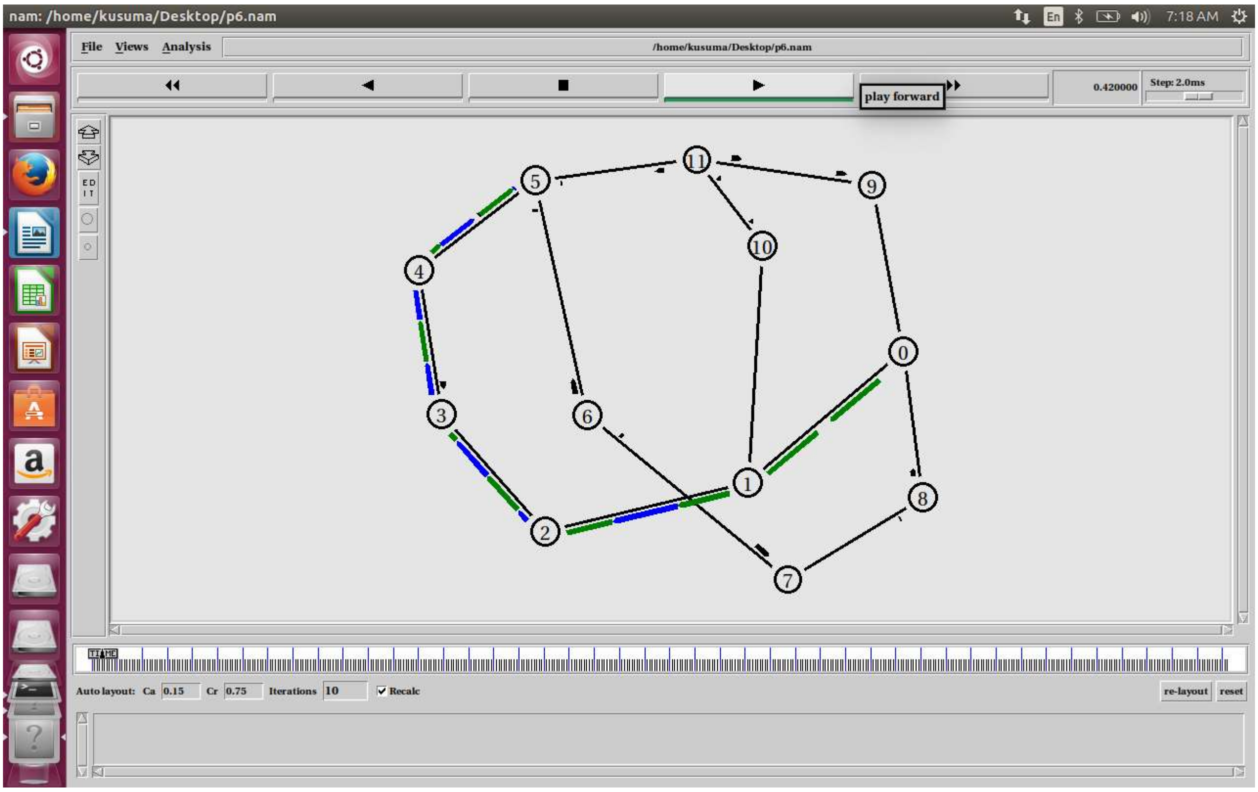
AWK SCRIPT:

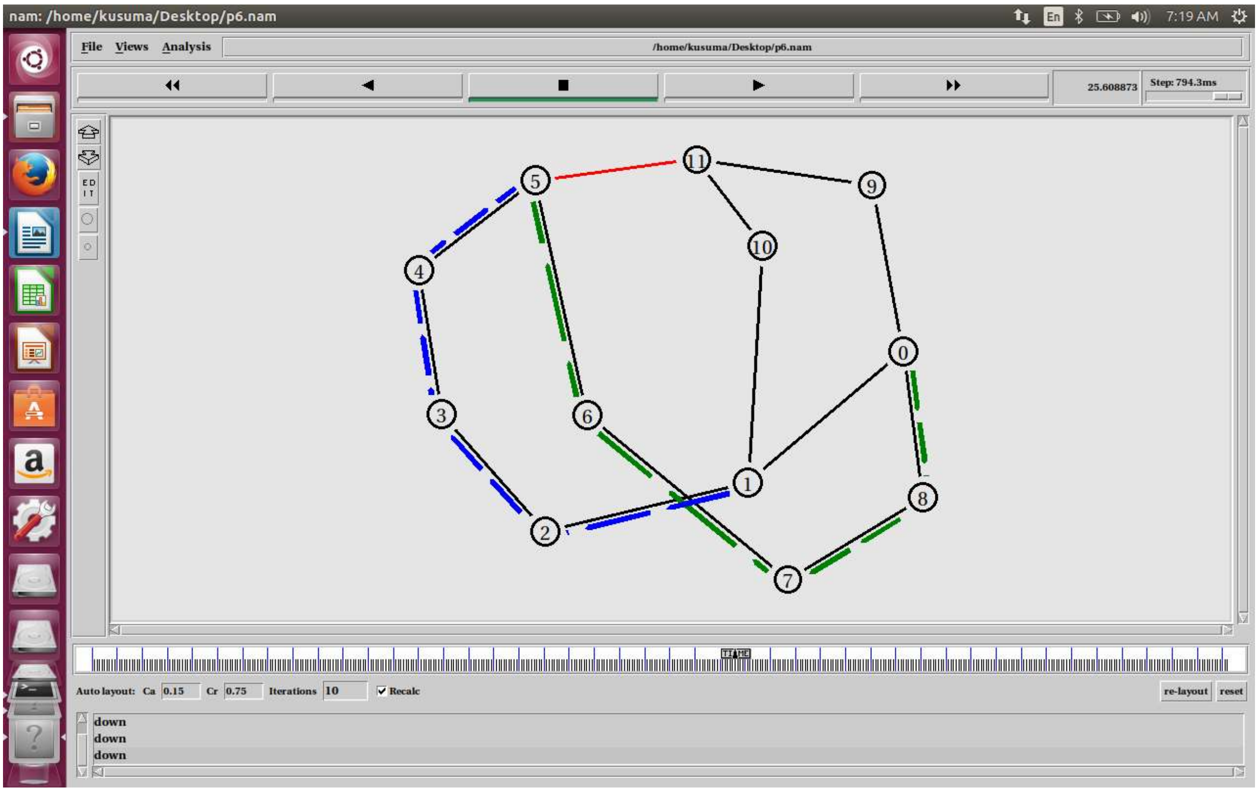
```
BEGIN {
pksend = 0
pkreceive = 0
pkdrop = 0
pkrouting = 0
}

#Executed for each line of input file thru.tr
{
#For udp packets
if ( $1=="+" && ($3=="0" || $3=="11") && $5=="cbr" )
{
pksend++;
}
if ( $1=="r" && $4=="5" && $5=="cbr" )
{
```

```
pkreceive++;
}
if ( $1=="d" )
{
pkdrop++;
}
if ( $1=="r" && ($5=="rtProtoDV" || $5=="rtProtoLS")  )
{
pkrouting++;
}
}
END {
print "No of send packets = " pksend
print "No of received packets = " pkreceive
print "No of dropped packets = " pkdrop
print "No of routing packets = " pkrouting
print "Normalized Overhead (routing/received), NOH = " pkrouting/pkreceive
print "Packet Delivery Ratio (received/send), PDR = " pkreceive/pksend
}
```

TOPOLOGY AND NAM:




nam: /home/kusuma/Desktop/p6.nam
7:18 AM
File Views Analysis
/home/kusuma/Desktop/p6.nam
play forward
0.420000
Step: 2.0ms
Auto layout: Ca 0.15 Cr 0.75 Iterations 10 Recalc
re-layout
reset


nam: /home/kusuma/Desktop/p6.nam
7:19 AM
File Views Analysis
/home/kusuma/Desktop/p6.nam
25.608873
Step: 794.3ms
Auto layout: Ca 0.15 Cr 0.75 Iterations 10 Recalc
re-layout
reset
down
down
down

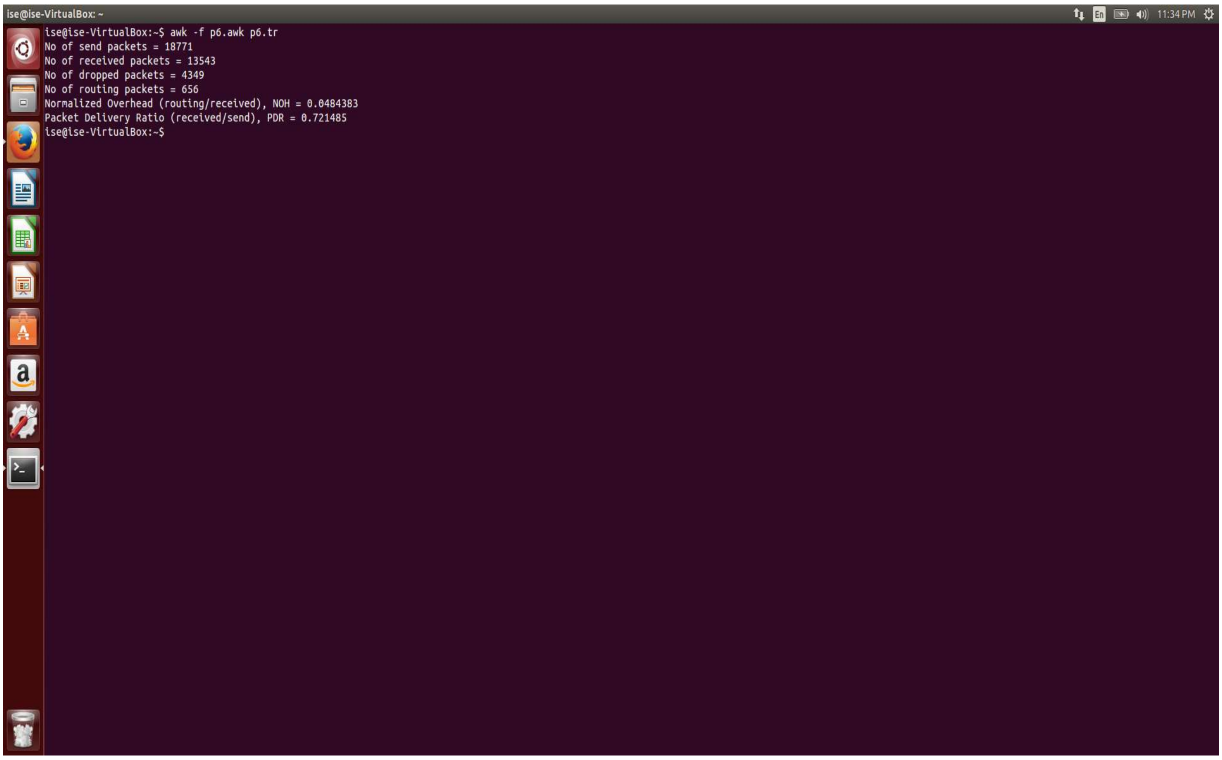
OUTPUT:

No of Send Packets= 18959

No of Received Packets =13543

No of Dropped Packets= 4349

No of Routing Packets=656



PROBABLE/SUGGESTED QUESTION BANK

1. What are functions of different layers in OSI model?

2. Differentiate between TCP/IP Layers and OSI Layers.

3. Why header is required?

4. What is the use of adding header and trailer to frames?

5. What is encapsulation?

6. Why fragmentation is required?

7. Name the protocols in the TCP/IP protocol suite?

8. What is bit stuffing and its applications?

9. What is byte stuffing and its applications?

10. Differentiate between flow control and congestion control.

10. Differentiate between Point-to-Point Connection and End-to-End connections.

11. What is HDLC protocol?

12. Explain the frame format of I, U and S frames in detail.

13. What is CRC? Explain in detail the procedure to find CRC? In which CRC is used?

13. Differentiate between TCP and UDP.

14. What is Sliding window protocol? Name the two protocols comes under Sliding window protocol.

15. Explain different types of routing protocols?

16. Explain the difference between Distance vector and Link state algorithm.

17. Explain the procedure to calculate the shortest path in Dijkstra's algorithm.

18. Explain different types of Congestion control algorithms?

19. Explain the difference between Leaky bucket and token bucket algorithm in detail.

20. Explain different types of Encryption and Decryption algorithms in detail.

21. Explain the procedure to calculate the Encryption and Decryption procedure in Substitution method.

22. Explain the procedure to calculate the Encryption and Decryption procedure in Transposition method.

23. What are the other error detection algorithms?

29. What are drawbacks in distance vector algorithm?
30. What is cryptography?
31. How do you classify cryptographic algorithms?
32. What are public key and private key?
33. What is NS2?
34. Explain the architecture of NS2.
35. Explain the frame format of Trace file for wired node in detail.
36. Which are scripts used in NS2?
37. What is awk?
36. What is an agent and traffic descriptors?
37. Name the agents and its associated traffic descriptors used in NS2 in detail.
38. For TCP and UDP the destination nodes are called as?
39. What are the applications of TCP and UDP?
40. What is an Ethernet LAN? Which protocol is used?
41. What is an error rate and data rate? Explain in detail.
42. How the throughput is calculated when the error rate and data rates are changed.
43. What is congestion window?
44. If the congestion window is not set what will happen?
45. Which are commands used to check the particular node congestion?
46. Explain the congestion window graph in detail.
47. What is ESS and BSS?
48. Explain the frame format of Trace file for wireless nodes in detail.
49. Which protocol is used in Wireless LAN?
50. How the throughput is calculated in the Wireless LAN?

Error Model

The error models, introduces packet losses into a simulation.

In addition to the basic class ErrorModel described in detail below, there are several other types of error modules not being completely documented yet, which include:

- ErrorModel/Trace: error model that reads a loss trace (instead of a math/computed model)
- MrouteErrorModel: error model for multicast routing, now inherits from trace.
- ErrorModel/Periodic: models periodic packet drops (drop every nth packet we see). This model can be conveniently combined with a flow-based classifier to achieve drops in particular flows
- SelectErrorModel: for Selective packet drop.
- ErrorModel/TwoStateMarkov, ErrorModel/Expo, ErrorModel/Empirical: inerit from ErrorModel/TwoState.
- ErrorModel/List: specify a list of packets/bytes to drop, which could be in any order.

Their definitions can be found in ~*ns*/queue/errmodel.{cc, h} and ~*ns*/tcl/lib/ns-errmodel.tcl, ns-default.tcl.

NETWORK THROUGHPUT:

It refers to the average data rate of successful data or message delivery over a specific communications link. Network throughput is measured in Mega bits per second (bps).

Step 1: Convert from bytes to bits: To convert the window size to bits, multiply the number of bytes by eight. 64 KB x 8 = 524,288 bits.

Step 2: Divide the converted bits by the network path latency. For this example, if the latency is 60 milliseconds. 524,288 bits / .060 seconds = 8,738,133 bits per second.

Step 3: Convert the result from step 2 to megabits per second by dividing the result by 1,000,000. In this example, the maximum throughput is 8.738 Mbps.