EAST POINT COLLEGE OF ENGINEERING & TECHNOLOGY

Department of Computer Science and Engineering

Jnanaprabha, Virgo Nagar Post, Bengaluru-560049

Academic Year: 2024-25

LABORATORY MANUAL

SEMESTER : V

SUBJECT : COMPUTER NETWORKS LABORATORY

SUBCODE : BCS502

NAME : _____

USN : _____

SECTION: _____

PROGRAM OUTCOMES

PO1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem analysis: Identify, formulate, review research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activitieswith an understanding of the limitations.

PO6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and Team Work: Function effectively as an individual and as a member or leader in diverse teams, and in multi – disciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Project management and finance: Demonstrate knowledge and understanding of the Engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-long learning: Recognize the need for and have the preparation and ability to engage in independent and life -long learning in the broadest context of technological change.

EAST POINT COLLEGE OF ENGINEERING & TECHNOLOGY

Department of Computer Science and Engineering

INSTITUTE VISION AND MISSION

VISION

The East Point College of Engineering and Technology aspires to be a globally acclaimed institution, recognized for excellence in engineering education, applied research and nurturing students for holistic development.

MISSION

M1: To create engineering graduates through quality education and to nurtureinnovation, creativity and excellence in teaching, learning and research

M2: To serve the technical, scientific, economic and societal developmental needs of our communities

M3: To induce integrity, teamwork, critical thinking, personality development andethics in students and to lay the foundation for lifelong learning

EAST POINT COLLEGE OF ENGINEERING & TECHNOLOGY
Department of Computer Science & Engineering
Approved by AICTE New Delhi | Affiliated to VTU, Belagavi,
Virgo Nagar, Bengaluru - 560049

DEPARTMENT VISION AND MISSION

VISION

The department aspires to be a Centre of excellence in Computer Science & Engineering to develop competent professionals through holistic development.

MISSION

M1: To create successful Computer Science Engineering graduates through effective pedagogies, the latest tools and technologies, and excellence in teaching and learning.

M2: To augment experiential learning skills to serve technical, scientific, economic, and social developmental needs.

M3: To instil integrity, critical thinking, personality development, and ethics in students for a successful career in Industries, Research, and Entrepreneurship.

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

PEO 1: To produce graduates who can perform technical roles to contribute effectively in software industries and R&D Centre

PEO 2: To produce graduates having the ability to adapt and contribute in key domains of computer science and engineering to develop competent solutions.

PEO 3: To produce graduates who can provide socially and ethically responsible solutions while adapting to new trends in the domain to carve a successful career in the industry

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO1: To conceptualize, model, design, simulate, analyse, develop, test, and validate computing systems and solve technical problems arising in the field of computer science & engineering.

PSO2: To specialize in the sub-areas of computer science & engineering systems such as cloud computing, Robotic Process Automation, cyber security, big data analytics, user interface design, and IOT to meet industry requirements.

PSO3: To build innovative solutions to meet the demands of the industry using appropriate tools and techniques.

COURSE LEARNING OBJECTIVES

CLO 1. Study the TCP/IP protocol suite, switching criteria and Medium Access Control protocols for reliable and noisy channels.
CLO 2. Learn network layer services and IP versions.
CLO 3. Discuss transport layer services and understand UDP and TCP protocols.
CLO 4. Demonstrate the working of different concepts of networking layers and protocols.

COURSE OUTCOMES

At the end of the course the student will be able to:
CO1. Learn the basic needs of communication system.
CO2. Explain the fundamentals of computer networks.
CO3. Apply the concepts of computer networks to demonstrate the working of various layers and protocols in communication network.
CO4. Analyze the principles of protocol layering in modern communication systems.
CO5: Demonstrate various Routing protocols and their services using tools such as Cisco packet tracer.

Computer Networks Laboratory			
Course Code	**BCS502**	CIE Marks	20
Teaching Hours/Week	2		
Total Hours of Pedagogy	20P		

Program List:

1	Implement three nodes point − to − point network with duplex links between them. Set the queue size, vary the bandwidth, and find the number of packets dropped.
2	Implement transmission of ping messages/trace route over a network topology consisting of 6 nodes and find the number of packets dropped due to congestion.
3	Implement an Ethernet LAN using n nodes and set multiple traffic nodes and plot congestion window for different source / destination.
4	Develop a program for error detecting code using CRC-CCITT (16- bits).
5	Develop a program to implement a sliding window protocol in the data link layer.
6	Develop a program to find the shortest path between vertices using the Bellman-Ford and path vector routing algorithm.
7	Using TCP/IP sockets, write a client − server program to make the client send the file name and to make the server send back the contents of the requested file if present.
8	Develop a program on a datagram socket for client/server to display the messages on client side, typed at the server side.
9	Develop a program for a simple RSA algorithm to encrypt and decrypt the data.
10	Develop a program for congestion control using a leaky bucket algorithm.

Index					
Sl. No.	**Program List**	**CO**	**PO**	**RBT**	**Page No**
1	Implement three nodes point − to − point network with duplex links between them. Set the queue size, vary the bandwidth, and find the number of packets dropped.	CO1	PO1, PO2, PO3, PO5, PO12	L3	
2	Implement transmission of ping messages/trace route over a network topology consisting of 6 nodes and find the number of packets dropped due to congestion.	CO1	PO1, PO2, PO3, PO5, PO12	L3	
3	Implement an Ethernet LAN using n nodes and set multiple traffic nodes and plot congestion window for different source / destination.	CO3	PO1, PO2, PO5, PO12	L2	
4	Develop a program for error detecting code using CRC-CCITT (16- bits).	CO1	PO1, PO2, PO3, PO5, PO12	L3	
5	Develop a program to implement a sliding window protocol in the data link layer.	CO2	PO1, PO2, PO5, PO12	L2	
6	Develop a program to find the shortest path between vertices using the Bellman-Ford and path vector routing algorithm.	CO1	PO1, PO2, PO3, PO5, PO12	L3	
7	Using TCP/IP sockets, write a client − server program to make the client send the file name and to make the server send back the contents of the requested file if present.	CO2	PO1, PO2, PO5, PO12	L2	
8	Develop a program on a datagram socket for client/server to display the messages on client side, typed at the server side.	CO2	PO1, PO2, PO3, PO5, PO12	L3	
9	Develop a program for a simple RSA algorithm to encrypt and decrypt the data.	CO2	PO1, PO2, PO3,	L3	

			PO5, PO12		
10	Develop a program for congestion control using a leaky bucket algorithm.	CO2	PO1, PO2, PO3, PO5, PO12	L3	

Course Articulation Matrix

COs	POs												PSOs		
	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	1	2	2	-	1	-	-	-	-	-	-	1	3	-	-
CO2	1	2	2	-	1	-	-	-	-	-	-	1	3	-	-
CO3	1	1	-	-	1	-	-	-	-	-	-	1	3	-	-
CO4	1	1	-	-	1	-	-	-	-	-	-	1	3	-	-
CO5	1	1	-	-	1	-	-	-	-	-	-	1	3	-	-

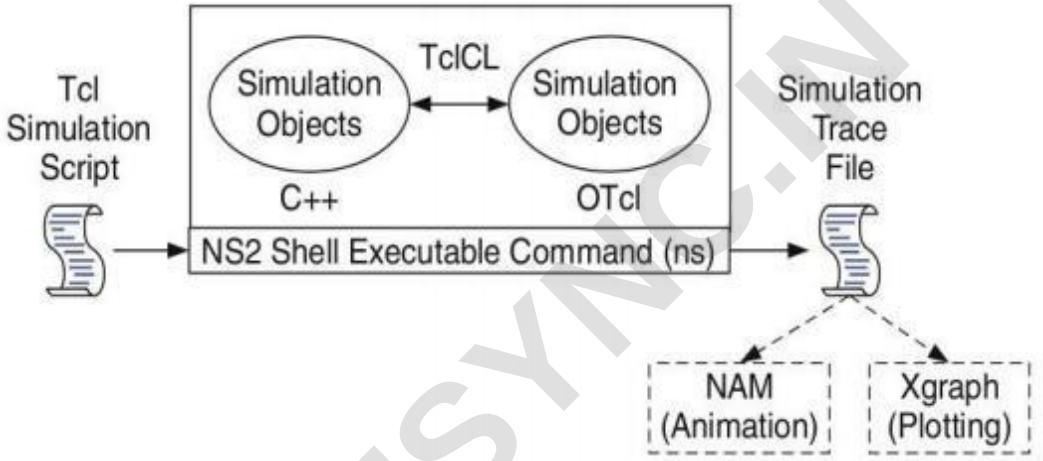
3 - High Correlation 2 - Medium Correlation 1 – Low Correlation

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



Tcl scripting

- Tcl is a general-purpose scripting language. [Interpreter]
- Tcl runs on most of the platforms such as Unix, Windows, and Mac.
- The strength of Tcl is its simplicity.
- It is not necessary to declare a data type for variable prior to the usage.

Basics of TCL

Syntax: command arg1 arg2 arg3

•Hello World!

puts stdout{Hello, World!}

Hello, World!

Variables Command Substitution

set a 5 set len [string length foobar]

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

• Simple Arithmetic

expr 7.2 / 4

• Procedures

proc Diag {a b} {

set c [expr sqrt($a * $a + $b * $b)]

return $c }

puts –Diagonal of a 3, 4 right triangle is [Diag 3 4]

Output: Diagonal of a 3, 4 right triangle is 5.0

set ns [new Simulator]

• **Loops**

while{$i < $n} { for {set i 0} {$i < $n} {incr i} {

. . . . . .

} }

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

1. An ns simulation starts with the command

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

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

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

#Open the Trace file

```
set tracefile1 [open out.tr w]
$ns trace-all $tracefile1
```

#Open the NAM trace file

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

The above creates a data 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.

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|.

```
set tracefile1 [open out.tr w]
$ns trace-all $tracefile1
set namfile [open out.nam w]
$ns namtrace-all $namfile
$ns run
set n0 [$ns node]
$ns duplex-link $n0 $n2 10Mb 10ms DropTail
```

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 is an 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

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

The way to define a node is

We created a node that 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

Proc finish { } {

global ns tracefile1 namfile

$ns flush-trace Close $tracefile1

Close $namfile

Exec nam out.nam & Exit 0

$ns at 125.0 –finish‖

#set Queue Size of link (n0-n2) to 20

$ns queue-limit $n0 $n2 20 set tcp [new Agent/TCP]

set sink [new Agent /TCPSink]

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 Discard) 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.

```
set sink [new Agent /TCPSink]
```

Defines the behaviour 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
```

set cbr [new Application/Traffic/CBR]

$cbr attach-agent $udp

$cbr set packetsize_ 100

$cbr set rate_ 0.01Mb

```
$cbr set random_ false

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
$ns at <time> <event>
```

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 at <time> <event>
```

NS is a discrete event based simulation. The tcp script defines when event should occur fig shown below, the meaning of the fields are:

Event	Time	From Node	To Node	PKT Type	PKT Size	Flags	Fid	Src Addr	Dest Addr	Seq Num	Pkt id

Event Time From

1. The first field is the event type. It is given by one of four possible symbols r, +, -, d which correspond respectively to receive (at the output of the link), enqueued, dequeued and dropped.

```
cbr set packetSize_ <packet size>
$cbr set interval_ 0.005
$ns at 0.1 –$cbr start
$ns at 1.0 — $ftp start
$ns at 124.0 –$ftp stop
```

2. The second field gives the time at which the event occurs.
3. Gives the input node of the link at which the event occurs.
4. Gives the output node of the link at which the event occurs
5. Gives the packet type (eg CBR or TCP)
6. Gives the packet size
7. Some flags
8. 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. This is the source address given in the form of –node.port.
10. This is the destination address, given in the same form.
11. 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. 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‖.

Awk- An Advanced

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. Xgraph [options] file-name

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 {count =count+1

printf—%3f%20s %-12s %d\n‖, count, $2,$3,$6 }‘ empn.lst THE –f OPTION: STORING awk

PROGRAMS INA 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 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:

Awk –F|| –f empawk.awk empn.lst

awk option _selection_criteria {action}' file(s) 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

The FILENAME Variable: FILENAME stores the name of the current file being processed. Like grep and sed, awk can also handle multiple filenames in the command line. By default, awk doesn't print the filename, but you can instruct it to do so:

_$6<4000 {print FILENAME, $0 }'

With FILENAME, you can device logic that does different things depending on the file that is processed.

NS2 Installation

• NS2 is a free simulation tool.

• It runs on various platforms including UNIX (or Linux), Windows, and Mac systems.

• NS2 source codes are distributed in two forms: the all-in-one suite and the component-wise.

• _all-in-one' package provides an –install| script which configures the NS2 environment and creates NS2 executable file using the –make| utility.

NS-2 installation steps in Linux

➢ Go to Computer File System now paste the zip file –ns-allinone-2.34.tar.gz| into

opt folder.

➢ Now unzip the file by typing the following command

[root@localhost opt] # tar -xzvf ns-allinone-2.34.tar.gz

➢ After the files get extracted, we get ns-allinone-2.34 folder as well as zip file ns-allinone-2.34.tar.gz

[root@localhost opt] # ns-allinone-2.34 ns-allinone-2.34.tar.gz

➢ Now go to ns-allinone-2.33 folder and install it

[root@localhost opt] # cd ns-allinone-2.34

[root@localhost ns-allinone-2.33] # ./install

➢ Once the installation is completed successfully we get certain pathnames in that terminal which must be pasted in –.bash_profile‖ file.

➢ First minimize the terminal where installation is done and open a new terminal and open the file –.bash_profile‖

[root@localhost ~] # vi .bash_profile

➢ When we open this file, we get a line in that file which is shown below

PATH=$PATH:$HOME/bin

To this line we must paste the path which is present in the previous terminal where ns was installed. First put —:‖ then paste the path in-front of bin. That path is shown below.

–:/opt/ns-allinone-2.33/bin:/opt/ns-allinone-2.33/tcl8.4.18/unix:/opt/ns-allinone-2.33/tk8.4.18/unix‖.

In the next line type –LD_LIBRARY_PATH=$LD_LIBRARY_PATH:‖ and paste the two paths separated by –:‖ which are present in the previous terminal i.e Important notices section (1)

–/opt/ns-allinone-2.33/otcl-1.13:/opt/ns-allinone-2.33/lib‖

➢ In the next line type –TCL_LIBRARY=$TCL_LIBRARY:‖ and paste the path which is present in previous terminal i.e Important Notices section (2)

–/opt/ns-allinone-2.33/tcl8.4.18/library‖

➢ In the next line type –export LD_LIBRARY_PATH‖

➢ In the next line type –export TCL_LIBRARY‖

➢ The next two lines are already present the file –export PATH‖ and –unset USERNAME‖

➢ Save the program (ESC + shift : wq and press enter)

➢ Now in the terminal where we have opened .bash_profile file, type the following command

to check if path is updated correctly or not

[root@localhost ~] # vi .bash_profile

[root@localhost ~] # source .bash_profile

➢ If path is updated properly, then we will get the prompt as shown below

[root@localhost ~] #

➢ Now open the previous terminal where you have installed ns

[root@localhost ns-allinone-2.33] #

➢ Here we need to configure three packages –ns-2.33‖, –nam-1.13‖ and –xgraph-12.1‖

➢ First, configure –ns-2.33‖ package as shown below

[root@localhost ns-allinone-2.33] # cd ns-2.33

[root@localhost ns-2.33] # ./configure

[root@localhost ns-2.33] # make clean

[root@localhost ns-2.33] # make

[root@localhost ns-2.33] # make install

[root@localhost ns-2.33] # ns

%

➢ If we get –%‖ symbol it indicates that ns-2.33 configuration was successful.

➢ Second, configure –nam-1.13‖ package as shown below

[root@localhost ns-2.33] # cd . .

[root@localhost ns-allinone-2.33] # cd nam-1.13

[root@localhost nam-1.13] # ./configure

[root@localhost nam-1.13] # make clean

[root@localhost nam-1.13] # make

[root@localhost nam-1.13] # make install

[root@localhost nam-1.13] # ns

%

➢ If we get –%‖ symbol it indicates that nam-1.13 configuration was successful.

➢ Third, configure –xgraph-12.1‖ package as shown below

[root@localhost nam-1.13] # cd . .

[root@localhost ns-allinone-2.33] # cd xgraph-12.1

[root@localhost xgraph-12.1] # ./configure

[root@localhost xgraph-12.1] # make clean

[root@localhost xgraph-12.1] # make

[root@localhost xgraph-12.1] # make install

[root@localhost xgraph-12.1] # ns

%

This completes the installation process of –NS-2‖ simulator.

1. Implement three nodes point – to – point network with duplex links between them. Set the queue size, vary the bandwidth and find the number of packets dropped.

```
set ns [new Simulator] /* Letter S is capital */
set nf [open lab1.nam w] /* open a nam trace file in write mode */
$ns namtrace-all $nf /* nf – nam file */
set tf [open lab1.tr w] /* tf- trace file */
$ns trace-all $tf
proc finish { } { /* provide space b/w proc and finish and all are in small case */
global ns nf tf
$ns flush-trace /* clears trace file contents */
close $nf
close $tf
exec nam lab1.nam &
exit 0
}
set n0 [$ns node] /* creates 4 nodes */
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
$ns duplex-link $n0 $n2 200Mb 10ms DropTail /*Letter M is capital Mb*/
$ns duplex-link $n1 $n2 100Mb 5ms DropTail /*D and T are capital*/
$ns duplex-link $n2 $n3 1Mb 1000ms DropTail
$ns queue-limit $n0 $n2 10
$ns queue-limit $n1 $n2 10
set udp0 [new Agent/UDP] /* Letters A,U,D and P are capital */
$ns attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR] /* A,T,C,B and R are capital*/
$cbr0 set packetSize_ 500 /*S is capital, space after underscore*/
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0
set udp1 [new Agent/UDP]
```

```
$ns attach-agent $n1 $udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp1
set udp2 [new Agent/UDP]
$ns attach-agent $n2 $udp2
set cbr2 [new Application/Traffic/CBR]
$cbr2 attach-agent $udp2
set null0 [new Agent/Null] /* A and N are capital */
$ns attach-agent $n3 $null0
$ns connect $udp0 $null0
$ns connect $udp1 $null0
$ns at 0.1 "$cbr0 start"
$ns at 0.2 "$cbr1 start"
$ns at 1.0 "finish"
$ns run
```

AWK file (Open a new editor using "vi command" and write awk file and save with

".awk" extension)

```
/*immediately after BEGIN should open braces „{„
BEGIN { c=0;
}
{
if ($1= ="d")
{
c++;
printf("%s\t%s\n",$5,$11);
}
}
```

/*immediately after END should open braces „{„

```
END{
printf("The number of packets dropped =%d\n",c);
}
```

Steps for execution

1) Open vi editor and type program. Program name should have the extension ― .tcl ‖
[root@localhost ~]# vi lab1.tcl

2) Save the program by pressing –ESC key‖ first, followed by –Shift and :‖ keys simultaneously and type ―wq‖ and press Enter key.

3) Open vi editor and type awk program. Program name should have the extension –.awk ‖
[root@localhost ~]# vi lab1.awk

4) Save the program by pressing –ESC key‖ first, followed by –Shift and :‖ keys simultaneously and type ―wq‖ and press Enter key.

5) Run the simulation program
[root@localhost~]# ns lab1.tcl

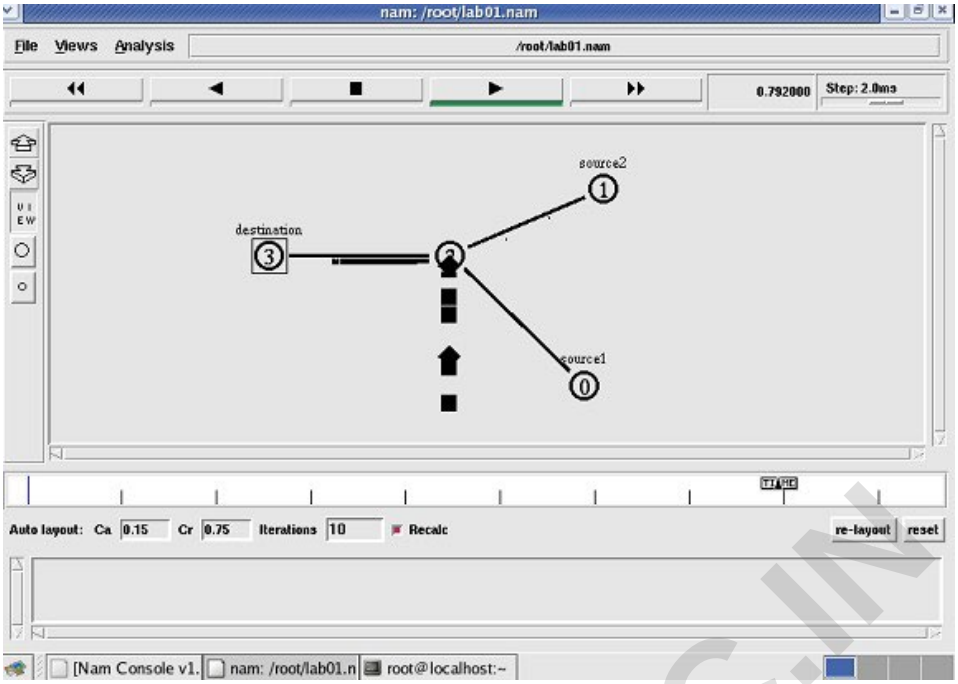
Here –ns‖ indicates network simulator. We get the topology shown in the snapshot.
ii) Now press the play button in the simulation window and the simulation will begins.

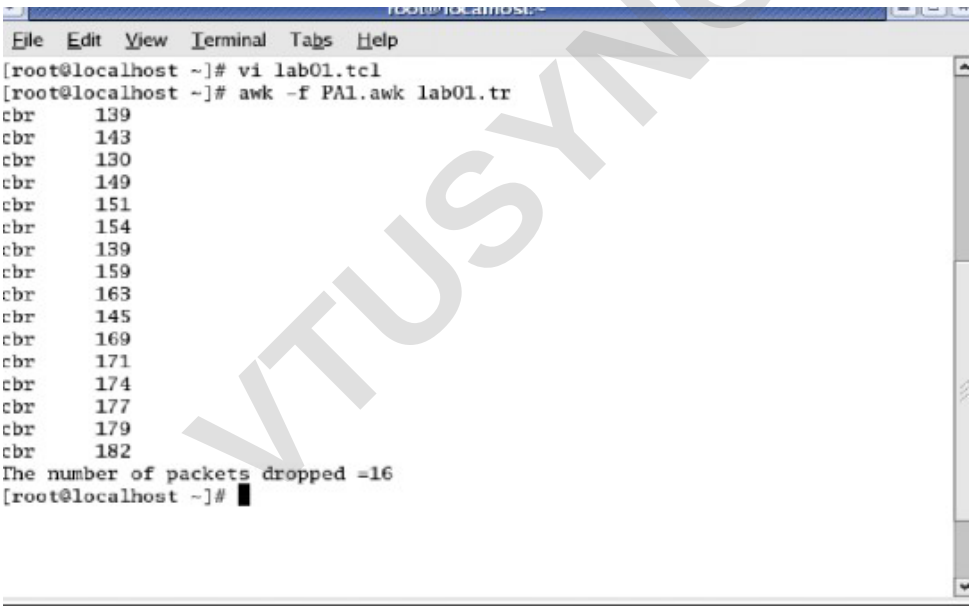
6) After simulation is completed run awk file to see the output ,
[root@localhost~]# awk –f lab1.awk lab1.tr

7) To see the trace file contents open the file as ,
[root@localhost~]# vi lab1.tr

Trace file contains 12 columns:-Event type, Event time, From Node, Source Node, Packet Type, Packet Size, Flags
(indicated by --------), Flow ID, Source address, Destination address, Sequence ID, Packet ID

OUTPUT:





Note:

1. Set the queue size fixed from n0 to n2 as 10, n1-n2 to 10 and from n2-n3 as 5. Syntax: To set the queue size

$ns set queue-limit <from> <to> <size> Eg:

$ns set queue-limit $n0 $n2 10

Go on varying the bandwidth from 10, 20 30 . . and find the number of packets dropped at the node 2

2. Implement transmission of ping messages/trace route over a network topology consisting of 6 nodes and find the number of packets dropped due to congestion in the network.

```
set ns [ new Simulator ]
set nf [ open lab2.nam w ]
$ns namtrace-all $nf
set tf [ open lab2.tr w ]
$ns trace-all $tf
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
$n4 shape box
$ns duplex-link $n0 $n4 1005Mb 1ms DropTail
$ns duplex-link $n1 $n4 50Mb 1ms DropTail
$ns duplex-link $n2 $n4 2000Mb 1ms DropTail
$ns duplex-link $n3 $n4 200Mb 1ms DropTail
$ns duplex-link $n4 $n5 1Mb 1ms DropTail
set p1 [new Agent/Ping]
$ns attach-agent $n0 $p1
$p1 set packetSize_ 50000
$p1 set interval_ 0.0001
set p2 [new Agent/Ping]
$ns attach-agent $n1 $p2
set p3 [new Agent/Ping]
$ns attach-agent $n2 $p3
$p3 set packetSize_ 30000
$p3 set interval_ 0.00001
set p4 [new Agent/Ping]
$ns attach-agent $n3 $p4
```

```
set p5 [new Agent/Ping]
$ns attach-agent $n5 $p5
$ns queue-limit $n0 $n4 5
$ns queue-limit $n2 $n4 3
$ns queue-limit $n4 $n5 2
Agent/Ping instproc recv {from rtt} {
$self instvar node_
puts "node [$node_ id] received answer from $from with round trip time $rtt msec"
}
# please provide space between $node_ and id. No space between $ and from. No
#space between and $ and rtt */
$ns connect $p1 $p5
$ns connect $p3 $p4
proc finish { } {
global ns nf tf
$ns flush-trace
close $nf
close $tf
exec nam lab2.nam &
exit 0
}
$ns at 0.1 "$p1 send"
$ns at 0.2 "$p1 send"
$ns at 0.3 "$p1 send"
$ns at 0.4 "$p1 send"
$ns at 0.5 "$p1 send"
$ns at 0.6 "$p1 send"
$ns at 0.7 "$p1 send"
$ns at 0.8 "$p1 send"
$ns at 0.9 "$p1 send"
$ns at 1.0 "$p1 send"
$ns at 1.1 "$p1 send"
$ns at 1.2 "$p1 send"
$ns at 1.3 "$p1 send"
```

```
$ns at 1.4 "$p1 send"
$ns at 1.5 "$p1 send"
$ns at 1.6 "$p1 send"
$ns at 1.7 "$p1 send"
$ns at 1.8 "$p1 send"
$ns at 1.9 "$p1 send"
$ns at 2.0 "$p1 send"
$ns at 2.1 "$p1 send"
$ns at 2.2 "$p1 send"
$ns at 2.3 "$p1 send"
$ns at 2.4 "$p1 send"
$ns at 2.5 "$p1 send"
$ns at 2.6 "$p1 send"
$ns at 2.7 "$p1 send"
$ns at 2.8 "$p1 send"
$ns at 2.9 "$p1 send"
$ns at 0.1 "$p3 send"
$ns at 0.2 "$p3 send"
$ns at 0.3 "$p3 send"
$ns at 0.4 "$p3 send"
$ns at 0.5 "$p3 send"
$ns at 0.6 "$p3 send"
$ns at 0.7 "$p3 send"
$ns at 0.8 "$p3 send"
$ns at 0.9 "$p3 send"
$ns at 1.0 "$p3 send"
$ns at 1.1 "$p3 send"
$ns at 1.2 "$p3 send"
$ns at 1.3 "$p3 send"
$ns at 1.4 "$p3 send"
$ns at 1.5 "$p3 send"
$ns at 1.6 "$p3 send"
$ns at 1.7 "$p3 send"
$ns at 1.8 "$p3 send"
```

```
$ns at 1.9 "$p3 send"
$ns at 2.0 "$p3 send"
$ns at 2.1 "$p3 send"
$ns at 2.2 "$p3 send"
$ns at 2.3 "$p3 send"
$ns at 2.4 "$p3 send"
$ns at 2.5 "$p3 send"
$ns at 2.6 "$p3 send"
$ns at 2.7 "$p3 send"
$ns at 2.8 "$p3 send"
$ns at 2.9 "$p3 send"
$ns at 3.0 "finish"
$ns run
```

AWK file (Open a new editor using "vi command" and write awk file and save with ".awk" extension)

```
BEGIN{
drop=0;
}
{
if($1= ="d" )
{
drop++;
}
} END{
printf("Total number of %s packets dropped due to congestion =%d\n",$5,drop);
}
```

STEPS FOR EXECUTION

1) Open vi editor and type program. Program name should have the extension ― .tcl ‖

2) Save the program by pressing "ESC key" first, followed by "Shift and :" keys simultaneously and type "wq" and press Enter key.

3) Open vi editor and type awk program. Program name should have the extension ―.awk ‖

4) Save the program by pressing "ESC key" first, followed by "Shift and :" keys simultaneously and type "wq" and press Enter key.

5) Run the simulation program

i) Here “ns” indicates network simulator. We get the topology shown in the snapshot.

ii) Now press the play button in the simulation window and the simulation will begins.

6) After simulation is completed run awk file to see the output ,

7) To see the trace file contents open the file as ,

Steps for execution

[root@localhost

~]# vi lab2.tcl

[root@localhost

~]# vi lab2.awk

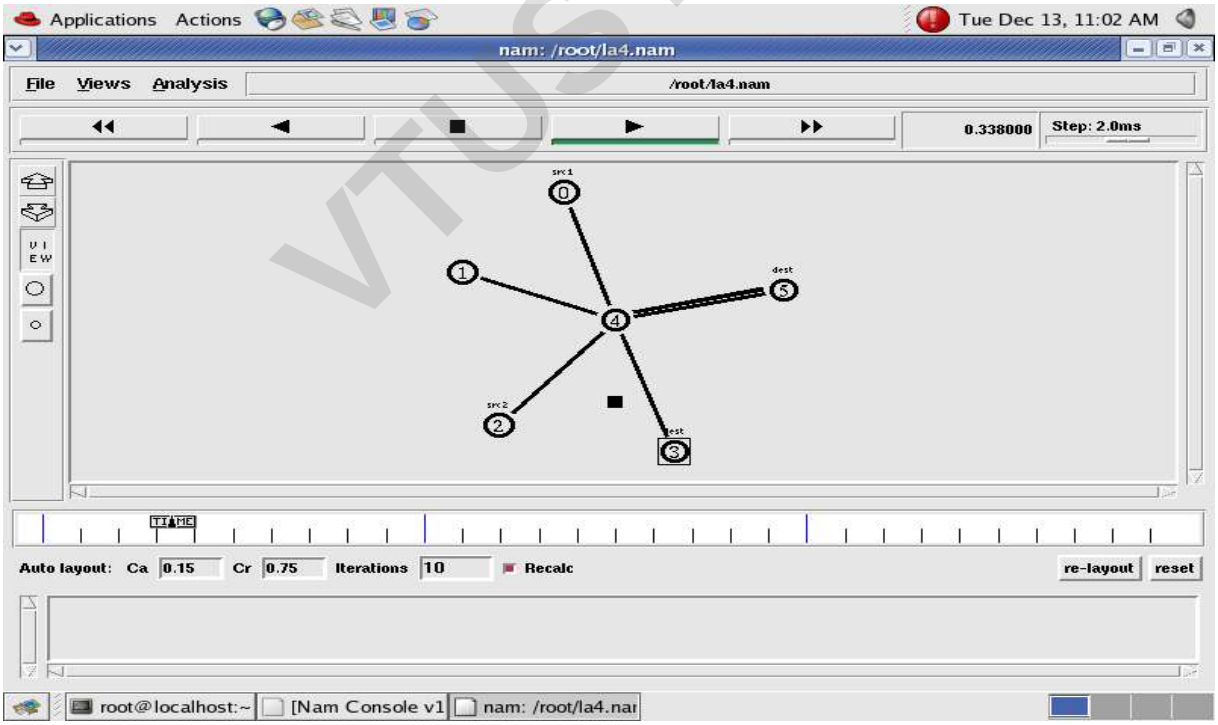
[root@localhost~]#

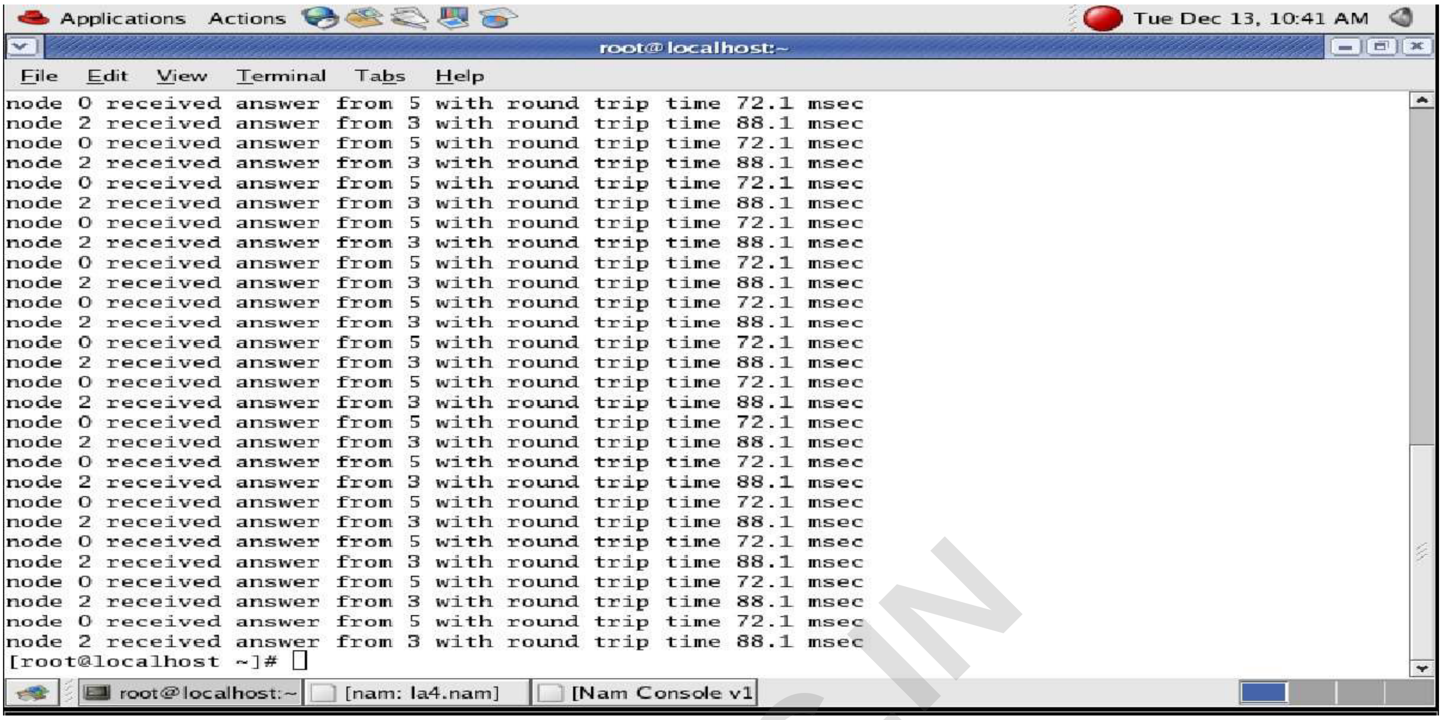
ns lab2.tcl

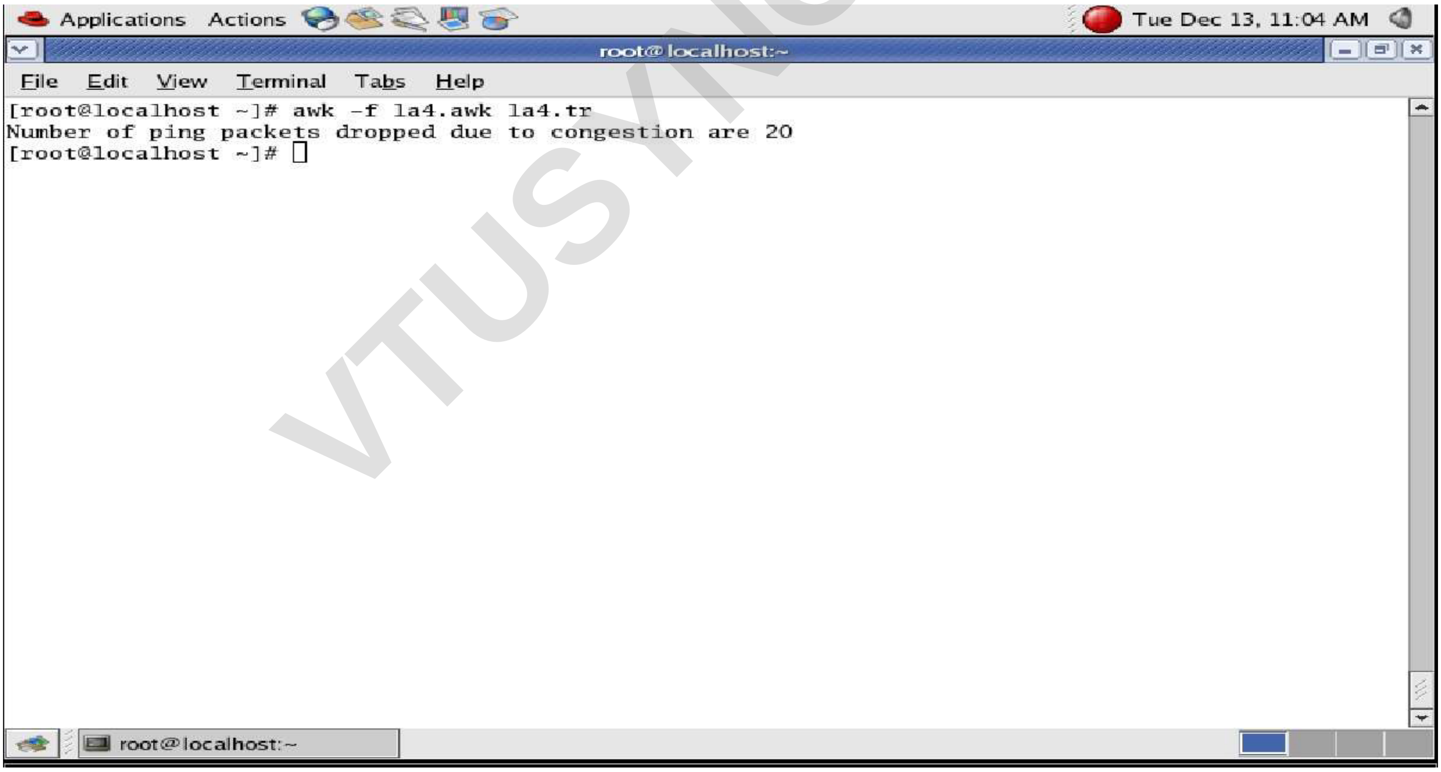
[root@localhost~]# awk –f lab2.awk lab2.tr

[root@localhost~]# vi lab2.tr

Topology: Output







Note:

Vary the bandwidth and queue size between the nodes n0-n2 , n2-n4. n6-n2 and n2- n5 and see the number of packets dropped at the nodes.

3. Implement an Ethernet LAN using n nodes and set multiple traffic nodes and plot congestion window for different source / destination

```
set ns [new Simulator]
set tf [open lab3.tr w]
$ns trace-all $tf
set nf [open lab3.nam w]
$ns namtrace-all $nf
set n0 [$ns node]
$n0 color "magenta"
$n0 label "src1"
set n1 [$ns node]
set n2 [$ns node]
$n2 color "magenta"
$n2 label "src2"
set n3 [$ns node]
$n3 color "blue"
$n3 label "dest2"
set n4 [$ns node]
set n5 [$ns node]
$n5 color "blue"
$n5 l
label "dest1"
$ns make-lan "$n0 $n1 $n2 $n3 $n4" 100Mb 100ms LL Queue/DropTail Mac/802_3
/* should come in single line */
$ns duplex-link $n4 $n5 1Mb 1ms DropTail
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
$ftp0 set packetSize_ 500
$ftp0 set interval_ 0.0001
set sink5 [new Agent/TCPSink]
```

```
$ns attach-agent $n5 $sink5
$ns connect $tcp0 $sink5 set tcp2 [new Agent/TCP]
$ns attach-agent $n2 $tcp2
set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2
$ftp2 set packetSize_ 600
$ftp2 set interval_ 0.001
set sink3 [new Agent/TCPSink]
$ns attach-agent $n3 $sink3
$ns connect $tcp2 $sink3
set file1 [open file1.tr w]
$tcp0 attach $file1
set file2 [open file2.tr w]
$tcp2 attach $file2
$tcp0 trace cwnd_ /* must put underscore ( _ ) after cwnd and no space between them*/
$tcp2 trace cwnd_
proc finish { } { global ns nf tf
$ns flush-trace close $tf
close $nf
exec nam lab3.nam & exit 0
}
$ns at 0.1 "$ftp0 start"
$ns at 5 "$ftp0 stop"
$ns at 7 "$ftp0 start"
$ns at 0.2 "$ftp2 start"
$ns at 8 "$ftp2 stop"
$ns at 14 "$ftp0 stop"
$ns at 10 "$ftp2 start"
$ns at 15 "$ftp2 stop"
$ns at 16 "finish"
$ns run
```

AWK file (Open a new editor using "vi command" and write awk file and save with

".awk" extension)

cwnd:- means congestion window

```
BEGIN {
}
{
if($6= ="cwnd_") /* don't leave space after writing cwnd_ */ printf("%f\t%f\t\n",$1,$7); /*
you must put \n in printf */
}
END {
}
```

Steps for execution

[root@localhost ~]# vi lab3.tcl

[root@localhost ~]# vi lab3.awk

[root@localhost~]#

ns lab3.tcl STEPS FOR EXECUTION:

1) Open vi editor and type program. Program name should have the extension **— .tcl**

2) Save the program by pressing **"ESC key"** first, followed by **"Shift and :"** keys simultaneously and type **"wq"** and press **Enter key**.

3) Open vi editor and type **awk** program. Program name should have the extension **–.awk**

4) Save the program by pressing **"ESC key"** first, followed by **"Shift and :"** keys simultaneously and type **"wq"** and press **Enter key**.

5) Run the simulation program

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

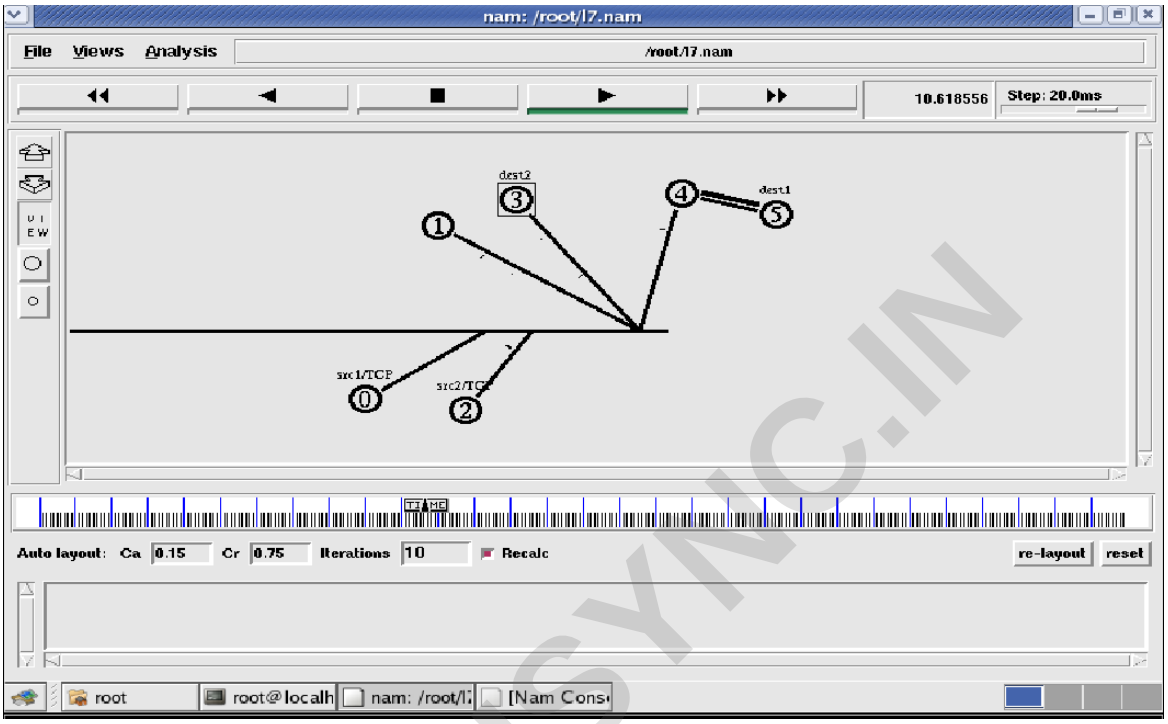
i. [root@localhost~]# awk –f lab3.awk file1.tr > a1

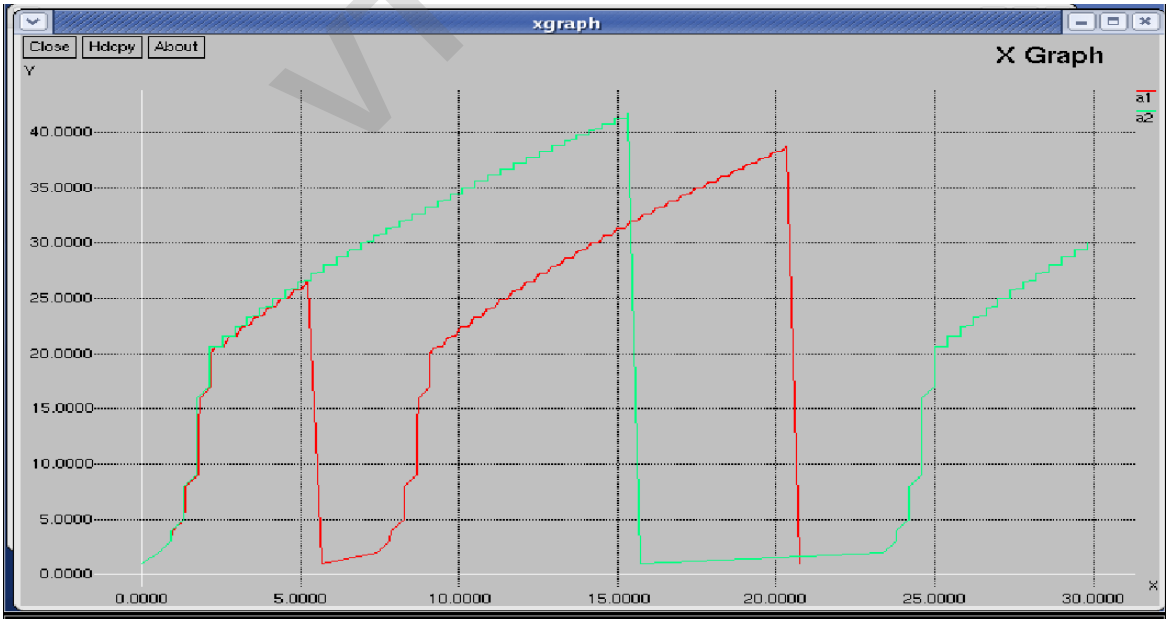
ii. [root@localhost~]# awk –f lab3.awk file2.tr > a2

iii. [root@localhost~]# xgraph a1 a2

Topology

Output





4.Write a program for error detecting code using CRC-CCITT (16- bits)

Whenever digital data is stored or interfaced, data corruption might occur. Since the beginning of computer science, developers have been thinking of ways to deal with this type of problem. For serial data they came up with the solution to attach a parity bit to each sent byte. This simple detection mechanism works if an odd number of bits in a byte changes, but an even number of false bits in one byte will not be detected by the parity check. To overcome this problem developers have searched for mathematical sound mechanisms to detect multiple false bits. The CRC calculation or cyclic redundancy check was the result of this. Nowadays CRC calculations are used in all types of communications. All packets sent over a network connection are checked with a CRC. Also each data block on your hard disk has a CRC value attached to it. Modern computer world cannot do without these CRC calculations. So let's see why they are so widely used. The answer is simple; they are powerful, detect many types of errors and are extremely fast to calculate especially when dedicated hardware chips are used.

The idea behind CRC calculation is to look at the data as one large binary number. This number is divided by a certain value and the remainder of the calculation is called the CRC. Dividing in the CRC calculation at first looks to cost a lot of computing power, but it can be performed very quickly if we use a method similar to the one learned at school. We will as an example calculate the remainder for the character 'm'—which is 1101101 in binary notation— by dividing it by 19 or 10011. Please note that 19 is an odd number. This is necessary as we will see further on. Please refer to your schoolbooks as the binary calculation method here is not very different from the decimal method you learned when you were young.

It might only look a little bit strange. Also notations differ between countries, but the method is similar.

```
                    1 0 1 = 5
               ------------
1 0 0 1 1 / 1 1 0 1 1 0 1
            1 0 0 1 1 | |
            --------- | |
              1 0 0 0 0 |
              0 0 0 0 0 |
              --------- |
              1 0 0 0 0 1
                1 0 0 1 1
                ---------
                  1 1 1 0 = 14 = remainder
```

With decimal calculations you can quickly check that 109 divided by 19 gives a quotient of 5 with 14 as the remainder. But what we also see in the scheme is that every bit extra to check only costs one binary comparison and in 50% of the cases one binary subtraction. You can easily increase the number of bits of the test data string—for example to 56 bits if we use our example value "Lammert"—and the result can be calculated with 56 binary comparisons and an average of 28 binary subtractions. This can be implemented in hardware directly with only very few transistors involved. Also software algorithms can be very efficient.

All of the CRC formulas you will encounter are simply checksum algorithms based on modulo-2 binary division where we ignore carry bits and in effect the subtraction will be equal to an exclusive or operation. Though some differences exist in the specifics across different CRC formulas, the basic mathematical process is always the same:

- The message bits are appended with c zero bits; this augmented message is the dividend
- A predetermined c+1-bit binary sequence, called the generator polynomial, is the divisor
- The checksum is the c-bit remainder that results from the division operation

Table 1 lists some of the most commonly used generator polynomials for 16- and 32-bit CRCs. Remember that the width of the divisor is always one bit wider than the remainder. So, for example, you'd use a 17-bit generator polynomial whenever a 16-bit checksum is required.

	CRC-CCITT	CRC-16	CRC-32
Checksum Width	16 bits	16 bits	32 bits
Generator Polynomial	10001000000100001	11000000000000101	100000100110000010001110110110111

International Standard CRC Polynomials

SOURCE CODE:

```
import java.util.Scanner; class CRC{
public static void main(String args[])
{
Scanner sc = new Scanner(System.in);
//Input Data Stream
System.out.print("Enter data stream: ");
String datastream = sc.nextLine();
 System.out.print("Enter generator: ");
 String generator = sc.nextLine();
int data[] = new int[datastream.length() + generator.length()-1];
int divisor[] = new int[generator.length()];
for(int i=0;i<datastream.length();i++)
data[i] = Integer.parseInt(datastream.charAt(i)+"");
 for(int i=0;i<generator.length();i++)
divisor[i] = Integer.parseInt(generator.charAt(i)+"");
//Calculation of CRC
for(int i=0;i<datastream.length();i++){
if(data[i]==1)
for(int j=0;j<divisor.length;j++)
data[i+j] ^= divisor[j];
}
//Display CRC
System.out.print("The CRC code is: ");
for(int i=0;i<datastream.length();i++)
data[i] = Integer.parseInt(datastream.charAt(i)+"");
for(int i=0;i<data.length;i++) System.out.print(data[i]);
System.out.println();
//Check for input CRC code
System.out.print("Enter CRC code: ");
datastream = sc.nextLine();
System.out.print("Enter generator: ");
```

```
generator = sc.nextLine();
data = new int[datastream.length() + generator.length()-1];
divisor = new int[generator.length()];
for(int i=0;i<datastream.length();i++)
data[i] = Integer.parseInt(datastream.charAt(i)+"");
for(int i=0;i<generator.length();i++)
divisor[i] = Integer.parseInt(generator.charAt(i)+"");
//Calculation of remainder
for(int i=0;i<datastream.length();i++){
if(data[i]==1)
for(int j=0;j<divisor.length;j++)
data[i+j] ^= divisor[j];
}
//Display validity of data
boolean valid = true;
for(int i=0;i<data.length;i++)
if(data[i]==1){
valid = false;
break;
}
if(valid==true) System.out.println("Data stream is valid");
else System.out.println("Data stream is invalid. CRC error occured.");
}
}
```

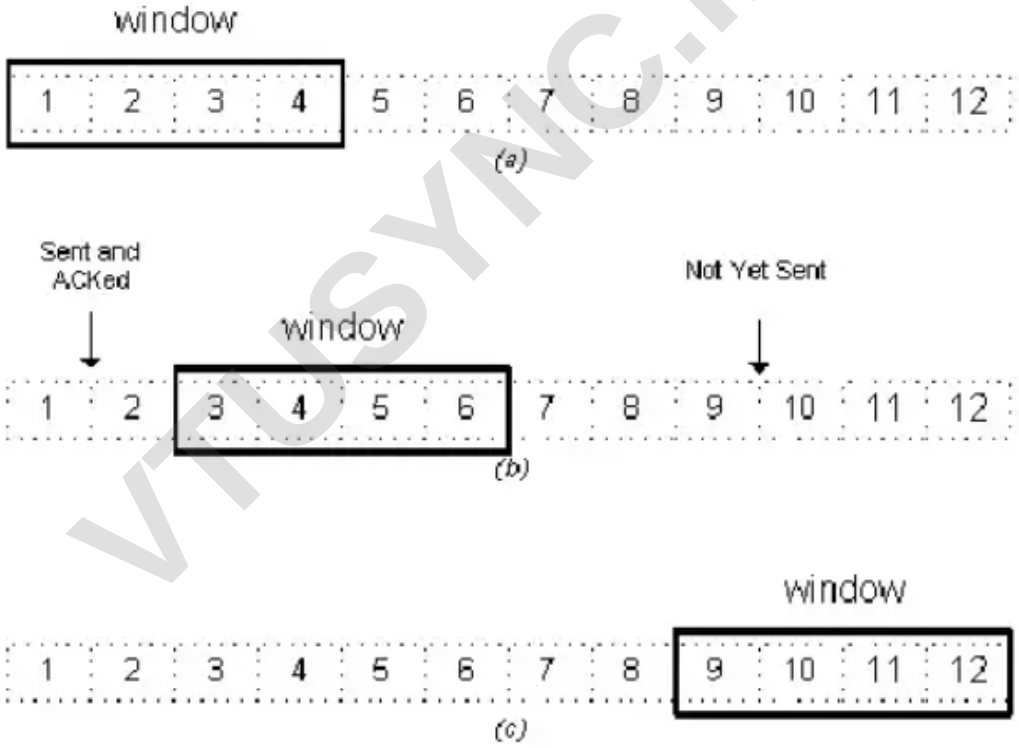

OUTPUT:

```
D:\teena-cn-prog>javac CRC.java

D:\teena-cn-prog>java CRC
Enter data stream: 101010
Enter generator: 111
The CRC code is: 10101000
Enter CRC code: 10101001
Enter generator: 111
Data stream is invalid. CRC error occured.

D:\teena-cn-prog>java CRC
Enter data stream: 101010
Enter generator: 111
The CRC code is: 10101000
Enter CRC code: 10101000
Enter generator: 111
Data stream is valid
```

5 Develop a program to implement a sliding window protocol in the data link layer.

In computer networks sliding window protocol is a method to transmit data on a network. Sliding window protocol is applied on the Data Link Layer of OSI model. At data link layer data is in the form of frames. In Networking, Window simply means a buffer which has data frames that needs to be transmitted.
Both sender and receiver agrees on some window size. If window size=w then after sending w frames sender waits for the acknowledgement (ack) of the first frame.

As soon as sender receives the acknowledgement of a frame it is replaced by the next frames to be transmitted by the sender. If receiver sends a collective or cumulative acknowledgement to sender then it understands that more than one frames are properly received, for eg:- if ack of frame 3 is received it understands that frame 1 and frame 2 are received properly.



In sliding window protocol the receiver has to have some memory to compensate any loss in transmission or if the frames are received unordered.

```
import java.util.LinkedList;
import java.util.Queue;
import java.util.Random;
import java.util.Scanner;

class SlidingWindowProtocol {
    private int windowSize;
    private int totalFrames;
    private Queue<Integer> senderWindow;
    private int nextFrameToSend;
    private int acknowledgedFrame;
    private Random random;

    public SlidingWindowProtocol(int windowSize, int totalFrames) {
        this.windowSize = windowSize;
        this.totalFrames = totalFrames;
        this.senderWindow = new LinkedList<>();
        this.nextFrameToSend = 0;
        this.acknowledgedFrame = -1; // No frames acknowledged initially
        this.random = new Random();
    }

    public void sendFrame(int frame) {
        System.out.println("Sending frame: " + frame);
        senderWindow.add(frame);
    }

    public void receiveAck(int ack) {
        if (ack > acknowledgedFrame) {
            System.out.println("Received ACK for frame: " + ack);
            acknowledgedFrame = ack;
            slideWindow();
        }
    }
```

```
private void slideWindow() {
    while (senderWindow.contains(acknowledgedFrame + 1)) {
        senderWindow.remove(acknowledgedFrame + 1);
        acknowledgedFrame++;
    }
}

public void run() {
    while (nextFrameToSend < totalFrames) {
        while (senderWindow.size() < windowSize && nextFrameToSend < totalFrames) {
            sendFrame(nextFrameToSend);
            nextFrameToSend++;
            try {
                Thread.sleep(random.nextInt(300) + 100); // Simulate network delay
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }

        // Simulate receiving ACKs randomly
        if (random.nextBoolean()) {
            int ack = random.nextInt(totalFrames);
            receiveAck(ack);
        } else {
            System.out.println("No ACK received.");
        }
    }
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter window size: ");
    int windowSize = scanner.nextInt();
    System.out.print("Enter total number of frames: ");
    int totalFrames = scanner.nextInt();
```

```
        SlidingWindowProtocol protocol = new SlidingWindowProtocol(windowSize, totalFrames);
        protocol.run();
    }
}
```

Input:

Enter window size: 4

Enter total number of frames: 10

Output:

Sending frame: 0

Sending frame: 1

Sending frame: 2

Sending frame: 3

Received ACK for frame: 1

Sending frame: 4

Sending frame: 5

No ACK received.

Received ACK for frame: 3

Sending frame: 6

Sending frame: 7

No ACK received.

Received ACK for frame: 6

Sending frame: 8

No ACK received.

Received ACK for frame: 7

Sending frame: 9

No ACK received.

Received ACK for frame: 8

Received ACK for frame: 9

6. Develop a program to find the shortest path between vertices using the Bellman-Ford and path vector routing algorithm.

Distance Vector Algorithm is a decentralized routing algorithm that requires that each router simply inform its neighbors of its routing table. For each network path, the receiving routers pick the neighbor advertising the lowest cost, then add this entry into its routing table for re-advertisement. To find the shortest path, Distance Vector Algorithm is based on one of two basic algorithms: the Bellman-Ford and the Dijkstra algorithms.

Routers that use this algorithm have to maintain the distance tables (which is a one-dimension array -- "a vector"), which tell the distances and shortest path to sending packets to each node in the network. The information in the distance table is always up date by exchanging information with the neighboring nodes. The number of data in the table equals

to that of all nodes in networks (excluded itself). The columns of table represent the directly attached neighbors whereas the rows represent all destinations in the network. Each data contains the path for sending packets to each destination in the network and distance/or time to transmit on that path (we call this as "cost"). The measurements in this algorithm are the number of hops, latency, the number of outgoing packets, etc.

The Bellman–Ford algorithm is an algorithm that computes shortest paths from a single source vertex to all of the other vertices in a weighted digraph. It is slower than Dijkstra's algorithm for the same problem, but more versatile, as it is capable of handling graphs in which some of the edge weights are negative numbers. Negative edge weights are found in various applications of graphs, hence the usefulness of this algorithm. If a graph contains a "negative cycle" (i.e. a cycle whose edges sum to a negative value) that is reachable from the source, then there is no cheapest path: any path that has a point on the negative cycle can be made cheaper by one more walk around the negative cycle. In such a case, the Bellman–Ford algorithm can detect negative cycles and report their existence.

SOURCE CODE:

```
import java.util.Scanner;
public class BellmanFord
 {
 private int D[];
 private int num_ver;
 public static final int MAX_VALUE = 999;
 public BellmanFord(int num_ver)
 {
 this.num_ver = num_ver;
 D = new int[num_ver + 1];
 }
 public void BellmanFordEvaluation(int source, int A[][])
```

```
{
for (int node = 1; node <= num_ver; node++)
{
D[node] = MAX_VALUE;

}
D[source] = 0;
for (int node = 1; node <= num_ver - 1; node++)
{
for (int sn = 1; sn <= num_ver; sn++)
{
for (int dn = 1; dn <= num_ver; dn++)
{
if (A[sn][dn] != MAX_VALUE)
{
if (D[dn] > D[sn]+ A[sn][dn])
D[dn] = D[sn] + A[sn][dn];
}
}
}
}
for (int sn = 1; sn <= num_ver; sn++)
{
for (int dn = 1; dn <= num_ver; dn++)
{
if (A[sn][dn] != MAX_VALUE)
{
if (D[dn] > D[sn]+ A[sn][dn])
System.out.println("The Graph contains negative egde cycle");
}
}
}
for (int vertex = 1; vertex <= num_ver; vertex++)
{
```

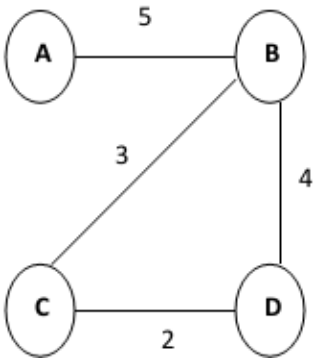
```
System.out.println("distance of source " + source + " to "+ vertex + " is " + D[vertex]);
}
}
public static void main(String[ ] args)
{
int num_ver = 0;
int source;
Scanner scanner = new Scanner(System.in);
System.out.println("Enter the number of vertices");
num_ver = scanner.nextInt();
int A[][] = new int[num_ver + 1][num_ver + 1];
System.out.println("Enter the adjacency matrix");
for (int sn = 1; sn <= num_ver; sn++)
{
for (int dn = 1; dn <= num_ver; dn++)
{
A[sn][dn] = scanner.nextInt();
if (sn == dn)
{
A[sn][dn] = 0;
continue;
}
if (A[sn][dn] == 0)
{
A[sn][dn] = MAX_VALUE;
}
}
}
System.out.println("Enter the source vertex");
source = scanner.nextInt();
BellmanFord b = new BellmanFord (num_ver);
b.BellmanFordEvaluation(source, A);
scanner.close();
```

```
}
}
```

OUTPUT:

```
D:\cn-lab>javac BellmanFord.java

D:\cn-lab>java BellmanFord
Enter the number of vertices
4
Enter the adjacency matrix
0 5 0 0
5 0 3 7
0 3 0 2
0 7 2 0
Enter the source vertex
2
distance of source 2 to 1 is 5
distance of source 2 to 2 is 0
distance of source 2 to 3 is 3
distance of source 2 to 4 is 5

D:\cn-lab>
```

Input graph:



7. Using TCP/IP sockets, write a client – server program to make the client send the file name and to make the server send back the contents of the requested file if present.

Socket is an interface which enables the client and the server to communicate and pass on information from one another. Sockets provide the communication mechanism between two computers using TCP. A client program creates a socket on its end of the communication and attempts to connect that socket to a server. When the connection is made, the server creates a socket object on its end of the communication. The client and the server can now communicate by writing to and reading from the socket.

tcp program server side

```
import java.net.*;
import java.io.*;
public class server1
{
```

```
public static void main(String args[]) throws Exception
{ // establishing the connection with the server
ServerSocket sersock = new ServerSocket(4000);
System.out.println("Server ready for connection");
Socket sock = sersock.accept(); // binding with port: 4000
System.out.println("Connection is successful");
// reading the file name from client
InputStream istream = sock.getInputStream( );
BufferedReader fileRead =new BufferedReader(new InputStreamReader(istream));
String fname = fileRead.readLine( );
// reading file contents
BufferedReader contentRead = new BufferedReader(new FileReader(fname) );
// keeping output stream ready to send the contents
OutputStream ostream = sock.getOutputStream( );
PrintWriter pwrite = new PrintWriter(ostream, true);
String str;
while((str = contentRead.readLine()) != null) // reading line-by-line from file
{
pwrite.println(str); // sending each line to client
}
sock.close(); sersock.close(); // closing network sockets
pwrite.close(); fileRead.close(); contentRead.close();
}
}
```

tcp program client side

```
import java.net.*;
import java.io.*;
public class client1
{
public static void main( String args[ ] ) throws Exception
{
Socket sock = new Socket( "127.0.0.1", 4000);
// reading the file name from keyboard. Uses input stream
```

```
System.out.print("Enter the file name");
BufferedReader keyRead = new BufferedReader(new InputStreamReader(System.in));
String fname = keyRead.readLine();
// sending the file name to server. Uses PrintWriter
OutputStream ostream = sock.getOutputStream( );
PrintWriter pwrite = new PrintWriter(ostream, true);
pwrite.println(fname);
// receiving the contents from server. Uses input stream
InputStream istream = sock.getInputStream();
BufferedReader socketRead = new BufferedReader(new InputStreamReader(istream));
String str;
while((str = socketRead.readLine()) != null) // reading line-by-line
{
System.out.println(str);
}
pwrite.close(); socketRead.close(); keyRead.close();
}
}
```

8. Develop a program on a datagram socket for client/server to display the messages on client side, typed at the server side.

A datagram socket is the one for sending or receiving point for a packet delivery service. Each packet sent or received on a datagram socket is individually addressed and routed. Multiple packets sent from one machine to another may be routed differently, and may arrive in any order.

SERVER

```
package server;
import java.io.*;
import java.net.*;
public class srvpgm {
public static void main(String[] args) throws Exception{
DatagramSocket serverSocket = new DatagramSocket(9876);
byte[] receivebuffer = new byte[1024];
```

```
byte[] sendbuffer = new byte[1024];
System.out.println("Waiting for client request!") ;
while(true)
{
DatagramPacket recvdpkt = new DatagramPacket(receivebuffer, 0, receivebuffer.length);
serverSocket.receive(recvdpkt);
InetAddress IP = recvdpkt.getAddress();
int portno = recvdpkt.getPort();
System.out.println("Connected to Client " +IP ) ;
String clientdata = new String(recvdpkt.getData(), recvdpkt.getOffset(),recvdpkt.getLength());
System.out.println("\nClient Message: "+ clientdata);
System.out.print("\nServer sending..: ");
BufferedReader serverRead = new BufferedReader(new InputStreamReader (System.in) );
String serverdata = serverRead.readLine();
sendbuffer = serverdata.getBytes();
DatagramPacket sendPacket = new DatagramPacket(sendbuffer, sendbuffer.length, IP,portno);
serverSocket.send(sendPacket);
System.out.println("Message sent to client!") ;
System.out.println() ;
}
}
}
```

CLIENT

```
package client;
import java.io.*;
import java.net.*;
public class clientpgm {
public static void main(String[] args) throws Exception{
BufferedReader clientRead =new BufferedReader(new InputStreamReader(System.in));
DatagramSocket clientSocket = new DatagramSocket();
InetAddress IP = InetAddress.getByName("127.0.0.1");
byte[] sendbuffer = new byte[1024];
byte[] receivebuffer = new byte[1024];
```

```
System.out.print("\nClient: ");
String clientData = clientRead.readLine();
sendbuffer = clientData.getBytes();
DatagramPacket sendPacket =
new DatagramPacket(sendbuffer, sendbuffer.length, IP, 9876);
clientSocket.send(sendPacket);
System.out.println("Waiting for Server response!") ;
DatagramPacket receivePacket =
new DatagramPacket(receivebuffer, receivebuffer.length);
clientSocket.receive(receivePacket);
String serverData = new String(receivePacket.getData());
System.out.println("\nServer Message received: " + serverData);
System.out.println("Client terminated.") ;
clientSocket.close();
}
}
```


OUTPUT:

SERVER END CLIENT END

Waiting for Client request Client: Hello I am client

Waiting for server response!

Connected to client 127.0.0.1

Client message: Hello I am client

Hello I am server

Message sent to client!

Server message received hello I am server

Client Terminated.

9. Develop a program for a simple RSA algorithm to encrypt and decrypt the data.

RSA is an example of public key cryptography. It was developed by Rivest, Shamir and Adelman. The RSA algorithm can be used for both public key encryption and digital signatures. Its security is based on the difficulty of factoring large integers.

The RSA algorithm's efficiency requires a fast method for performing the modular exponentiation operation. A less efficient, conventional method includes raising a number (the input) to a power (the secret or public key of the algorithm, denoted *e* and *d*, respectively) and taking the remainder of the division with *N*. A straight-forward implementation performs these two steps of the operation sequentially: first, raise it to the power and second, apply modulo. The RSA algorithm comprises of three steps, which are depicted below:

Key Generation Algorithm

1. Generate two large random primes, p and q, of approximately equal size such that their product n = p*q

2. Compute n = p*q and Euler's totient function (φ) phi(n) = (p-1)(q-1).

3. Choose an integer e, 1 < e < phi, such that gcd(e, phi) = 1.

4. Compute the secret exponent d, 1 < d < phi, such that e*d ≡ 1 (mod phi).

5. The public key is (e, n) and the private key is (d, n). The values of p, q, and phi should also be kept secret.

Encryption

Sender A does the following:-

1. Using the public key (e,n)

2. Represents the plaintext message as a positive integer M

3. Computes the cipher text C = M^e mod n.

4. Sends the cipher text C to B (Receiver).

Decryption

Recipient B does the following:-

1. Uses his private key (d, n) to compute M = C^d mod n.

2. Extracts the plaintext from the integer representative m.

SOURCE CODE:

```
import java.math.BigInteger;
import java.util.Random;
import java.io.*;
public class RSA {
BigInteger p, q, N, phi, e, d;
int bitlength = 100;
Random r;
public RSA() {
r = new Random();
// returns a prime number with in the specified bit length
p = BigInteger.probablePrime(bitlength, r);
q = BigInteger.probablePrime(bitlength, r);
N = p.multiply(q);
// In c equivalent – phi = (p-1)*(q-1)
phi = p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));
e = BigInteger.probablePrime(bitlength/2, r);
// In C equivalent - while ( phi.gcd(e) > 1 && e < phi )
while (phi.gcd(e).compareTo(BigInteger.ONE) > 0 && e.compareTo(phi) < 0 ) {
```

```
e.add(BigInteger.ONE); // In C equivalent- e+1
}
d = e.modInverse(phi);
}
// RSA Main
public static void main(String[] args) throws IOException{
RSA rsa = new RSA();
DataInputStream in=new DataInputStream(System.in);
String teststring ;
System.out.println("Enter the plain text:");
teststring=in.readLine();
System.out.println("Encrypting String: " + teststring);
System.out.println("String in Bytes: " + bytesToString(teststring.getBytes()));
// encrypt
byte[] encrypted = rsa.encrypt(teststring.getBytes());
System.out.println("Encrypted String in Bytes: " + bytesToString(encrypted));
// decrypt
byte[] decrypted = rsa.decrypt(encrypted);
System.out.println("Decrypted String in Bytes: " + bytesToString(decrypted));
System.out.println("Decrypted String: ");
System.out.println(new String(decrypted));
}
private static String bytesToString(byte[] encrypted) {
String test = "";
for (byte b : encrypted) {
test += Byte.toString(b);
}
return test;
}
// Encrypt message
public byte[] encrypt(byte[] message) {
return (new BigInteger(message)).modPow(e, N).toByteArray();
}
// Decrypt message
public byte[] decrypt(byte[] message) {
return (new BigInteger(message)).modPow(d, N).toByteArray();
}
}
```

OUTPUT

```
Enter the plain text:
EPCET
Encrypting String: EPCET
String in Bytes: 8397112116104971031051141105
```

Encrypted String in Bytes: 91-2-10491-456154-32-71-122-35-187884101-3125-1856102-17356-582

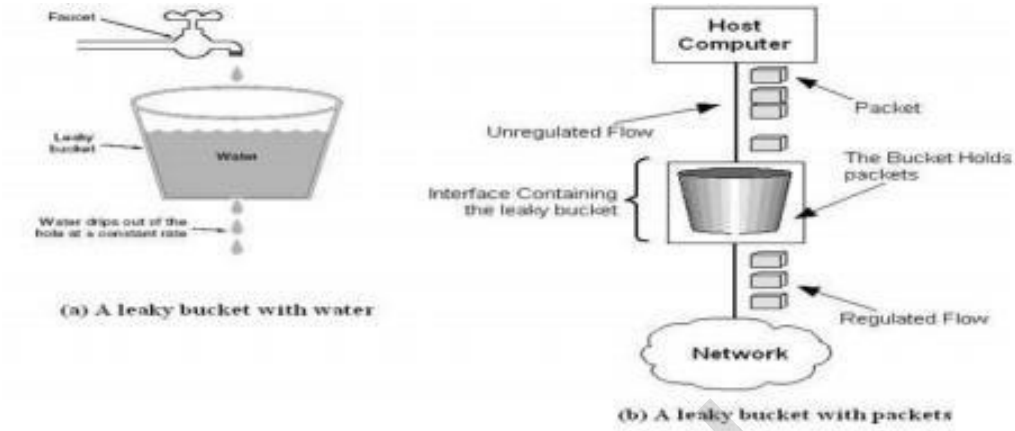
Decrypted String in Bytes: 8397112116104971031051141 05

Decrypted String:

EPCET

10. Develop a program for congestion control using a leaky bucket algorithm.

The main concept of the leaky bucket algorithm is that the output data flow remains constant despite the variant input traffic, such as the water flow in a bucket with a small hole at the bottom. In case the bucket contains water (or packets) then the output flow follows a constant rate, while if the bucket is full any additional load will be lost because of spillover. In a similar way if the bucket is empty the output will be zero. From network perspective, leaky bucket consists of a finite queue (bucket) where all the incoming packets are stored in case there is space in the queue, otherwise the packets are discarded. In order to regulate the output flow, leaky bucket transmits one packet from the queue in a fixed time (e.g. at every clock tick). In the following figure we can notice the main rationale of leaky bucket algorithm, for both the two approaches (e.g. leaky bucket with water (a) and with packets (b)).



While leaky bucket eliminates completely bursty traffic by regulating the incoming data flow its main drawback is that it drops packets if the bucket is full. Also, it doesn‘t take into account the idle process of the sender which means that if the host doesn‘t transmit data for some time the bucket becomes empty without permitting the transmission of any packet.

Source Code:

```
import java.util.Scanner;

public class LBucket

{ static int random(int

a) {
return (int) (Math.random() * a);
}

public static void main(String[] args) throws
InterruptedException { int[] packetSize = new int[5];
for (int i = 0; i < 5; i++) {
packetSize[i] = random(10);
System.out.println("PacketSize[" + i + "] : " + packetSize[i]);
}
Scanner scanner = new Scanner(System.in);
System.out.print("Enter output rate : ");
int outputRate = scanner.nextInt();
System.out.print("Enter bucket size :
"); int bucketSize =
scanner.nextInt(); scanner.close();
int p_zsz_rm = 0;
for (int i = 0; i < 5; i++) {
if (packetSize[i] + p_zsz_rm >
bucketSize) { if (packetSize[i] >
bucketSize) {
System.out.println("Incoming packet" + i + " of size " + packetSize[i]
+ " is greater than bucket capacity : PACKET REJECTION");
} else {
System.out.println("Packet " + i + ": Bucket capacity exceeded :
REJECTING new packet");
}
} else {
p_zsz_rm += packetSize[i];
System.out.println("Incoming packet " + i + " of size : " +
packetSize[i]); System.out.println("Bytes remaining for
transmission : " + p_zsz_rm); int p_time = random(6) * 10;
System.out.println("Time left for transmission is " +
p_time); for (int clk = 10; clk <= p_time; clk += 10) {
Thread.sleep(1);
if (p_zsz_rm > 0) {
if (p_zsz_rm <= outputRate) {
```

```
System.out.println("Packet of size " + p_zsz_rm + " transmitted");
p_zsz_rm = 0;
} else {
System.out.println("Packet of size " + outputRate + " transmitted");
p_zsz_rm -= outputRate;
System.out.println("Bytes remaining after transmission " + p_zsz_rm);
System.out.println("Time left : " + (p_time - clk));
}
} else {
System.out.println("No packets to transmit");
}
} //for
} //else
} //for
}  //main
}  //class
```

OUTPUT:

```
PacketSize[0] : 7
PacketSize[1] : 9
PacketSize[2] : 1
PacketSize[3] : 6
PacketSize[4] : 7
Enter output rate : 1
Enter bucket size : 7
Incoming packet 0 of size : 7
Bytes remaining for transmission : 7
Time left for transmission is 40
Packet of size 1 transmitted
Bytes remaining after transmission 6
Time left : 30
Packet of size 1 transmitted
Bytes remaining after transmission 5
Time left : 20
Packet of size 1 transmitted
Bytes remaining after transmission 4
Time left : 10
Packet of size 1 transmitted
Bytes remaining after transmission 3
Time left : 0
Incoming packet1 of size 9 is greater than bucket capacity : PACKET REJECTION
```

```
Incoming packet 2 of size : 1
Bytes remaining for transmission : 4
Time left for transmission is 40
Packet of size 1 transmitted
Bytes remaining after transmission 3
Time left : 30
Packet of size 1 transmitted
Bytes remaining after transmission 2
Time left : 20
Packet of size 1 transmitted
Bytes remaining after transmission 1
Time left : 10
Packet of size 1 transmitted
Incoming packet 3 of size : 6
Bytes remaining for transmission : 6
Time left for transmission is 0
Packet 4: Bucket capacity exceeded : REJECTING new packet
```

Viva Questions

1. What are functions of different layers?
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 requires?
7. What is MTU?
8. Which layer imposes MTU?
9. Differentiate between flow control and congestion control.
10. Differentiate between Point-to-Point Connection and End-to-End connections.
11. What are protocols running in different layers?
12. What is Protocol Stack?
13. Differentiate between TCP and UDP.
14. Differentiate between Connectionless and connection oriented connection.
15. Why frame sorting is required?
16. What is meant by subnet?
17. What is meant by Gateway?
18. What is an IP address?
19. What is MAC address?
20. Why IP address is required when we have MAC address?
21. What is meant by port?
22. What are ephemerical port number and well known port numbers?
23. What is a socket?
24. What are the parameters of socket()?
25. Describe bind(), listen(), accept(),connect(), send() and recv().
26. What are system calls? Mention few of them.
27. What is IPC? Name three techniques.
28. Explain mkfifo(), open(), close() with parameters.
29. What is meant by file descriptor?
30. What is meant by traffic shaping?
31. How do you classify congestion control algorithms?
32. Differentiate between Leaky bucket and Token bucket.
33. How do you implement Leaky bucket?
34. How do you generate busty traffic?
35. What is the polynomial used in CRC-CCITT?
36. What are the other error detection algorithms?
37. What is difference between CRC and Hamming code?
38. Why Hamming code is called 7,4 code?
39. What is odd parity and even parity?
40. What is meant by syndrome?
41. What is generator matrix?
42. What is spanning tree?
43. Where Pirm's algorithm does finds its use in Networks?
44. Differentiate between Prim's and Kruskal's algorithm.
45. What are Routing algorithms?
46. How do you classify routing algorithms? Give examples for each.
47. What are drawbacks in distance vector algorithm?

48. How routers update distances to each of its neighbor?
49. How do you overcome count to infinity problem?
50. What is cryptography?
51. How do you classify cryptographic algorithms?
52. What is public key?
53. What is private key?
54. What are key, ciphertext and plaintext?
55. What is simulation?
56. What are advantages of simulation?
57. Differentiate between Simulation and Emulation.
58. What is meant by router?
59. What is meant by bridge?
60. What is meant by switch?
61. What is meant by hub?
62. Differentiate between route, bridge, switch and hub.
63. What is ping and telnet?
64. What is FTP?
65. What is BER?
66. What is meant by congestion window?
67. What is BSS?
68. What is incoming throughput and outgoing throughput?
69. What is collision?
70. How do you generate multiple traffics across different sender-receiver pairs?
71. How do you setup Ethernet LAN?
72. What is meant by mobile host?
73. Name few other Network simulators
74. Differentiate between logical and physical address.
75. Which address gets affected if a system moves from one place to another place?
76. What is ICMP? What are uses of ICMP? Name few.
77. Which layer implements security for data?