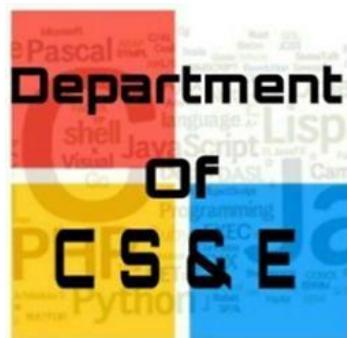**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

COMPUTER NETWORK LAB MANUAL

2024-2025

ATRIA INSTITUTE OF TECHNOLOGY
Adjacent to Bangalore Baptist Hospital
Hebbal Bengaluru-560024

COMPUTER NETWORK LABORATORY

Subject Code: BCS502	IA Marks: 20
Total Number of Lecture Hours: 40	Exam Hours: 03

CONTENTS

8	Develop a program on a datagram socket for client server to display the messages on client side typed at the server side.	60
9	Develop a program for a simple RSA algorithm to encrypt and decrypt the data.	56
10	Develop a program for congestion control using leaky bucket algorithm.	43

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 behaviours.

Basic Architecture of NS2
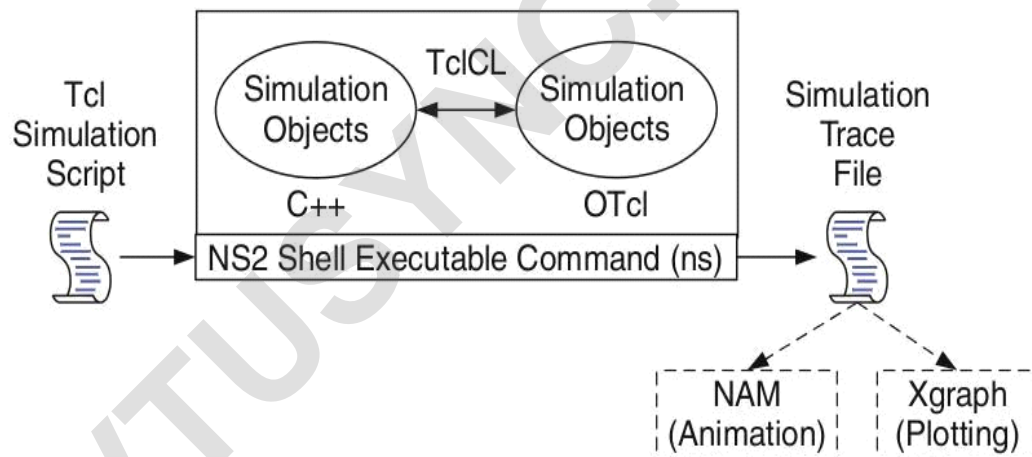
Tcl scripting

- 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 Arithmeti**

  c 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

- **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.

- Initialization and termination aspects of the ns simulator.
- Definition of network nodes, links, queues and topology.
- Definition of agents and of applications.
- The nam visualization tool.

- Tracing and random variables.

Initialization and Termination of TCL
Script in NS-2 An ns simulation starts with
the command

> **set ns [new Simulator]**

Which is thus the first line in the tcl script? This line declares a new variable as using the set command, you can call this variable as you wish, In general people declares it as ns because it is an instance of the Simulator class, so an object the code[new Simulator] is indeed the installation of the class Simulator using the reserved word new.
In order to have output files with data on the simulation (trace files) or files used for visualization (nam files), we need to create the files using "open" command:

#Open the Trace file

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

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

The above creates a 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".

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 a 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]

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 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.

The command	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

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"

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:

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

- 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.
- The second field gives the time at which the event occurs.

- Gives the input node of the link at which the event occurs.
- Gives the output node of the link at which the event occurs.
- Gives the packet type (eg CBR or TCP)
- Gives the packet size
- Some flags
- 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.
- This is the source address given in the form of "node.port".
- This is the destination address, given in the same form.
- 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
- 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.

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 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

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 infront 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.

PART-A

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 f [open lab1.tr w]         /* open a nam trace file in write mode */
set nf [open lab1.nam w]       /* nf – nam file */
$ns trace-all $f               /* tf- trace file */
$ns namtrace-all $nf

proc finish {} {               /* provide space b/w proc and finish and all
are in smallcase */
global f nf ns
$ns flush-trace                /* clears trace file contents */
close $f
close $nf
exec nam lab1.nam &
exit 0
}

set n0 [$ns node]              /* create 3 nodes */
set n1 [$ns node]
set n2 [$ns node]

$n0 label "TCP source"        /* to label nodes */
$n1 label "UDP source"
$n2 label "Sink"

$ns color 1 red
```

$ns color 2 yellow

$ns duplex-link $n0 $n1 1.0Mb 10ms DropTail   /*Letter **M** is capital Mb*/
$ns duplex-link $n1 $n2 1.0Mb 20ms DropTail   /***D** and **T** are capital*/
$ns queue-limit $n1 $n2 10

$ns duplex-link-op $n0 $n1 orient right
$ns duplex-link-op $n1 $n2 orient right

set udp0 [new Agent/UDP]   /* Letters **A,U,D** and **P** are capital */
$ns attach-agent $n1 $udp0

set cbr0 [new Application/Traffic/CBR]   /* **A,T,C,B** and **R** are capital*/
$cbr0 attach-agent $udp0
$cbr0 set packetSize_ 500   /***S** is capital, space after underscore*/
$cbr0 set interval_ 0.005

set null0 [new Agent/Null]   /* **A** and **N** are capital */
$ns attach-agent $n2 $null0
$ns connect $udp0 $null0

set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0

set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0

set sink [new Agent/TCPSink]   /* **T,C,P** and **S** are capital, no space between TCP and   Sink */
$ns attach-agent $n2 $sink
$ftp0 set maxpkts_ 1000
$ns connect $tcp0 $sink

$udp0 set class_ 1   /* space after class underscore*/
$tcp0 set class_ 2

$ns at 0.1 "$cbr0 start"
$ns at 1.0 "$ftp0 start"
$ns at 4.0 "$ftp0 stop"
$ns at 4.5 "$cbr0 stop"

$ns at 5.0 "finish"
$ns run

Steps for execution

1) Open Text editor and type program. Program name should have
   the extension   " .tcl "
   [root@localhost ~]$ gedit lab1.tcl

2) Run the simulation program
   [root@localhost~]$ ns lab1.tcl
   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.

3) After simulation is completed run grep command to find packets
   drop to see the output
   [root@localhost~]$ grep ^d lab1.tr | grep "cbr"
   -c

   [root@localhost~]$ grep ^d lab1.tr | grep "tcp" -
   c

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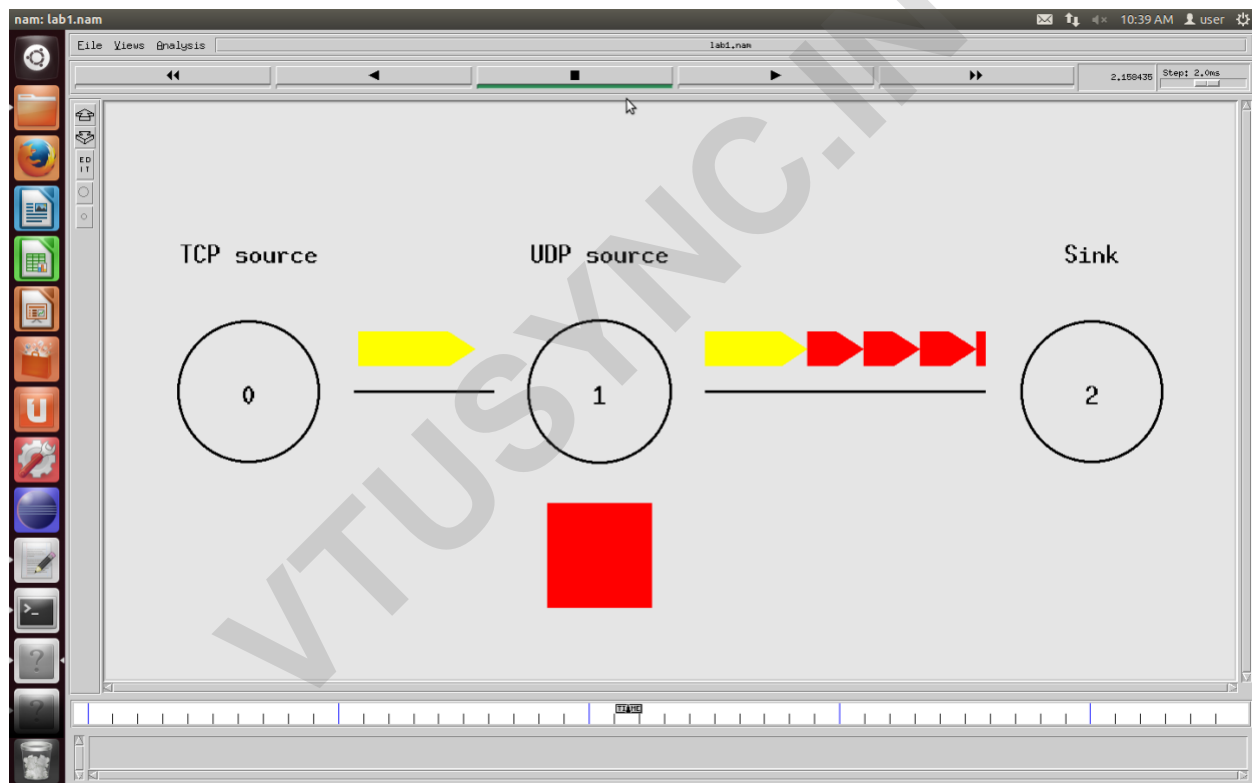   **[root@localhost~]$ vi lab1.tr**

5) To see the total number of tcp and cbr packets drop
   [root@localhost~]$ grep ^d 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

Topology

Output

```
user@user-ThinkCentre-M83:~$ ns lab1.tcl
user@user-ThinkCentre-M83:~$ grep ^d lab1.tr | grep "cbr" -c
17
user@user-ThinkCentre-M83:~$ grep ^d lab1.tr | grep "tcp" -c
13
user@user-ThinkCentre-M83:~$ grep ^d lab1.tr
d 1.57664 0 1 tcp 1040 ------- 2 0.1 2.1 44 367
d 1.585 0 1 cbr 500 ------- 1 0.0 2.0 297 369
d 1.60128 0 1 tcp 1040 ------- 2 0.1 2.1 46 376
d 1.64 0 1 cbr 500 ------- 1 0.0 2.0 308 388
d 1.64624 0 1 tcp 1040 ------- 2 0.1 2.1 49 390
d 1.65 0 1 cbr 500 ------- 1 0.0 2.0 310 391
d 1.665 0 1 cbr 500 ------- 1 0.0 2.0 313 395
d 1.695 0 1 cbr 500 ------- 1 0.0 2.0 319 403
d 1.71088 0 1 tcp 1040 ------- 2 0.1 2.1 51 408
d 1.715 0 1 cbr 500 ------- 1 0.0 2.0 323 410
d 1.72 0 1 cbr 500 ------- 1 0.0 2.0 324 412
d 1.73552 0 1 tcp 1040 ------- 2 0.1 2.1 53 417
d 1.75 0 1 cbr 500 ------- 1 0.0 2.0 330 422
d 1.76416 0 1 tcp 1040 ------- 2 0.1 2.1 55 427
d 1.77248 0 1 tcp 1040 ------- 2 0.1 2.1 56 430
d 1.7848 0 1 tcp 1040 -------▶2 0.1 2.1 57 433
d 1.785 0 1 cbr 500 ------- 1 0.0 2.0 337 434
d 1.8 0 1 cbr 500 ------- 1 0.0 2.0 340 439
d 1.825 0 1 cbr 500 ------- 1 0.0 2.0 345 447
d 1.85 0 1 cbr 500 ------- 1 0.0 2.0 350 454
d 1.88 0 1 cbr 500 ------- 1 0.0 2.0 356 463
d 1.89472 0 1 tcp 1040 ------- 2 0.1 2.1 63 467
d 1.915 0 1 cbr 500 ------- 1 0.0 2.0 363 472
d 3.205 0 1 cbr 500 ------- 1 0.0 2.0 621 793
d 3.21296 0 1 tcp 1040 ------- 2 0.1 2.1 76 796
d 3.22128 0 1 tcp 1040 ------- 2 0.1 2.1 77 800
d 3.24 0 1 cbr 500 ------- 1 0.0 2.0 628 807
d 3.255 0 1 cbr 500 ------- 1 0.0 2.0 631 813
d 3.48784 0 1 tcp 1040 ------- 2 0.1 2.1 86 868
d 3.51216 0 1 tcp 1040 ------- 2 0.1 2.1 87 876
user@user-ThinkCentre-M83:~$ ▯
```

Note:

➤ Set the queue size fixed from n0 to n2 as 10, n1-n2 to 10
   and from n2-n3 as 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.

```
set ns [new Simulator]
set f [open lab2.tr w]
set nf [open lab2.nam w]
$ns trace-all $f
$ns namtrace-all $nf

proc finish {} {
    global ns f nf
    $ns flush-trace
    close $f
    close $nf
    exec nam lab2.nam &
    exit 0
}

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]

$n0 label "ping0"
$n1 label "ping1"
$n2 label "R1"
$n3 label "R2"
$n4 label "ping4"
```

$n5 label "ping5"

$ns color 1 red
$ns color 2 blue
$ns color 3 green
$ns color 4 orange

$ns duplex-link $n0 $n2 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link $n2 $n3 1Mb 30ms DropTail
$ns duplex-link $n3 $n4 1Mb 10ms DropTail
$ns duplex-link $n3 $n5 1Mb 10ms DropTail
set ping0 [new Agent/Ping]
$ns attach-agent $n0 $ping0

set ping1 [new Agent/Ping]
$ns attach-agent $n1 $ping1

set ping4 [new Agent/Ping]
$ns attach-agent $n4 $ping4

set ping5 [new Agent/Ping]
$ns attach-agent $n5 $ping5

$ns connect $ping0 $ping4
$ns connect $ping1 $ping5

proc sendPingPacket {} {
        global ns ping0 ping1 ping4 ping5
        set intervalTime 0.001
        set now [$ns now]
        $ns at [expr $now + $intervalTime] "$ping0 send"
        $ns at [expr $now + $intervalTime] "$ping1 send"
        $ns at [expr $now + $intervalTime] "$ping4 send"
        $ns at [expr $now + $intervalTime] "$ping5 send"

```
        $ns at [expr $now + $intervalTime] "sendPingPacket"
}

Agent/Ping instproc recv {from rtt} {
        global seq
        $self instvar node_
        puts "The node [$node_ id] received an ACK from the node $from
with RTT $rtt ms"
}

$ping0  set class_  1
$ping1  set class_  2
$ping4  set class_  3
$ping5  set class_  4
$ns at 0.01 "sendPingPacket"
$ns at 10.0 "finish"
$ns run
```

Steps for execution

1) Open Text editor and type program. Program name should have
   the extension
" .tcl "

[root@localhost ~]$ gedit lab2.tcl

2) Run the simulation program

[root@localhost~]$ ns lab2.tcl

  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.

3) After simulation is completed **r**un grep command to find packets
    drop to see the output

[root@localhost~]$ grep ^d lab2.tr -c

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

[root@localhost~]$ vi lab2.tr

Topology
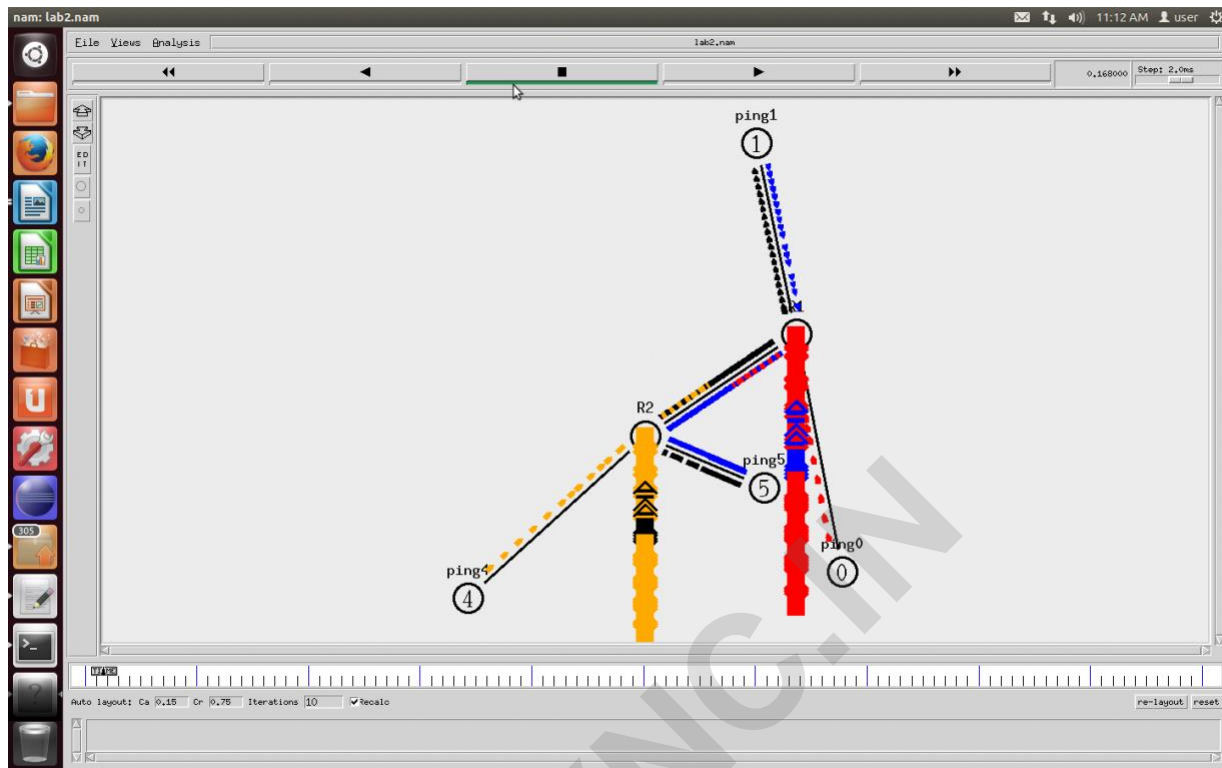
Output

```
The node 0 received an ACK from the node 4
with RTT 153.1 ms
The node 4 received an ACK from the node 0
with RTT 153.1 ms
The node 0 received an ACK from the node 4
with RTT 153.1 ms
The node 4 received an ACK from the node 0
with RTT 153.1 ms
The node 0 received an ACK from the node 4
with RTT 153.1 ms
The node 4 received an ACK from the node 0
with RTT 153.1 ms
The node 0 received an ACK from the node 4
with RTT 153.1 ms
The node 4 received an ACK from the node 0
with RTT 153.2 ms
The node 0 received an ACK from the node 4
with RTT 153.2 ms
The node 4 received an ACK from the node 0
with RTT 153.2 ms
The node 0 received an ACK from the node 4
with RTT 153.2 ms
The node 4 received an ACK from the node 0
with RTT 153.2 ms
The node 0 received an ACK from the node 4
with RTT 153.2 ms
The node 5 received an ACK from the node 1
with RTT 152.8 ms
The node 1 received an ACK from the node 5
with RTT 152.8 ms
The node 5 received an ACK from the node 1
with RTT 152.8 ms
The node 1 received an ACK from the node 5
with RTT 152.8 ms
The node 5 received an ACK from the node 1
with RTT 152.8 ms
The node 1 received an ACK from the node 5
with RTT 152.8 ms
The node 5 received an ACK from the node 1
with RTT 152.8 ms
The node 1 received an ACK from the node 5
with RTT 152.8 ms
The node 5 received an ACK from the node 1
with RTT 152.8 ms
The node 1 received an ACK from the node 5
with RTT 152.8 ms
The node 5 received an ACK from the node 1
```

```
user@user-ThinkCentre-M83:~$ grep ^d lab2.tr -c
22704
user@user-ThinkCentre-M83:~$ ▯
```

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 label "dest1"

```
$ns make-lan "$n0 $n1 $n2 $n3 $n4" 5Mb 100ms LL Queue/DropTail
Mac/802_3
$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_
$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\t%f\t\n",$1,$7);}          /* you must put \n in printf */
```

END {
}

Steps for execution

1) Open Text editor and type program. Program name should have the extension " **.tcl** "
   > **[root@localhost ~]$ gedit lab3.tcl**

2) Open Text editor and type **awk** program. Program name should have the extension "**.awk** "
   > **[root@localhost ~]$ gedit lab3.awk**

3) Run the simulation program
   > **[root@localhost~]$ ns lab3.tcl**

   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.

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

   > **[root@localhost~]# awk –f lab3.awk file1.tr > a1**
   > **[root@localhost~]# awk –f lab3.awk file2.tr > a2**
   > **[root@localhost~]# xgraph a1 a2**

   Here we are using the congestion window trace files i.e. **file1.tr** and **file2.tr** and we are redirecting the contents of those files to new files say **a1** and **a2** using **output redirection operator (>)**.

5) To see the trace file contents open the file as ,
   > **[root@localhost~]# gedit lab3.tr**

Topology
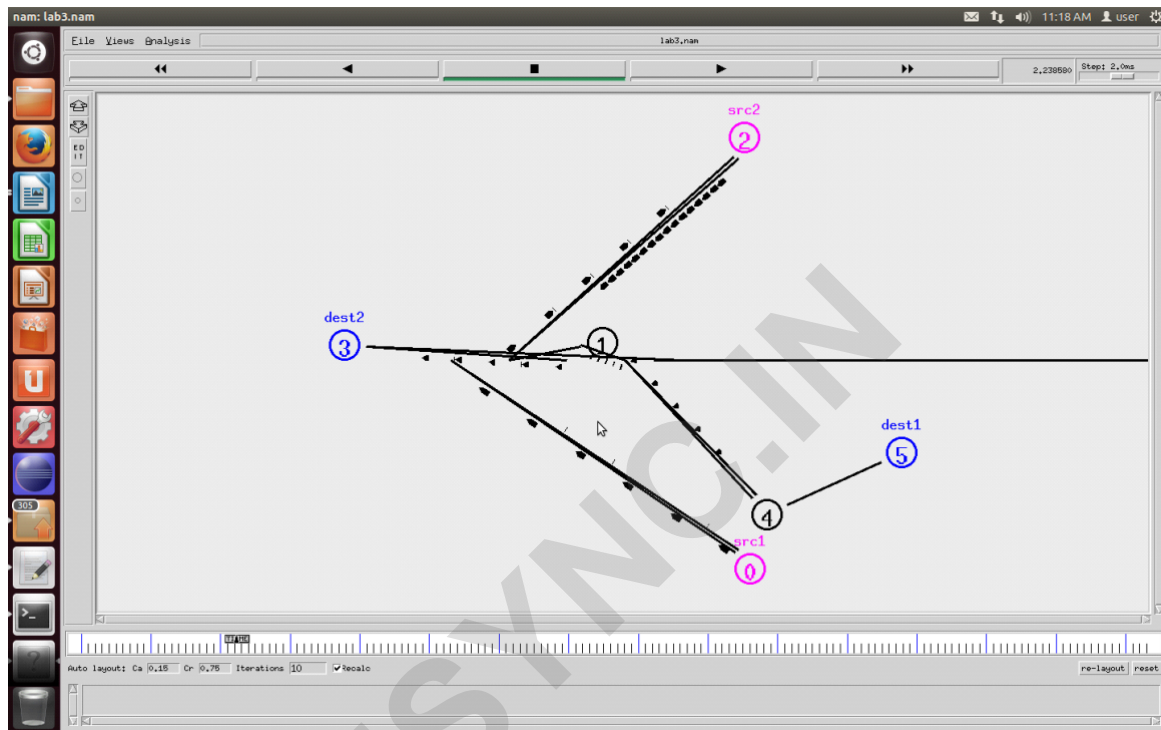
Output

```
user@user-ThinkCentre-M83:~$ ns lab3.tcl
warning: no class variable LanRouter::debug_

        see tcl-object.tcl in tclcl for info about this warning.

user@user-ThinkCentre-M83:~$ awk -f lab3.awk file1.tr > a1
user@user-ThinkCentre-M83:~$ awk -f lab3.awk file2.tr > a2
user@user-ThinkCentre-M83:~$ xgraph a1 a2
Parameter LabelFont: can't translate `helvetica-10' into a font (defaulting to `
fixed')
Parameter TitleFont: can't translate `helvetica-18' into a font (defaulting to `
fixed')
```

Xgraph
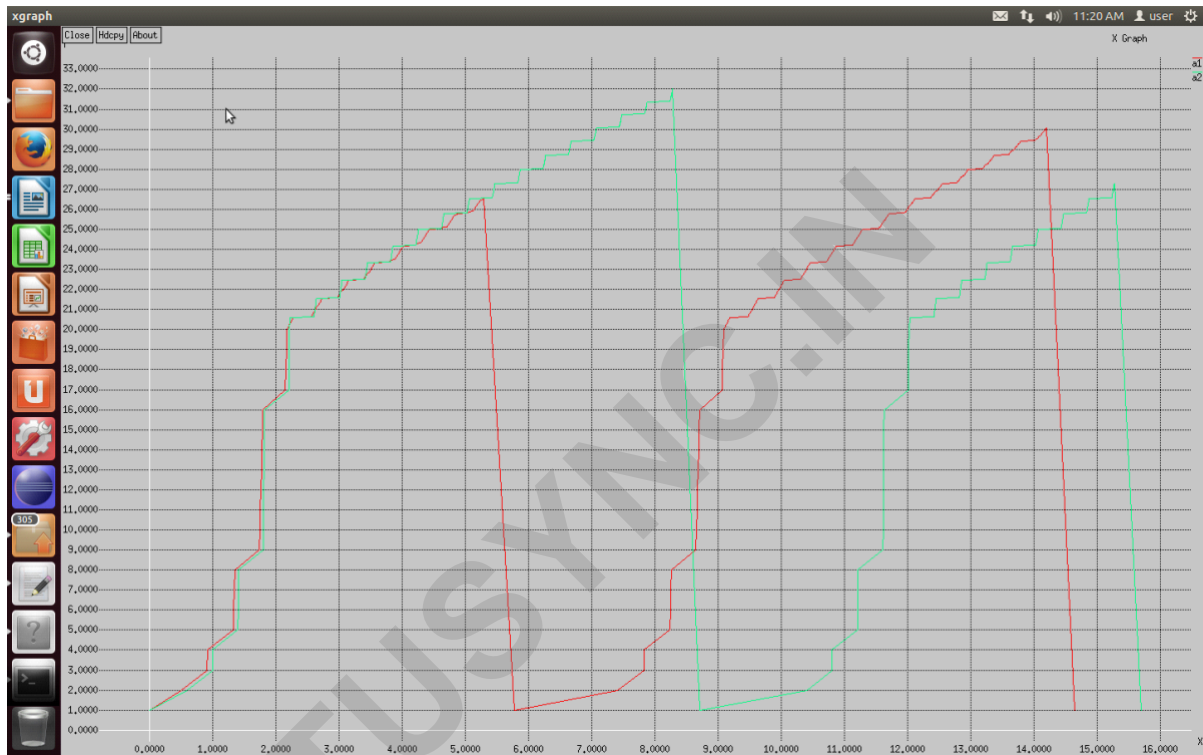
PART-B

Java is a general-purpose computer programming language that is simple, concurrent, class-based, object-oriented language. The compiled Java code can run on all platforms that support Java without the need for recompilation hence Java is called as "write once, run anywhere" (WORA).The Java compiled intermediate output called "byte-code" that can run on any Java virtual machine (JVM) regardless of computer architecture. The language derives much of its syntax from C and C++, but it has fewer low-level facilities than either of them.

In Linux operating system Java libraries are preinstalled. It's very easy and convenient to compile and run Java programs in Linux environment. To compile and run Java Program is a two-step process:

➢ Compile Java Program from Command Prompt
[root@host ~]# javac Filename.java

The Java compiler (Javac) compiles java program and generates a byte-code with the same file name and .class extension.

➢ Run Java program from Command Prompt
[root@host ~]# java Classname

The java interpreter (Java) runs the byte-code and gives the respective output. It is important to note that in above command we have omitted the .class suffix of the byte- code (Filename.class).

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	1000100000 0100001	1100000000 0000101	1000001001100000 1000 1110110110111

Name	Polynomial	Application
CRC-8	$x^8 + x^2 + x + 1$	ATM header
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x^2 + 1$	ATM AAL
CRC-16	$x^{16} + x^{12} + x^5 + 1$	HDLC
CRC-32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$	LANs

Source Code:

```java
import java.io.*;
class Crc
{
    public static void main(String args[]) throws IOException
    {
        BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
        int[ ] data;
        int[ ]div;
        int[ ]divisor;
        int[ ]rem;
        int[ ] crc;
        int data_bits, divisor_bits, tot_length;

        System.out.println("Enter number of data bits : ");
        data_bits=Integer.parseInt(br.readLine());
        data=new int[data_bits];

        System.out.println("Enter data bits : ");
        for(int i=0; i<data_bits; i++)
            data[i]=Integer.parseInt(br.readLine());

        System.out.println("Enter number of bits in divisor : ");
        divisor_bits=Integer.parseInt(br.readLine());
        divisor=new int[divisor_bits];
```

```
System.out.println("Enter Divisor bits : ");
for(int i=0; i<divisor_bits; i++)
        divisor[i]=Integer.parseInt(br.readLine());
tot_length=data_bits+divisor_bits-1;

div=new int[tot_length];
rem=new int[tot_length];
crc=new int[tot_length];
```

/*----------------- **CRC GENERATION**-------------------
                                                    -*/

```
for(int i=0;i<data.length;i++)
        div[i]=data[i];

System.out.print("Dividend (after appending 0's) are : ");

for(int i=0; i< div.length; i++)
        System.out.print(div[i]);
System.out.println();

for(int j=0; j<div.length; j++)
{
        rem[j] = div[j];
}

rem=divide(div, divisor, rem);

for(int i=0;i<div.length;i++)  //append dividend and
ramainder
        {
            crc[i]=(div[i]^rem[i]);
        }
System.out.println();
System.out.println("CRC code : ");
```

```
        for(int i=0;i<crc.length;i++)
                System.out.print(crc[i]);


  /*-----------------ERROR DETECTION--------------------*/
        System.out.println();
        System.out.println("Enter CRC code of "+tot_length+" bits :
");
        for(int i=0; i<crc.length; i++)
                crc[i]=Integer.parseInt(br.readLine());
        for(int j=0; j<crc.length; j++)
        {
                rem[j] = crc[j];
        }
        rem=divide(crc, divisor, rem);
        for(int i=0; i< rem.length; i++)
        {
                if(rem[i]!=0)
                {
                        System.out.println("Error");
                        break;
                }
                if(i==rem.length-1)
                        System.out.println("No Error");
        }
        System.out.println("THANK YOU.... :)");
    }

  static int[] divide(int div[],int divisor[], int rem[])
  {
        int cur=0;
        while(true)
        {
                for(int i=0;i<divisor.length;i++)
                        rem[cur+i]=(rem[cur+i]^divisor[i]);
```

```
                 while(rem[cur]==0 && cur!=rem.length-1)
                        cur++;
                 if((rem.length-cur)<divisor.length)
                        break;
          }
          return rem;
     }
}
```

Steps for execution

> Open Text editor and type program. Program name should have the extension " **.java** "
> **[root@localhost ~]$ gedit Crc.java**
> To Complie the program
> **[root@localhost ~]$ javac Crc.java**
> To Run the Program
> **[root@localhost ~]$ java Crc**

Output

```
user@user-ThinkCentre-M83:~$ javac Crc.java
user@user-ThinkCentre-M83:~$ java Crc
Enter number of data bits :
7
Enter data bits :
1
0
1
1
0
0
1
Enter number of bits in divisor :
3
Enter Divisor bits :
1
0
1
Dividend (after appending 0's) are : 101100100

CRC code :
101100111
Enter CRC code of 9 bits :
1
0
1
1
0
0
1
0
1
Error
THANK YOU.... :)
user@user-ThinkCentre-M83:~$ ▯
```

5.Write a simple program for Implementing Sliding window protocol in data link layer

The Sliding Window protocol is used in the Data Link layer (Layer 2) for efficient transmission of packets between two nodes. It helps manage flow control and ensures reliable delivery of data frames, especially in noisy networks.

Steps to Run the Program in Eclipse:

1. **Install Eclipse** (if not already installed):
   o Download and install Eclipse IDE from the official website.
2. **Launch Eclipse**:
   o Open Eclipse and choose a workspace (the folder where your projects will be saved).
3. **Create a New Java Project**:
   o Click on File > New > Java Project.
   o Enter a project name (e.g., SlidingWindowProject).
   o Click Finish.
4. **Create a New Java Class**:
   o Right-click on the src folder in the Project Explorer panel.
   o Click New > Class.
   o Enter a class name (e.g., SlidingWindowProtocol).
   o Check the box for public static void main(String[] args) to generate the main method.
   o Click Finish.

Write the Java Program:

- import java.util.Scanner;
-
- public class SlidingWindowProtocol {
- public static void main(String[] args) {
- Scanner scanner = new Scanner(System.in);
-

```
•          // Get window size
•          System.out.print("Enter the window size: ");
•          int windowSize = scanner.nextInt();
•
•          // Get number of frames
•          System.out.print("Enter the number of frames to send: ");
•          int totalFrames = scanner.nextInt();
•
•          // Send frames using sliding window
•          int sentFrames = 0;
•          while (sentFrames < totalFrames) {
•             // Sending frames within the window
•             for (int i = 0; i < windowSize && sentFrames <
    totalFrames; i++) {
•                 System.out.println("Frame " + (sentFrames + 1) + "
    sent.");
•                 sentFrames++;
•             }
•
•             // Simulating acknowledgment for frames sent
•             System.out.print("Enter the number of frames
    acknowledged: ");
•             int ack = scanner.nextInt();
•
•             // Adjust window based on acknowledgments
•             sentFrames -= (windowSize - ack); // Sliding window
    forward
•
•             if (sentFrames < totalFrames) {
•                 System.out.println("Sliding window...");
•             }
•          }
•
•          System.out.println("All frames sent successfully!");
•          scanner.close();
```

- }
- }

☐ **Save the Program**:

- Save the file by clicking Ctrl + S (Windows/Linux) or Cmd + S (Mac).

☐ **Run the Program**:

- Right-click on the file in the Project Explorer panel.
- Select Run As > Java Application.

☐ **Interact with the Program**:

- The program will run in the Eclipse Console, and you can provide input when prompted (e.g., window size, number of frames, and acknowledgments).

  Output with screenshot :

```
Enter the window size: 3
Enter the number of frames to send: 5
Frame 1 sent.
Frame 2 sent.
Frame 3 sent.
Enter the number of frames acknowledged: 2
Sliding window...
```

6). Write a program to find the shortest path between vertices using bellman-ford 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 DijkstraHYPERLINK "https://en.wikipedia.org/wiki/Dijkstra's_algorithm"'HYPERLINK "https://en.wikipedia.org/wiki/Dijkstra's_algorithm"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:

```java
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();
    }
}
```

Steps for execution

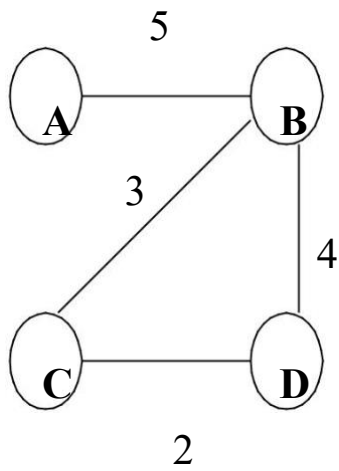1) Open Text editor and type program. Program name should have the extension " **.java** "
    [root@localhost ~]$ gedit BellmanFord .java

2) To Complie the program
    [root@localhost ~]$ javac BellmanFord .java

3) To Run the Program
    [root@localhost ~]$ java BellmanFord

Input graph:

Output:

```
user@user-ThinkCentre-M83:~$ javac BellmanFord.java
user@user-ThinkCentre-M83:~$ java BellmanFord
Enter the number of vertices
4
Enter the adjacency matrix
0 5 0 0
5 0 3 4
0 3 0 2
0 4 2 0
Enter the source vertex
2
distance of source 2 to 1is 5
distance of source 2 to 2is 0
distance of source 2 to 3is 3
distance of source 2 to 4is 4
user@user-ThinkCentre-M83:~$ 
```

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

Java DatagramSocket and DatagramPacket :

Java DatagramSocket and DatagramPacket classes are used for connection-less socket programming using the UDP instead of TCP.

Datagram

Datagrams are collection of information sent from one device to another device via the established network. When the datagram is sent to the targeted device, there is no assurance that it will reach to the target device safely and completely. It may get damaged or lost in between. Likewise, the receiving device also never know if the datagram received

is damaged or not. The UDP protocol is used to implement the datagrams in Java.

Java DatagramSocket class

Java DatagramSocket class represents a connection-less socket for sending and receiving datagram packets. It is a mechanism used for transmitting datagram packets over network.`

A datagram is basically an information but there is no guarantee of its content, arrival or arrival time.

Commonly used Constructors of DatagramSocket class

- ○ DatagramSocket() throws SocketEeption: it creates a datagram socket and binds it with the available Port Number on the localhost machine.

- ○ DatagramSocket(int port) throws SocketEeption: it creates a datagram socket and binds it with the given Port Number.

- ○ DatagramSocket(int port, InetAddress address) throws SocketEeption: it creates a datagram socket and binds it with the specified port number and host address.

Java DatagramPacket Class

Java DatagramPacket is a message that can be sent or received. It is a data container. If you send multiple packet, it may arrive in any order. Additionally, packet delivery is not guaranteed.

Commonly used Constructors of DatagramPacket class

- ○ **DatagramPacket(byte[] barr, int length):** it creates a datagram packet. This constructor is used to receive the packets.

○ **DatagramPacket(byte[] barr, int length, InetAddress address, int port):** it creates a datagram packet. This constructor is used to send the packets.

Writing a Datagram Client and Server

The example featured in this section consists of two applications: a client and a server. The server continuously receives datagram packets over a datagram socket. Each datagram packet received by the server indicates a client request for a quotation. When the server receives a datagram, it replies by sending a datagram packet that contains a one-line "quote of the moment" back to the client.

The client application in this example is fairly simple. It sends a single datagram packet to the server indicating that the client would like to receive a quote of the moment. The client then waits for the server to send a datagram packet in response.

Serverside:

```
import java.util.*;

import java.net.*;

import java.io.*;

public class udpserver

{

public static void main(String args[])

{

DatagramSocket aSocket=null;

Scanner scan=new Scanner(System.in);
```

```
int ServerPort=999;

System.out.println("Server ready  \n Waiting for connection\n");

 try

  {

aSocket=new DatagramSocket(ServerPort);

byte[] buffer=new byte[1000];

byte[] buff=new byte[1000];

DatagramPacket data1=new DatagramPacket(buff,buff.length);

aSocket.receive(data1);

byte[] msg=new byte[1000];

msg=data1.getData();

System.out.println(new String(msg,0,data1.getLength()));

 System.out.println("\nEnter message to be sent:");

String str=scan.nextLine();

buffer=str.getBytes();

DatagramPacket data=new
DatagramPacket(buffer,buffer.length,InetAddress.getLocalHost(),998);

aSocket.send(data);

}

 catch(SocketException e)

 {              System.out.println("Socket "+e.getMessage());

}

catch(IOException o)
```

```
{          System.out.println("IO :"+o.getMessage());

}

finally

{              System.out.println("\nMessage sent\nconnection terminated\n");

   if(aSocket!=null)

aSocket.close();

scan.close();

}

}

}
```

ClientSide:

```
import java.net.*;

import java.io.*;

public class udpclient


{

public static void main(String args[])

{

DatagramSocket aSocket=null;

int clientPort=998;

try{

aSocket=new DatagramSocket(clientPort);
```

```
byte[] buf=new byte[1000];

byte[] buf1=new byte[1000];

DatagramPacket data=new DatagramPacket(buf,buf.length);

String conf="connected to client";

buf1=conf.getBytes();

DatagramPacket data1=new
DatagramPacket(buf1,buf1.length,InetAddress.getLocalHost(),999);


aSocket.send(data1);


System.out.println("connected to server");


aSocket.receive(data);


byte[] msg=new byte[1000];


msg=data.getData();


System.out.println("\n message:");


System.out.println(new String(msg,0,data.getLength()));


}
```

```
catch(SocketException e)

{

System.out.println("Socket:"+e.getMessage());

}

catch(IOException e)

{

System.out.println("IO:"+e.getMessage());

}

finally

{

if(aSocket!=null)

aSocket.close();

}
```

}

}

Steps for execution

1) Open Text editor and type program. Program name should have the extension   " **.java** "
   udpserver.java

   updclient.java

2) Open the command prompt:
   Compile the serverside program
   javac udpserver.java

   Run the serverside program

   java udpserver

3) Open the new command prompt:
   Compile the clientside program
   javac udpclient.java

   Run the udpclient program

   java udpserver

4)pass the msg from serverside to clientside

Output:

ServerSide

```
C:\>cd program files

C:\Program Files>cd java

C:\Program Files\Java>cd jdk1.6.0_45

C:\Program Files\Java\jdk1.6.0_45>cd bin

C:\Program Files\Java\jdk1.6.0_45\bin>cd Clientpgr

C:\Program Files\Java\jdk1.6.0_45\bin\Clientpgr>javac udpserver.java

C:\Program Files\Java\jdk1.6.0_45\bin\Clientpgr>java udpserver
Server ready
 Waiting for connection

connected to client

Enter message to be sent:
Hello,hi......

Message sent
connection terminated


C:\Program Files\Java\jdk1.6.0_45\bin\Clientpgr>_
```

ClientSide:

```
C:\>cd program files

C:\Program Files>cd java

C:\Program Files\Java>cd jdk1.6.0_45

C:\Program Files\Java\jdk1.6.0_45>cd bin

C:\Program Files\Java\jdk1.6.0_45\bin>cd Clientpgr

C:\Program Files\Java\jdk1.6.0_45\bin\Clientpgr>javac udpclient.java

C:\Program Files\Java\jdk1.6.0_45\bin\Clientpgr>java udpclient
connected to server

 message:
Hello,hi......

C:\Program Files\Java\jdk1.6.0_45\bin\Clientpgr>
```

8.Using tcp/ip sockets write a client server program to make the client send the filename and to make the server send back the contents of the requested file if present.

Sockets in JAVA

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.

ServerSide:

```
import java.net.*;

import java.io.*;

public class ContentsServer

{

  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 and wating for client request");


    // 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();

}

}

ClientSide:

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

```java
public static void main( String args[ ] ) throws Exception
{
    Socket sock = new Socket("10.0.68.4", 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();

}

}

Output:

Steps for execution

1) Open Text editor and type program. Program name should have the extension   " **.java** "
   ContentsServer.java

   ContentsClient.java

2) Create 1 txt file….with content

   abc.txt

3) Open the command prompt:
    Compile the serverside program
   javac ContentsServer.java

   Run the serverside program

   java ContentsServer

) Open the new command prompt:
    Compile the clientside program
   javac ContentsClient.java

Run the udpclient program

java ContentsClient

4)Enter the file name and read the content

ServerSide:

```
C:\>cd program files

C:\Program Files>cd java

C:\Program Files\Java>cd jdk1.6.0_45

C:\Program Files\Java\jdk1.6.0_45>cd bin

C:\Program Files\Java\jdk1.6.0_45\bin>cd fileprg

C:\Program Files\Java\jdk1.6.0_45\bin\fileprg>javac ContentsServer.java

C:\Program Files\Java\jdk1.6.0_45\bin\fileprg>java ContentsServer
Server ready for connection
Connection is successful and wating for client request

C:\Program Files\Java\jdk1.6.0_45\bin\fileprg>
```

ClientSide :

```
C:\>cd program files

C:\Program Files>cd java

C:\Program Files\Java>cd jdk1.6.0_45

C:\Program Files\Java\jdk1.6.0_45>cd bin

C:\Program Files\Java\jdk1.6.0_45\bin>cd fileprg

C:\Program Files\Java\jdk1.6.0_45\bin\fileprg>javac ContentsClient.java

C:\Program Files\Java\jdk1.6.0_45\bin\fileprg>java ContentsClient
Enter the file nameabc.txt
hi hellooooo ,Computer network

C:\Program Files\Java\jdk1.6.0_45\bin\fileprg>
```

9) Develop a program for a sample RSA Algorithm to Encrypt and Decrypt the data

The **RSA algorithm** (Rivest–Shamir–Adleman) is a widely used public-key cryptosystem for secure data transmission. It is based on the mathematical properties of large prime numbers and is used for encrypting and decrypting messages as well as for digital signatures.

Key Concepts:

1. **Public and Private Key Pair**:
   - **Public Key**: Used for encryption and is shared with everyone.
   - **Private Key**: Used for decryption and is kept secret by the owner.

Steps in RSA Algorithm:

1. **Key Generation**:
   - Choose two large prime numbers p and q.
   - Compute $n = p \times q$. The number n is used as part of both the public and private keys.
   - Compute the **totient** $\varphi(n) = (p - 1) \times (q - 1)$.
   - Choose an integer e such that $1 < e < \varphi(n)$ and $gcd(e, \varphi(n)) = 1$ (i.e., e is co-prime with $\varphi(n)$). This e is the public exponent.
   - Compute d as the modular multiplicative inverse of e modulo $\varphi(n)$, i.e., $d \equiv e^{-1} \pmod{\varphi(n)}$. This d is the private exponent.
2. **Public Key**: The pair (e, n).
3. **Private Key**: The pair (d, n).

Encryption:

To encrypt a message M, convert M into an integer such that M < n, then: C=Memod nC = M^e \mod nC=Memodn where C is the encrypted message (ciphertext).

Decryption:

To decrypt the ciphertext C, use the private key (d, n): M=Cdmod nM = C^d \mod nM=Cdmodn This recovers the original message M.

Security:

- The security of RSA relies on the difficulty of factoring large numbers. While n is public, factoring it into p and q (its prime factors) is computationally challenging, which ensures the security of the private key.

RSA is used in a variety of applications, including securing web traffic (HTTPS), email encryption, and digital signatures.

Steps for Execution:

Steps to Implement RSA Algorithm in Eclipse:

1. Set up Eclipse and Create a New Project:

- Open **Eclipse IDE**.
- Go to **File** > **New** > **Java Project**.
- Give your project a name (e.g., RSAAlgorithm) and click **Finish**.

2. Create a New Java Class:

- Right-click on the src folder in your project.
- Select **New** > **Class**.
- Name your class (e.g., RSA) and check the box for public static void main(String[] args) if you want to run this as a standalone program. Click **Finish**.

Write the RSA Algorithm Code:

Copy the following RSA code into your RSA.java class:

```java
package mat;

import java.math.BigInteger;

import java.security.KeyFactory;

import java.security.KeyPair;

import java.security.KeyPairGenerator;

import java.security.PrivateKey;

import java.security.PublicKey;

import java.security.spec.RSAPrivateKeySpec;

import java.security.spec.RSAPublicKeySpec;

import java.util.Scanner;

import javax.crypto.Cipher;


public class rsaalgo {


    public static void main(String[] args)throws Exception

    {

            // Step 1: Generate RSA key pair
```

```java
KeyPairGenerator keyPairGen =
KeyPairGenerator.getInstance("RSA");

keyPairGen.initialize(2048); // Key size

KeyPair keyPair = keyPairGen.generateKeyPair();

PublicKey publicKey = keyPair.getPublic();

PrivateKey privateKey = keyPair.getPrivate();


// Get user input

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the text to encrypt: ");

String inputText = scanner.nextLine();


// Step 2: Encrypt the input text

Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");

cipher.init(Cipher.ENCRYPT_MODE, publicKey);

byte[] encryptedText = cipher.doFinal(inputText.getBytes());

System.out.println("Encrypted Text: " + new
String(encryptedText));


// Step 3: Decrypt the text

cipher.init(Cipher.DECRYPT_MODE, privateKey);
```

byte[] decryptedText = cipher.doFinal(encryptedText);

System.out.println("Decrypted Text: " + new
String(decryptedText));

scanner.close();

}   // TODO Auto-generated method stub

}

4. Save the File:

- Save the RSA.java file.
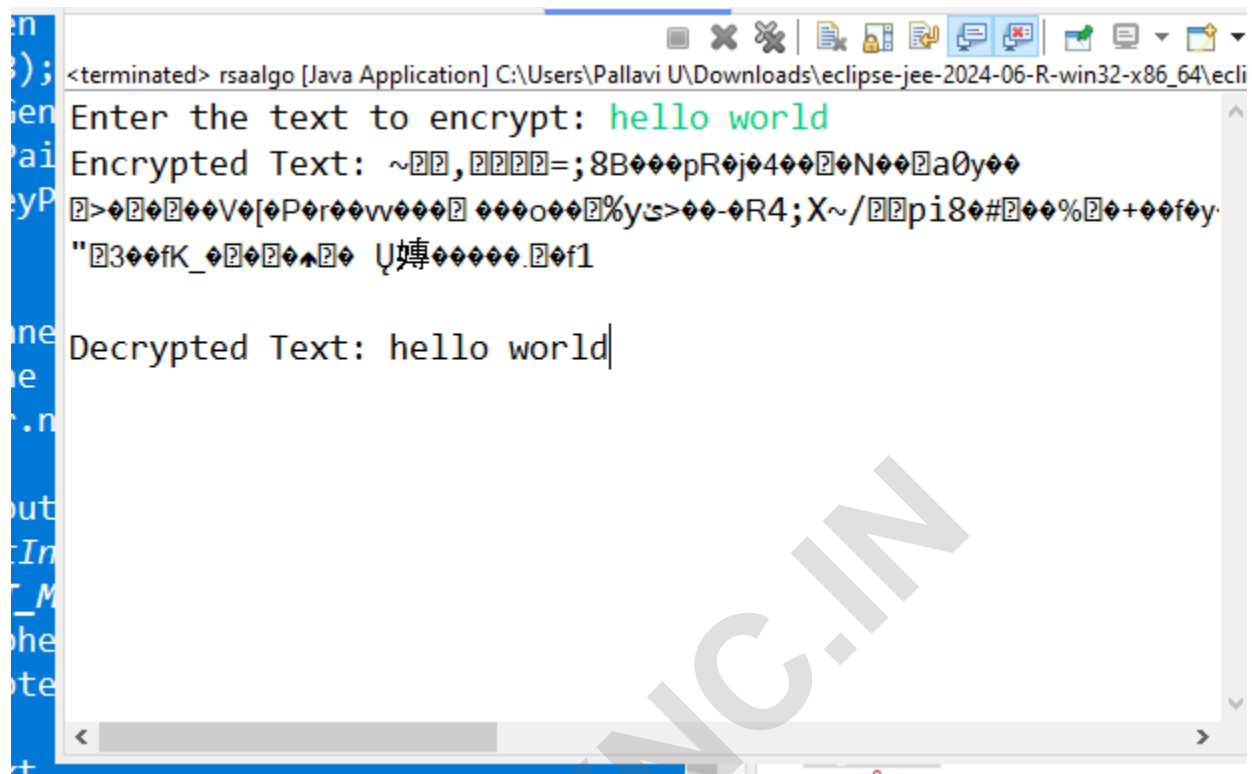
5. Run the Program:

- Right-click on the class file (e.g., RSA.java).
- Select **Run As** > **Java Application**.
- The program will compile and execute.

6. Input and Output:

- When prompted, enter a message (like "12345") for encryption.
- The console will display the **encrypted** and **decrypted** messages.

Output and screenshots

```
<terminated> rsaalgo [Java Application] C:\Users\Pallavi U\Downloads\eclipse-jee-2024-06-R-win32-x86_64\ecli
Enter the text to encrypt: hello world
Encrypted Text: ~□□,□□□□=;8B♦♦♦pR♦j♦4♦♦□♦N♦♦□a0y♦♦
□>♦□♦□♦♦V♦[♦P♦r♦♦vv♦♦♦□ ♦♦♦o♦♦□%y:s>♦♦-♦R4;X~/□□pi8♦#□♦♦%□♦+♦♦f♦y·
"□3♦♦fK_♦□♦□♦♠□♦ U嫭♦♦♦♦♦.□♦f1


Decrypted Text: hello world
```

10. Write a program for congestion control using 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)).

(a) A leaky bucket with water

(b) A leaky bucket with packets
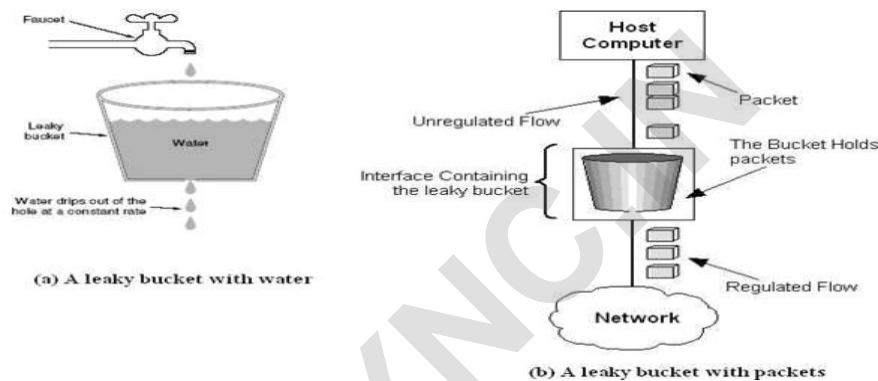
While leaky bucket eliminates completely bursty traffic by regulating the incoming data flow its main drawback is that it drops packets if the bucket is full. Also, it doesn't take into

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:

```java
import java.io.*;
import java.util.*;

class Queue
{
        int q[],f=0,r=0,size;

        void insert(int n)
        {
                Scanner in = new Scanner(System.in);
                q=new int[10];
                for(int i=0;i<n;i++)
                {
                        System.out.print("\nEnter " + i + " element: ");
                        int ele=in.nextInt();
                        if(r+1>10)
                        {
                                System.out.println("\nQueue is full \nLost Packet: "+ele); break;
                        }
                        else
                        {
                                r++;
                                q[i]=ele;
                        }
                }
        }

        void delete()
        {
                Scanner in = new Scanner(System.in);
                Thread t=new Thread();
```

```
                if(r==0)
                        System.out.print("\nQueue empty ");
                else
                {
                        for(int i=f;i<r;i++)
                        {
                                try
                                {
                                        t.sleep(1000);
                                }
                                catch(Exception e){}
                                System.out.print("\nLeaked Packet: "+q[i]);
                                f++;
                        }
                }
                System.out.println();
        }
}

class leaky extends Thread
{
        public static void main(String ar[]) throws Exception
        {
                Queue q=new Queue();
                Scanner src=new Scanner(System.in);
                System.out.println("\nEnter the packets to be sent:");
                int size=src.nextInt();
                q. insert(size);
                q.delete();
        }
}
```

Steps for execution

1) Open Text editor and type program. Program name should have the extension   " **.java** "
    [root@localhost ~]$ gedit leaky.java

2) To Complie the program
    [root@localhost ~]$ javac leaky.java

3) To Run the Program
    [root@localhost ~]$ java leaky

Output:

Queue Overflow

```
user@user-ThinkCentre-M83:~$ javac leaky.java
user@user-ThinkCentre-M83:~$ java leaky

Enter the packets to be sent:
11

Enter 0 element: 1

Enter 1 element: 2

Enter 2 element: 3

Enter 3 element: 4

Enter 4 element: 5

Enter 5 element: 6

Enter 6 element: 7

Enter 7 element: 8

Enter 8 element: 9

Enter 9 element: 10

Enter 10 element: 11
Queue is full
Lost Packet: 11

Leaked Packet:  1
Leaked Packet:  2
Leaked Packet:  3
Leaked Packet:  4
Leaked Packet:  5
Leaked Packet:  6
Leaked Packet:  7
Leaked Packet:  8
Leaked Packet:  9
Leaked Packet:  10
user@user-ThinkCentre-M83:~$ ▯
```

Queue Empty

```
user@user-ThinkCentre-M83:~$ javac leaky.java
user@user-ThinkCentre-M83:~$ java leaky

Enter the packets to be sent:
0

Queue empty
user@user-ThinkCentre-M83:~$
```

Queue-Size

```
user@user-ThinkCentre-M83:~$ javac leaky.java
user@user-ThinkCentre-M83:~$ java leaky

Enter the packets to be sent:
10

Enter 0 element: 1

Enter 1 element: 2

Enter 2 element: 3

Enter 3 element: 4

Enter 4 element: 5

Enter 5 element: 6

Enter 6 element: 7

Enter 7 element: 8

Enter 8 element: 9

Enter 9 element: 10

Leaked Packet: 1
Leaked Packet: 2
Leaked Packet: 3
Leaked Packet: 4
Leaked Packet: 5
Leaked Packet: 6
Leaked Packet: 7
Leaked Packet: 8
Leaked Packet: 9
Leaked Packet: 10
user@user-ThinkCentre-M83:~$
```

VIVA QUESTIONS

1) What is a Link?

A link refers to the connectivity between two devices. It includes the type of cables and protocols used in order for one device to be able to communicate with the other.

2) What are the layers of the OSI reference model?

There are 7 OSI layers: Physical Layer, Data Link Layer, Network Layer, Transport Layer,
Session Layer, Presentation Layer and Application Layer.

3) What is backbone network?

A backbone network is a centralized infrastructure that is designed to distribute different routes and data to various networks. It also handles management of bandwidth and various channels.

4) What is a LAN?

LAN is short for Local Area Network. It refers to the connection between computers and other network devices that are located within a small physical location.

5) What is a node?

A node refers to a point or joint where a connection takes place. It can be computer or device that is part of a network. Two or more nodes are needed in order to form a network connection.

6) What are routers?

Routers can connect two or more network segments. These are intelligent network devices that store information in its routing table such as paths, hops and bottlenecks. With this info, they are able to determine the best path for data transfer. Routers operate at the OSI Network Layer.

7) What is point to point link?

It refers to a direct connection between two computers on a network. A point to point connection does not need any other network devices other than connecting a cable to the NIC cards of both computers.

8) What is data encapsulation?

Data encapsulation is the process of breaking down information into smaller manageable
chunks before it is transmitted across the network. It is also in this process that the source and destination addresses are attached into the headers, along with parity checks.

9) Describe Network Topology

Network Topology refers to the layout of a computer network. It shows how devices and cables are physically laid out, as well as how they connect to one another.

10) What is VPN?

VPN means Virtual Private Network, a technology that allows a secure tunnel to be created
across a network such as the Internet. For example, VPNs allow you to establish a secure dialup connection to a remote server.

11) How does a network topology affect your decision in setting up a network?

Network topology dictates what media you must use to interconnect devices. It also serves as basis on what materials, connector and terminations that is applicable for the setup.

12) What are different ways of securing a computer network?

There are several ways to do this. Install reliable and updated anti-virus program on all
computers. Make sure firewalls are setup and configured properly. User authentication will also help a lot. All of these combined would make a highly secured network.

13) What is NIC?

NIC is short for Network Interface Card. This is a peripheral card that is attached to a PC in
order to connect to a network. Every NIC has its own MAC address that identifies the PC on the network.

14) What is WAN?

WAN stands for Wide Area Network. It is an interconnection of computers and devices that are geographically dispersed. It connects networks that are located in different regions and
 countries.

15) How many layers are there under TCP/IP?

 There are four layers: the Network Layer, Internet Layer, Transport

 Layer and Application Layer.

16) What are gateways?

Gateways provide connectivity between two or more network segments. It is usually a computer that runs the gateway software and provides translation services. This translation is a key in allowing different systems to communicate on the network.

17) What is Hybrid Network?

A hybrid network is a network setup that makes use of both client-server and peer-to-peer
architecture.

18) What is TCP/IP?

TCP/IP is short for Transmission Control Protocol / Internet Protocol. This is a set of protocol
layers that is designed to make data exchange possible on different types of computer
 networks, also known as heterogeneous network.

20) What is the difference between a hub and a switch?

A hub acts as a multiport repeater. However, as more and more devices connect to it, it would not be able to efficiently manage the volume of traffic that passes through it.

A switch provides a better alternative that can improve the performance especially when high traffic volume is expected across all ports.

21) What is client/server?

Client/server is a type of network wherein one or more computers act as servers. Servers
provide a centralized repository of resources such as printers and files. Clients refers to
workstation that access the server.

22) Describe networking.

Networking refers to the inter connection between computers and peripherals for data
 communication. Networking can be done using wired cabling or through

 wireless link.

23)Describe Ethernet.

Ethernet is one of the popular networking technologies used these days. It was developed
during the early 1970s and is based on specifications as stated in the IEEE. Ethernet is used in local area networks.

24) What is the difference between CSMA/CD and CSMA/CA?

CSMA/CD, or Collision Detect, retransmits data frames whenever a collision occurred. CSMA/CA, or Collision Avoidance, will first broadcast intent to send prior to data transmission.

25) What is the importance of Encryption on a network?

Encryption is the process of translating information into a code that is unreadable by the user. It is then translated back or decrypted back to its normal readable format using a secret key or password. Encryption help ensure that information that is intercepted halfway would remain unreadable because the user has to have the correct password or key for it.

26) What is RSA algorithm?

RSA is short for Rivest-Shamir-Adleman algorithm. It is the most commonly used public key
encryption algorithm in use today.

27) What are the various types of key used in cryptography ?

Ans: here are two main types of cryptography:

- Secret key cryptography
- Public key cryptography

Secret key cryptography is also known as symmetric key cryptography. With this type of cryptography, both the sender and the receiver know the same secret code, called the key. Messages are encrypted by the sender using the key and decrypted by the receiver using the same key.
Public key cryptography also called asymmetric encryption, uses a pair of keys for encryption and decryption. With public key cryptography, keys work in pairs of matched public and private keys.

28) What is OSI and what role does it play in computer networks?

OSI (Open Systems Interconnect) serves as a reference model for data communication. It is
made up of 7 layers, with each layer defining a particular aspect on how network devices connect and communicate with one another. One layer

may deal with the physical media used, while another layer dictates how data is actually transmitted across the network.

29) What are MAC addresses?

MAC, or Media Access Control, uniquely identifies a device on the network. It is also known as physical address or Ethernet address. A MAC address is made up of 6-byte parts.

30) What is the equivalent layer or layers of the TCP/IP Application layer in terms of OSI reference model?

The TCP/IP Application layer actually has three counterparts on the OSI model: the Session
layer, Presentation Layer and Application Layer.

REFERENCES

➢ https://www.isi.edu/nsnam/ns/

➢ http://aknetworks.webs.com/e-books

➢ **Communication Networks: Fundamental Concepts and Key Architectures -** Alberto

  Leon, Garcia and Indra Widjaja, 4th Edition, Tata McGraw-Hill, reprint-2012.

➢ **Data and Computer Communication,** William
   Stallings, 8th ᵗEdition, Pearson Education, 2009.

➢ **Computer Networks: A Systems Approach -** Larry L.
   Peterson and Bruce S. David, 4th Edition, Elsevier, 2009.

➢ **Introduction to Data Communications and Networking**
   – Wayne Tomasi, Pearson Education, 2009.

➢ **Communication Networks – Fundamental Concepts and
   Key architectures –** Alberto Leon- Garcia and Indra
   Widjaja:, 2rd Edition, Tata McGraw-Hill, 2009

➢ **Computer and Communication Networks – Nader F.
   Mir:, Pearson Education, 2012**