BANGALORE TECHNOLOGICAL INSTITUTE

(An ISO 9001:2015 Certified Institution)

Kodathi Village,VarthoorHobli, Bangalore East Tq, Bangalore Urban District, Bangalore-560035, Karnataka

principal@btibangalore.org www.btibangalore.org

Phone: 7090404050

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

(BCS502) COMPUTER NETWORK LAB MANUAL

SEMESTER –V -(2025–26)

PREPARED BY:

Mrs.Manjupriya S

BANGALORE TECHNOLOGICAL INSTITUTE

(An ISO 9001:2015 Certified Institution)

Kodathi Village,VarthoorHobli, Bangalore East Tq, Bangalore

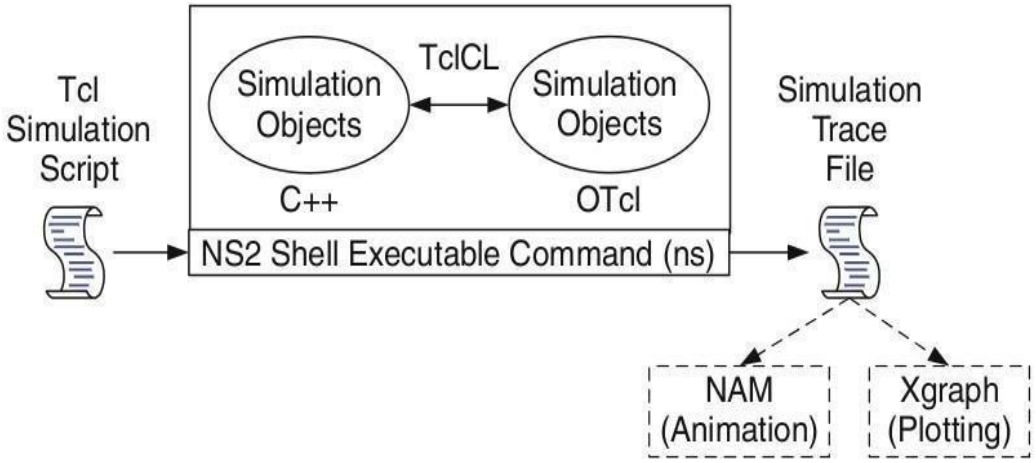
Urban District, Bangalore-560035, Karnataka

TABLE OF CONTENTS

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 usingNS2.
- 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 std out{Hello,
  World!} Hello, World!

- **Variables** Command Substitution

  seta5 set len [string lengthfoobar]

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

- **SimpleArithmetic**

  expr 7.2 / 4

- **Procedures**

```
proc Diag {a b} {
set c [expr sqrt($a * $a + $b * $b)]
return $c }
puts—Diagonalofa3,4righttriangleis[Diag34]‖ Output:
Diagonal of a 3, 4 right triangle is5.0
```

- **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.

2. Initialization and termination aspects of the nssimulator.
3. Definition of network nodes, links, queues andtopology.
4. Definition of agents and ofapplications.
5. The nam visualizationtool.
6. Tracing and randomvariables.

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
```

Theabovecreatesatracefilecalled―out.tr‖andanamvisualizationtracefilecalled ―out.nam‖. Within the tcl script, these files are not called explicitly by their names, but instead bypointersthatare declaredaboveandcalled―tracefile1‖and―namfile‖respectively. Remarkthattheybeginswitha#symbol.Thesecondlineopenthefile―out.tr‖ tobeusedfor writing,declaredwiththeletter―w‖.Thethirdlineusesasimulatormethodcalledtrace-all that have as parameter the name of the file where the traces willgo.

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‖.

Thetermination ofthe program isdoneusinga―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 traceson the respective files. The tcl command**—close"**closesthetracefilesdefinedbeforeand**exec**executesthenamprogramfor 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 somethingfails.

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 allowsus to schedule events explicitly.

The simulation can then begin using the command

```
$ns run
```

Definition of a network of links and nodes

The way to define a node is

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

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

Once we define several nodes, we can define the links that connect them. An example of a definition of a link is:

```
$ns duplex-link $n0 $n2 10Mb 10ms DropTail
```

Which means that $n0 and $n2 are connected using a bi-directional link that has 10ms of propagation delay and a capacity of 10Mb per sec for each direction.

Todefine adirectionallinkinsteadofabi-directionalone,we shouldreplace―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 EarlyDiscard) 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 usethem.

FTP over TCP

TCP is a dynamic reliable congestion control protocol. It uses Acknowledgements created by the destination to know whether packets are well received.

There are number variants of the TCP protocol, such as Tahoe, Reno, NewReno, Vegas. The type of agent appears in the first line:

```
set tcp [new Agent/TCP]
```

The command **$ns attach-agent $n0 $tcp** defines the source node of the tcp connection.
The command

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

Defines the behavior of the destination node of TCP and assigns to it a pointer called sink.

#Setup a UDP connection

```
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n5 $null
$ns connect $udp $null
$udp set fid_2
```

#setup a CBR over UDP connection

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

Above shows the definition of a CBR application using a UDP agent

The command **$ns attach-agent $n4 $sink** defines the destination node. The command **$ns connect $tcp $sink** finally makes the TCP connection between the source and destination nodes.

TCP has many parameters with initial fixed defaults values that can be changed if mentioned explicitly. For example, the default TCP packet size has a size of 1000bytes.This can be changed to another value, say 552bytes, using the command **$tcp set packetSize_ 552**.

When we have several flows, we may wish to distinguish them so that we can identify them with different colors in the visualization part. This is done by the command **$tcp set fid_1**thatassignstotheTCPconnectionaflowidentificationof—1‖.Weshalllatergivethe flowidentification 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 theformat:

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

The scheduler is started when running ns that is through the command $ns run.

The beginning and end of the FTP and CBR application can be done through the following command

```
$ns at 0.1 "$cbr start"

$ns at 1.0 " $ftp start"

$ns at 124.0 "$ftp stop"

$ns at 124.5 "$cbr stop"
```

Structure of Trace Files

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

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

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.
2. The second field gives the time at which the eventoccurs.
3. Gives the input node of the link at which the eventoccurs.
4. Gives the output node of the link at which the eventoccurs.
5. Gives the packet type (eg CBR orTCP)
6. Gives the packetsize
7. Someflags
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 NAMdisplay.

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)

Thisis the unit name forthe x-axis. Its default is ―X‖.

/-y <unit name> (YunitText)

This is the unit name forthey-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 {
kount =kount+1
printf " %3f %20s %-12s %d\n", kount,$2,$3,$6 }‟ empn.lst
```

THE –f OPTION: STORING awk PROGRAMS IN A FILE

You should holds large awk programs in separate file and provide them with the awk extension for easier identification. Let's first store the previous program in the file empawk.awk:

$ cat empawk.awk

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

```
Awk –F"|" –f empawk.awk empn.lst
```

THE BEGIN AND END SECTIONS

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

The BEGIN and END sections are optional and take the form

BEGIN {action}

END {action}

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

BUILT-IN VARIABLES

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

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

BEGIN {FS="|"}

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

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

BEGIN { OFS="~" }

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

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

$awk „BEGIN {FS = "|"}

NF! =6 {

Print "Record No ", NR, "has", "fields"}" empx.lst

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-allinone2.34.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"/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 "unsetUSERNAME"

➢ 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

Program-1

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

Step1: Open text editor, type the below program and save with extention .tcl (**prog1.tcl**)

```
# Create Simulator
set ns [new Simulator]

# Open Trace file and NAM file
set ntrace [open prog1.tr w]
$ns trace-all $ntrace
set namfile [open prog1.nam w]
$ns namtrace-all $namfile

# Finish Procedure
proc Finish {} {
   global ns ntrace namfile

   # Dump all the trace data and close the files
   $ns flush-trace
   close $ntrace
   close $namfile

   # Execute the NAM animation file
   exec nam prog1.nam &

   # Show the number of packets dropped
   if {[catch {exec grep -c "^d" prog1.tr} result]} {
      puts "Error running grep: $result"
   } else {
      puts "The number of packet drops is: $result"
   }

   exit 0
}

# Create 3 nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
# Label the nodes
$n0 label "TCP Source"
$n2 label "Sink"
```

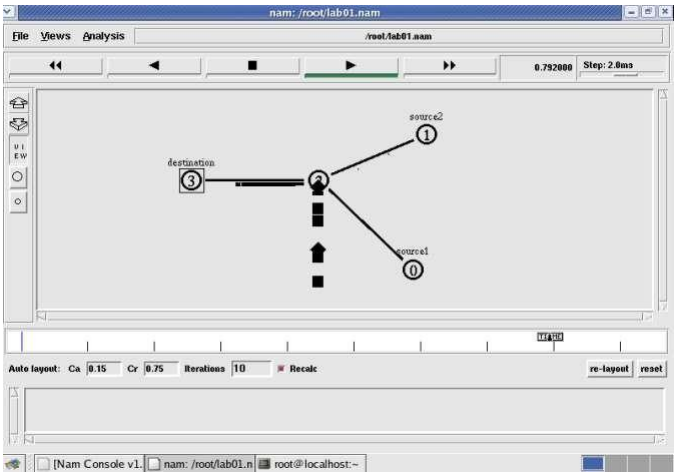
```
# Set the color
$ns color 1 blue

# Create Links between nodes
$ns duplex-link $n0 $n1 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail

# Make the Link Orientation
$ns duplex-link-op $n0 $n1 orient right
$ns duplex-link-op $n1 $n2 orient right

# Set Queue Size
$ns queue-limit $n0 $n1 10
$ns queue-limit $n1 $n2 10

# Set up a Transport layer connection
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set sink0 [new Agent/TCPSink]
$ns attach-agent $n2 $sink0
$ns connect $tcp0 $sink0

# Set up an Application layer Traffic
set cbr0 [new Application/Traffic/CBR]
$cbr0 set type_ CBR
$cbr0 set packetSize_ 100
$cbr0 set rate_ 1Mb
$cbr0 set random_ false
$cbr0 attach-agent $tcp0

$tcp0 set class_ 1

# Schedule Events
$ns at 0.0 "$cbr0 start"
$ns at 5.0 "Finish"

# Run the Simulation
$ns run
```

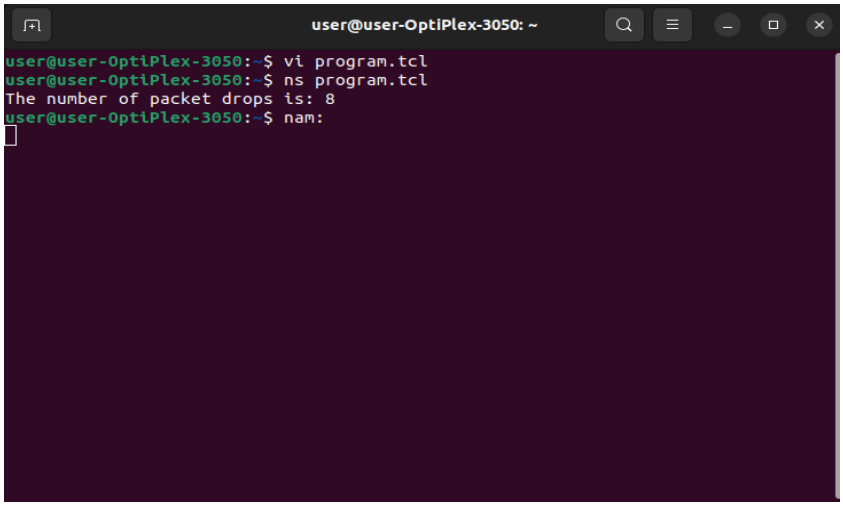
Step2: Run the simulation program

```
[root@localhost~]# ns prog1.tcl
```

(Here **"ns"** indicates network simulator. We get the topology shown in the snapshot.)

OUTPUT:






```
n -t 0 -s 0 -S DLABEL -l "TCP Source" -L ""
n -t 0 -s 2 -S DLABEL -l "Sink" -L ""
V -t * -v 1.0a5 -a 0
A -t * -n 1 -p 0 -o 0x7fffffff -c 30 -a 1
A -t * -h 1 -m 1073741823 -s 0
c -t * -i 1 -n blue
n -t * -a 0 -s 0 -S UP -v circle -c black -i black
n -t * -a 1 -s 1 -S UP -v circle -c black -i black
n -t * -a 2 -s 2 -S UP -v circle -c black -i black
l -t * -s 0 -d 1 -S UP -r 1000000 -D 0.01 -c black -o right
l -t * -s 1 -d 2 -S UP -r 1000000 -D 0.01 -c black -o right
+ -t 0 -s 0 -d 1 -p tcp -e 40 -c 1 -i 0 -a 1 -x {0.0 2.0 0 ------- null}
- -t 0 -s 0 -d 1 -p tcp -e 40 -c 1 -i 0 -a 1 -x {0.0 2.0 0 ------- null}
h -t 0 -s 0 -d 1 -p tcp -e 40 -c 1 -i 0 -a 1 -x {0.0 2.0 -1 ------- null}
r -t 0.01032 -s 0 -d 1 -p tcp -e 40 -c 1 -i 0 -a 1 -x {0.0 2.0 0 ------- null}
+ -t 0.01032 -s 1 -d 2 -p tcp -e 40 -c 1 -i 0 -a 1 -x {0.0 2.0 0 ------- null}
- -t 0.01032 -s 1 -d 2 -p tcp -e 40 -c 1 -i 0 -a 1 -x {0.0 2.0 0 ------- null}
h -t 0.01032 -s 1 -d 2 -p tcp -e 40 -c 1 -i 0 -a 1 -x {0.0 2.0 -1 ------- null}
r -t 0.02064 -s 1 -d 2 -p tcp -e 40 -c 1 -i 0 -a 1 -x {0.0 2.0 0 ------- null}
+ -t 0.02064 -s 2 -d 1 -p ack -e 40 -c 1 -i 1 -a 1 -x {2.0 0.0 0 ------- null}
- -t 0.02064 -s 2 -d 1 -p ack -e 40 -c 1 -i 1 -a 1 -x {2.0 0.0 0 ------- null}
h -t 0.02064 -s 2 -d 1 -p ack -e 40 -c 1 -i 1 -a 1 -x {2.0 0.0 -1 ------- null}
r -t 0.03096 -s 2 -d 1 -p ack -e 40 -c 1 -i 1 -a 1 -x {2.0 0.0 0 ------- null}
+ -t 0.03096 -s 1 -d 0 -p ack -e 40 -c 1 -i 1 -a 1 -x {2.0 0.0 0 ------- null}
- -t 0.03096 -s 1 -d 0 -p ack -e 40 -c 1 -i 1 -a 1 -x {2.0 0.0 0 ------- null}
h -t 0.03096 -s 1 -d 0 -p ack -e 40 -c 1 -i 1 -a 1 -x {2.0 0.0 -1 ------- null}
r -t 0.04128 -s 1 -d 0 -p ack -e 40 -c 1 -i 1 -a 1 -x {2.0 0.0 0 ------- null}
+ -t 0.04128 -s 0 -d 1 -p tcp -e 1040 -c 1 -i 2 -a 1 -x {0.0 2.0 1 ------- null}
- -t 0.04128 -s 0 -d 1 -p tcp -e 1040 -c 1 -i 2 -a 1 -x {0.0 2.0 1 ------- null}
h -t 0.04128 -s 0 -d 1 -p tcp -e 1040 -c 1 -i 2 -a 1 -x {0.0 2.0 -1 ------- null}
```

Program- 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.

Step1: Open text editor, type the below program and save with extention .tcl (**prog2.tcl**)

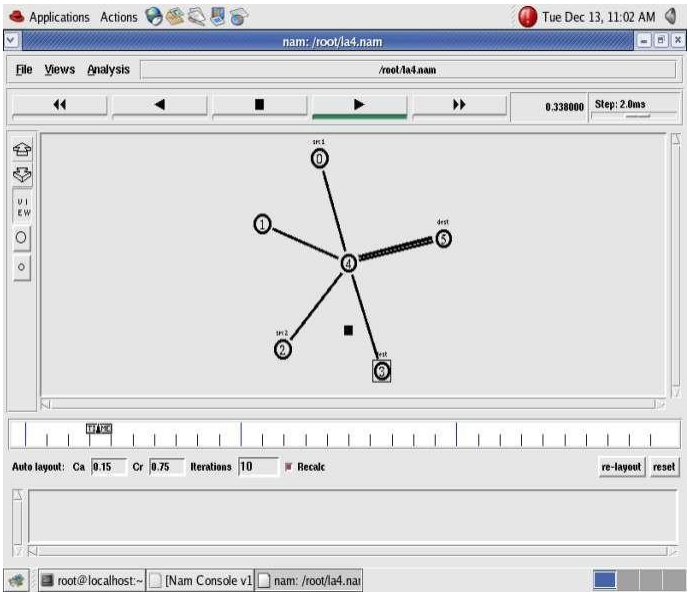
```
# Initialize the NS-2 simulator
set ns [new Simulator]

# Use colors to differentiate the traffic
$ns color 1 Blue
$ns color 2 Red

# Open trace and NAM trace files
set ntrace [open prog2.tr w]
$ns trace-all $ntrace

set namfile [open prog2.nam w]
$ns namtrace-all $namfile

# Create a global variable to count dropped packets
set dropped_packets 0

# Define the recv function for the class 'Agent/Ping'
Agent/Ping instproc recv {from rtt} {
   $self instvar node_
   puts "node [$node_ id] received ping answer from $from with round trip time $rtt ms"
}

# Finish Procedure
proc Finish {} {
   global ns ntrace namfile dropped_packets

   # Dump all trace data and close the files
   $ns flush-trace
   close $ntrace
   close $namfile

   # Execute the NAM animation file
   exec nam prog2.nam &

   # Analyze the trace file to count dropped packets
   set drop_count [exec grep "^d" prog2.tr | grep "ping" | wc -l]
   puts "The number of packets dropped: $drop_count"

   exit 0
}

# Create six nodes
for {set i 0} {$i < 6} {incr i} {
```

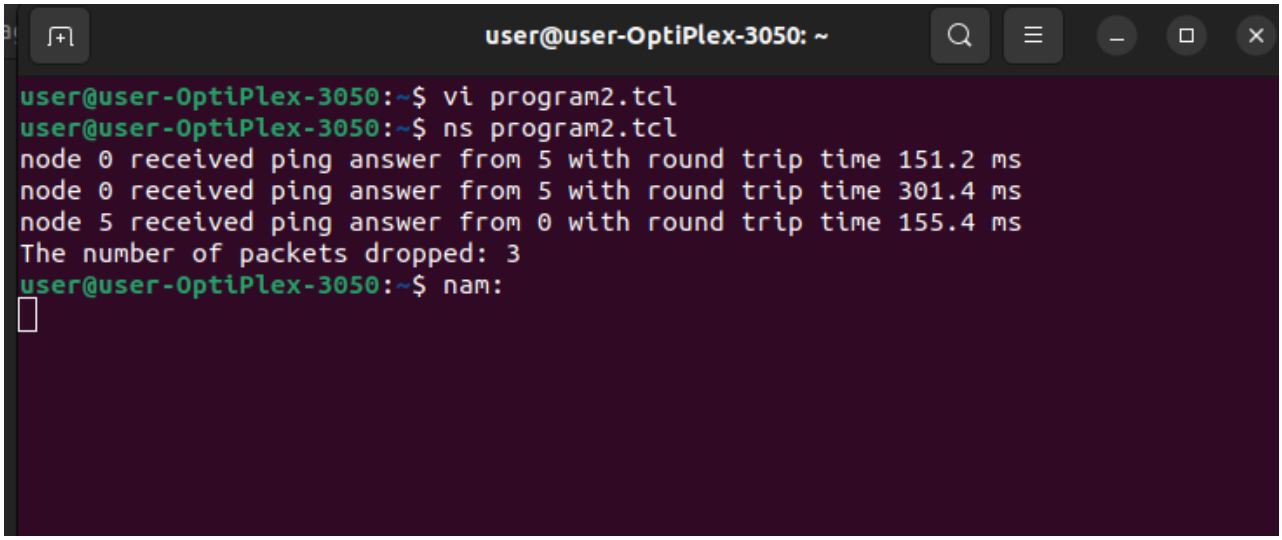
```
set n($i) [$ns node]
}

# Connect the nodes with duplex links
for {set j 0} {$j < 5} {incr j} {
   $ns duplex-link $n($j) $n([expr $j + 1]) 0.1Mb 10ms DropTail
}

# Create two ping agents and attach them to n(0) and n(5)
set p0 [new Agent/Ping]
$p0 set class_ 1
$ns attach-agent $n(0) $p0

set p1 [new Agent/Ping]
$p1 set class_ 1
$ns attach-agent $n(5) $p1
$ns connect $p0 $p1

# Set queue size and monitor the queue
$ns queue-limit $n(2) $n(3) 2
$ns duplex-link-op $n(2) $n(3) queuePos 0.5

# Create congestion with TCP and CBR traffic
set tcp0 [new Agent/TCP]
$tcp0 set class_ 2
$ns attach-agent $n(2) $tcp0

set sink0 [new Agent/TCPSink]
$ns attach-agent $n(4) $sink0
$ns connect $tcp0 $sink0

# Apply CBR traffic over TCP
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set rate_ 1Mb
$cbr0 attach-agent $tcp0

# Schedule events
$ns at 0.2 "$p0 send"
$ns at 0.4 "$p1 send"
$ns at 0.4 "$cbr0 start"
$ns at 0.8 "$p0 send"
$ns at 1.0 "$p1 send"
$ns at 1.2 "$cbr0 stop"
$ns at 1.4 "$p0 send"
$ns at 1.6 "$p1 send"
$ns at 1.8 "Finish"

# Run the simulation
$ns run
```

Step: Run the simulation program

[root@localhost~]# ns prog3.tcl

(Here **"ns"** indicates network simulator. We get the topology shown in the snapshot.)





```
V -t * -v 1.0a5 -a 0
A -t * -n 1 -p 0 -o 0x7fffffff -c 30 -a 1
A -t * -h 1 -m 1073741823 -s 0
c -t * -i 1 -n Blue
c -t * -i 2 -n Red
n -t * -a 4 -s 4 -S UP -v circle -c black -i black
n -t * -a 0 -s 0 -S UP -v circle -c black -i black
n -t * -a 5 -s 5 -S UP -v circle -c black -i black
n -t * -a 1 -s 1 -S UP -v circle -c black -i black
n -t * -a 2 -s 2 -S UP -v circle -c black -i black
n -t * -a 3 -s 3 -S UP -v circle -c black -i black
l -t * -s 0 -d 1 -S UP -r 100000 -D 0.01 -c black
l -t * -s 1 -d 2 -S UP -r 100000 -D 0.01 -c black
l -t * -s 2 -d 3 -S UP -r 100000 -D 0.01 -c black
l -t * -s 3 -d 4 -S UP -r 100000 -D 0.01 -c black
l -t * -s 4 -d 5 -S UP -r 100000 -D 0.01 -c black
q -t * -s 3 -d 2 -a 0.5
q -t * -s 2 -d 3 -a 0.5
+ -t 0.2 -s 0 -d 1 -p ping -e 64 -c 1 -i 0 -a 1 -x {0.0 5.0 -1 ------- null}
- -t 0.2 -s 0 -d 1 -p ping -e 64 -c 1 -i 0 -a 1 -x {0.0 5.0 -1 ------- null}
h -t 0.2 -s 0 -d 1 -p ping -e 64 -c 1 -i 0 -a 1 -x {0.0 5.0 -1 ------- null}
r -t 0.21512 -s 0 -d 1 -p ping -e 64 -c 1 -i 0 -a 1 -x {0.0 5.0 -1 ------- null}
+ -t 0.21512 -s 1 -d 2 -p ping -e 64 -c 1 -i 0 -a 1 -x {0.0 5.0 -1 ------- null}
- -t 0.21512 -s 1 -d 2 -p ping -e 64 -c 1 -i 0 -a 1 -x {0.0 5.0 -1 ------- null}
h -t 0.21512 -s 1 -d 2 -p ping -e 64 -c 1 -i 0 -a 1 -x {0.0 5.0 -1 ------- null}
r -t 0.23024 -s 1 -d 2 -p ping -e 64 -c 1 -i 0 -a 1 -x {0.0 5.0 -1 ------- null}
+ -t 0.23024 -s 2 -d 3 -p ping -e 64 -c 1 -i 0 -a 1 -x {0.0 5.0 -1 ------- null}
- -t 0.23024 -s 2 -d 3 -p ping -e 64 -c 1 -i 0 -a 1 -x {0.0 5.0 -1 ------- null}
h -t 0.23024 -s 2 -d 3 -p ping -e 64 -c 1 -i 0 -a 1 -x {0.0 5.0 -1 ------- null}
r -t 0.24536 -s 2 -d 3 -p ping -e 64 -c 1 -i 0 -a 1 -x {0.0 5.0 -1 ------- null}
+ -t 0.24536 -s 3 -d 4 -p ping -e 64 -c 1 -i 0 -a 1 -x {0.0 5.0 -1 ------- null}
- -t 0.24536 -s 3 -d 4 -p ping -e 64 -c 1 -i 0 -a 1 -x {0.0 5.0 -1 ------- null}
```

Program- 3

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

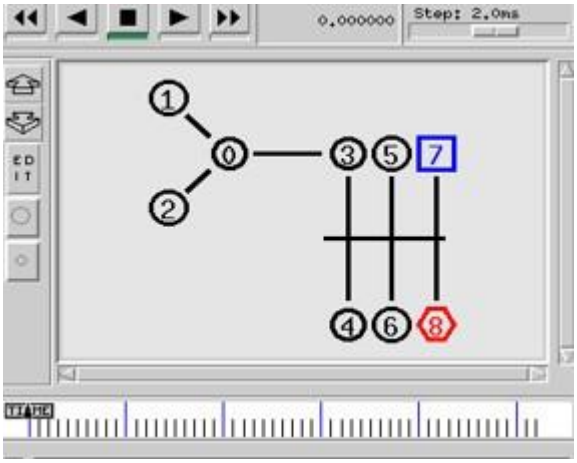
Step1: Open text editor, type the below program and save with extention .tcl (**prog5.tcl**)

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

#Open the NS trace file
set tracefile [open 5.tr w]
$ns trace-all $tracefile

#Open the NAM trace file
set namfile [open 5.nam w]
$ns namtrace-all $namfile
$ns color 1 Blue
$ns color 2 Red

# Nodes Definition
#Create 9 nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
set n7 [$ns node]
set n8 [$ns node]

# Create LAN
$ns make-lan "$n3 $n4 $n5 $n6 $n7 $n8" 512Kb 50ms LL Queue/DropTail

# Links Definition
#Createlinks between nodes
$ns duplex-link $n1 $n0 2.0Mb 50ms DropTail
$ns queue-limit $n1 $n0 50
$ns duplex-link $n2 $n0 2.0Mb 50ms DropTail
$ns queue-limit $n2 $n0 50
$ns duplex-link $n0 $n3 1.0Mb 50ms DropTail
$ns queue-limit $n0 $n3 7
#Give node position (for NAM)
$ns duplex-link-op $n1 $n0 orient right-down
$ns duplex-link-op $n2 $n0 orient right-up
$ns duplex-link-op $n0 $n3 orient right
# Agents Definition
#Setup a TCP/Reno connection
```

```
set tcp0 [new Agent/TCP/Reno]
$ns attach-agent $n1 $tcp0
set sink1 [new Agent/TCPSink]
$ns attach-agent $n7 $sink1
$ns connect $tcp0 $sink1
$tcp0 set packetSize_ 1500
$tcp0 set class_ 1
set tfile1 [open cwnd1.tr w]
$tcp0 attach $tfile1
$tcp0 trace cwnd_

#Setup a TCP/Vegas connection
set tcp5 [new Agent/TCP/Vegas]
$ns attach-agent $n2 $tcp5
set sink6 [new Agent/TCPSink]
$ns attach-agent $n8 $sink6
$ns connect $tcp5 $sink6
$tcp5 set packetSize_ 1500
$tcp5 set class_ 2
set tfile2 [open cwnd2.tr w]
$tcp5 attach $tfile2
$tcp5 trace cwnd_

# Applications Definition
#Setup a FTP Application over TCP/Reno connection
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
$ns at 0.3 "$ftp0 start"
$ns at 8.0 "$ftp0 stop"

#Setup a FTP Application over TCP/Vegas connection
set ftp4 [new Application/FTP]
$ftp4 attach-agent $tcp5
$ns at 0.3 "$ftp4 start"
$ns at 8.0 "$ftp4 stop"

# Termination
#Define a 'finish' procedure
proc finish {} {
global ns tracefile namfile
$ns flush-trace
close $tracefile
close $namfile
exec nam 5.nam &
exit 0
}

$ns at $val(stop) "$ns nam-end-wireless $val(stop)"
$ns at $val(stop) "finish"
$ns at $val(stop) "puts \"done\" ; $ns halt"
$ns run
```

Step: Run the simulation program

[root@localhost~]# ns prog5.tcl



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:

1. 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.

2. Run Java program from Command Prompt

   [root@host ~]# java Filename

   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).

Program- 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.*;
class crc
{
void div(int a[],int k)
{
int gp[]={1,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,1};
int count=0;
for(int i=0;i<k;i++)
{
if(a[i]==gp[0])
if(a[i]==gp[0])
{
for(int j=i;j<17+i;j++)
{
a[j]=a[j]^gp[count++];
}
count=0;
}
}
}
public static void main(String args[])
{
int a[]=new int[100];
int b[]=new int[100];
int len,k;
crc ob=new crc();
System.out.println("Enter the length of Data Frame:");
Scanner sc=new Scanner(System.in);
len=sc.nextInt();
int flag=0;
System.out.println("Enter the Message:");
for(int i=0;i<len;i++)
{ a[i]=sc.nextInt();
}
for(int i=0;i<16;i++)
{ a[len++]=0;
}
k=len-16;
for(int i=0;i<len;i++)
{ b[i]=a[i];
}
ob.div(a,k);
for(int i=0;i<len;i++)
a[i]=a[i]^b[i];
System.out.println("Data to be transmitted: ");
for(int i=0;i<len;i++)
{
```

```
System.out.print(a[i]+" ");
}
System.out.println();
System.out.println("Enter the Received Data: ");
for(int i=0;i<len;i++)
{
a[i]=sc.nextInt();
}
ob.div(a, k);
for(int i=0;i<len;i++)
{
if(a[i]!=0)
{
flag=1;
break;
}
}
if(flag==1)
System.out.println("ERROR in data");
else
System.out.println("NO ERROR");
}
}
```

OUTPUT-1

```
Enter the length of Data Frame: 4
Enter the Message: 1 0 1 1
Data to be transmitted: 1 0 1 1 1 0 1 1 0 0 0 1 0 1 1 0 1 0 1 1
Enter the Received Data: 1 0 1 1 1 0 1 1 0 0 0 0 0 1 1 0 1 0 1 1
ERROR in Data
```

OUTPUT-2

```
Enter the length of Data Frame: 4
Enter the Message: 1 0 1 1
Data to be transmitted: 1 0 1 1 1 0 1 1 0 0 0 1 0 1 1 0 1 0 1 1
Enter the Received Data: 1 0 1 1 1 0 1 1 0 0 0 1 0 1 1 0 1 0 1 1
NO ERROR
```

Program- 5

Write a program to implement sliding window protocol in datalink layer.

The sliding window protocol is a method used in network communications to manage the flow of data between two devices and ensure that data is transmitted efficiently and accurately. It's particularly useful in scenarios where data is sent over a network with potential delays or errors. Here's a breakdown of how it works:

1. **Window Size**: Both sender and receiver maintain a "window" of frames or packets that can be sent or received. This window size determines the maximum number of frames that can be sent without receiving an acknowledgment.
2. **Sender's Window**: The sender can transmit a number of frames up to the window size before needing to wait for an acknowledgment from the receiver. Once the sender receives an acknowledgment for some of the frames, it can slide the window forward and send new frames.
3. **Receiver's Window**: The receiver also has a window size that specifies the number of frames it can buffer. It keeps track of which frames it has received correctly and sends acknowledgments back to the sender. If the receiver's buffer is full, it will need to wait before it can accept more frames.
4. **Sliding the Window**: As acknowledgments are received, both the sender and receiver "slide" their windows forward. This sliding process means that as old frames are acknowledged and removed from the window, new frames can be sent or received.
5. **Error Handling**: The protocol includes mechanisms for handling errors. If a frame is lost or corrupted, the receiver may request retransmission of that frame, and the sender will retransmit it, usually sliding the window back to the point where the error occurred.
6. **Flow Control**: The sliding window protocol helps with flow control by preventing the sender from overwhelming the receiver with too many frames at once. The receiver's window size ensures that it only processes a manageable number of frames at any time.

This protocol is commonly used in various data link layer and transport layer protocols, such as the Transmission Control Protocol (TCP) in the Internet Protocol Suite.

Source code:

```
import java.util.LinkedList;
import java.util.Queue;
public class SlidingWindowProtocol {

    private static final int WINDOW_SIZE = 4;  // Size of the sliding window
    private static final int TOTAL_PACKETS = 10; // Total number of packets to send

    // Simulates the sender's sliding window protocol
    public static void sender() {
        Queue<Integer> window = new LinkedList<>();
        for (int i = 0; i < WINDOW_SIZE; i++) {
            if (i < TOTAL_PACKETS) {
                window.add(i);
                System.out.println("Sent packet: " + i);
            }
        }

        int nextPacketToSend = WINDOW_SIZE;

        while (nextPacketToSend < TOTAL_PACKETS || !window.isEmpty()) {
            if (!window.isEmpty()) {
                // Simulate receiving acknowledgment for the packet at the start of the window
                int ack = window.poll();
                System.out.println("Acknowledgment received for packet: " + ack);
            }

            // Slide the window
            if (nextPacketToSend < TOTAL_PACKETS) {
                window.add(nextPacketToSend);
System.out.println("Sent packet: " + nextPacketToSend);
                nextPacketToSend++;
            }

            // Simulate a delay for demonstration purposes
            try {
                Thread.sleep(1000); // 1 second delay
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        System.out.println("All packets sent and acknowledged.");
    }

    public static void main(String[] args) {
        sender();
    }

}
```

Explanation

1. **Initialization**:
    - The window size (WINDOW_SIZE) determines how many packets can be sent before receiving acknowledgments.
    - TOTAL_PACKETS represents the total number of packets to send.
2. **Sender Function**:
    - Initializes the window with the first set of packets.
    - Simulates sending packets and receiving acknowledgments.
    - Slides the window by removing the acknowledged packet and adding a new packet.
3. **Sliding the Window**:
    - After sending packets, the window slides by removing the packet at the start of the window (simulating acknowledgment) and adding new packets as they are sent.
4. **Simulation**:
    - The Thread.sleep(1000) call simulates a delay, representing the time between sending packets and receiving acknowledgments.

OUTPUT

Sent packet: 0
Sent packet: 1
Sent packet: 2
Sent packet: 3
Acknowledgment received for packet: 0
Sent packet: 4
Acknowledgment received for packet: 1
Sent packet: 5
Acknowledgment received for packet: 2
Sent packet: 6
Acknowledgment received for packet: 3
Sent packet: 7
Acknowledgment received for packet: 4
Sent packet: 8
Acknowledgment received for packet: 5
Sent packet: 9
Acknowledgment received for packet: 6
Acknowledgment received for packet: 7
Acknowledgment received for packet: 8
Acknowledgment received for packet: 9
All packets sent and acknowledged.

Program- 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 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,n;

public static final int MAX_VALUE = 999;

public BellmanFord(int n)

{

this.n=n;

D = new int[n+1];

}
```

```
public void shortest(int s,int A[][])
{
for (int i=1;i<=n;i++)
{
D[i]=MAX_VALUE;
}
D[s] = 0;
for(int k=1;k<=n-1;k++)
{
for(int i=1;i<=n;i++)
{
for(int j=1;j<=n;j++)
{
if(A[i][j]!=MAX_VALUE)
{
if(D[j]>D[i]+A[i][j])
D[j]=D[i]+A[i][j];
}
}
}
}
for(int i=1;i<=n;i++)
{
for(int j=1;j<=n;j++)
{
if(A[i][j]!=MAX_VALUE)
{
if(D[j]>D[i]+A[i][j])
{
System.out.println("The Graph contains negative egde cycle");
```

```
return;
}
}
}
}
for(int i=1;i<=n;i++)
{
System.out.println("Distance of source " + s + " to "+ i + " is " + D[i]);
}
}
public static void main(String[ ] args)
{
int n=0,s;
Scanner sc = new Scanner(System.in);
System.out.println("Enter the number of vertices");
n = sc.nextInt();
int A[][] = new int[n+1][n+1];
System.out.println("Enter the Weighted matrix");
for(int i=1;i<=n;i++)
{
for(int j=1;j<=n;j++)
{
A[i][j]=sc.nextInt();
if(i==j)
{
A[i][j]=0;
continue;
}
if(A[i][j]==0)
{
```

```
A[i][j]=MAX_VALUE;
}
}}
System.out.println("Enter the source vertex");
s=sc.nextInt();
BellmanFord b = new BellmanFord(n);
b.shortest(s,A);
sc.close();
}
}
```

Output:

```
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 1 is 5

Distance of source 2 to 2 is 0

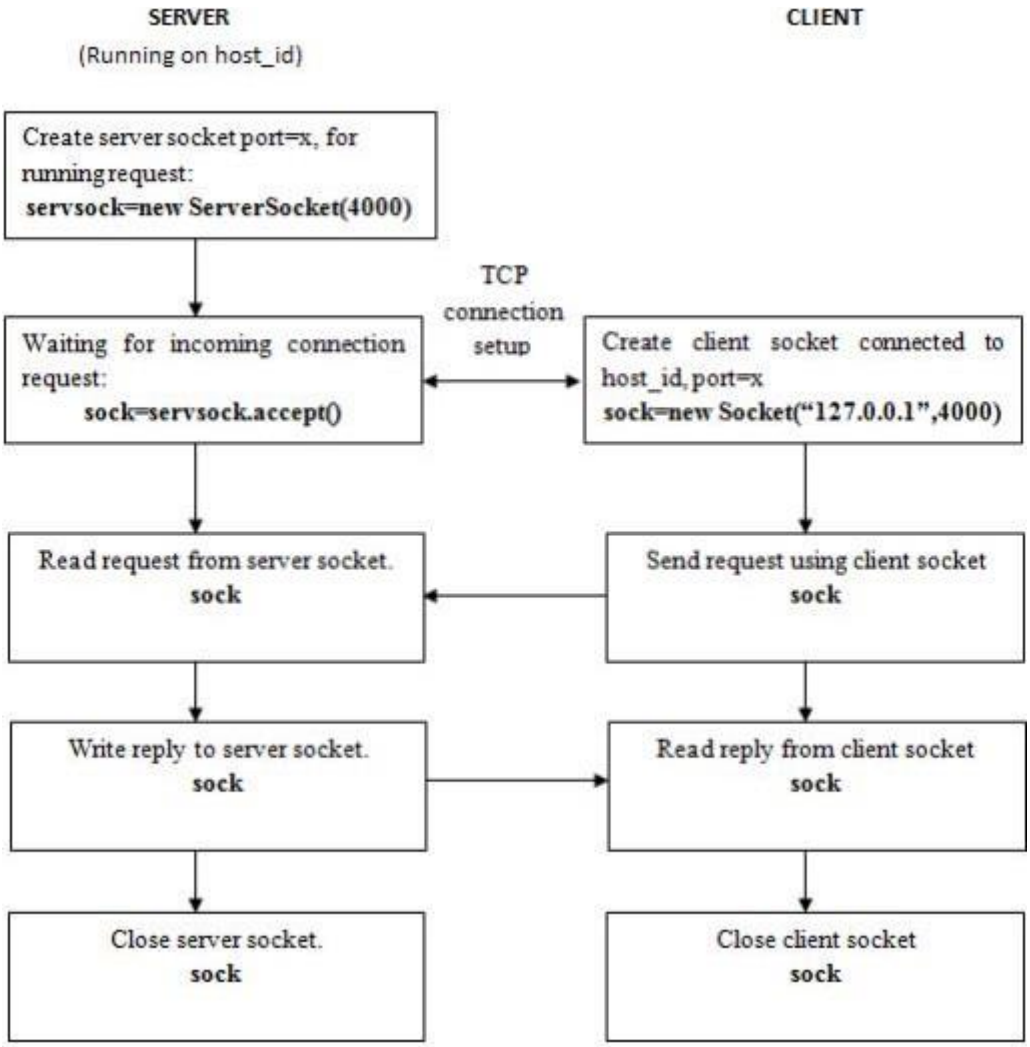
Distance of source 2 to 3 is 3

Distance of source 2 to 4 is 4

Program- 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.



Client program

```
import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.EOFException;
import java.io.File;
import java.io.FileOutputStream;
import java.io.InputStreamReader;
import java.net.Socket;
import java.util.Scanner;
class Client
{
public static void main(String args[])throws Exception
{
String address = "";
Scanner sc=new Scanner(System.in);
System.out.println("Enter Server Address: ");
address=sc.nextLine();
//create the socket on port 5000
Socket s=new Socket(address,5000);
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
System.out.println("Send Get to start...");
String str="",filename="";
try
{
while(!str.equals("start"))
str=br.readLine();
```

```
dout.writeUTF(str);
dout.flush();
filename=din.readUTF();
System.out.println("Receving file: "+filename);
filename="client"+filename;
System.out.println("Saving as file: "+filename);
long sz=Long.parseLong(din.readUTF());
System.out.println ("File Size: "+(sz/(1024*1024))+" MB");
byte b[]=new byte [1024];
System.out.println("Receving file..");
FileOutputStream fos=new FileOutputStream(new File(filename),true);
long bytesRead;
do
{
bytesRead = din.read(b, 0, b.length);
fos.write(b,0,b.length);
}while(!(bytesRead<1024));
System.out.println("Completed");
fos.close();
dout.close();
s.close();
}
catch(EOFException e)
{
}
}
}
```

SERVER PROGRAM

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
```

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Scanner;
public class Server
   public static void main(String[] args) {
      String filename;
      Scanner sc = new Scanner(System.in);
      System.out.println("Enter File Name: ");
      filename = sc.nextLine();
      sc.close();
      try (ServerSocket serverSocket = new ServerSocket(5000)) {
         System.out.println("Server started and waiting for a request...");
         while (true) {
            try (Socket socket = serverSocket.accept();
               DataInputStream din = new DataInputStream(socket.getInputStream());
               DataOutputStream dout = new DataOutputStream(socket.getOutputStream()))
{
             System.out.println("Connected with " + socket.getInetAddress().toString())
              String request = din.readUTF();
              if (request.equals("start")) {
                 System.out.println("Received 'start' request.");
                 File file = new File(filename);
                 if (!file.exists()) {
                    System.out.println("File not found: " + filename);
                    dout.writeUTF("File not found");
                    dout.flush();
                    continue;  }
```

```
    // Send the filename and file size
            dout.writeUTF(filename);
            dout.flush();
            long fileSize = file.length();
            dout.writeUTF(Long.toString(fileSize));
            dout.flush();
            System.out.println("Sending file: " + filename);
            System.out.println("File size: " + (fileSize / (1024 * 1024)) + " MB");
            try (FileInputStream fin = new FileInputStream(file)) {
               byte[] buffer = new byte[1024];
               int bytesRead;
               while ((bytesRead = fin.read(buffer)) != -1) {
                  dout.write(buffer, 0, bytesRead);
                  dout.flush();
               }
            }
            System.out.println("File sent successfully.");
         } else {
            dout.writeUTF("Invalid request");
            dout.flush();
         }
      } catch (IOException e) {
         e.printStackTrace();
         System.out.println("An error occurred while handling the client request.");
      }
   }
     } catch (IOException e) {
   e.printStackTrace();
   System.out.println("Failed to start the server.");
} }}
```

Note: Create two different files Client.java and Server.java. Follow the steps given:

1. Open a terminal run the server program and provide the filename to send

2. Open one more terminal run the client program and provide the IP address of the server. We can give localhost address "127.0.0.1" as it is running on same machine or give the IP address of the machine.

3. Send any start bit to start sending file.

4. Refer https://www.tutorialspoint.com/java/java_networking.htm for all the parameters, methods description in socket communication.

OUTPUT:

AT SERVER SIDE

```
user@user-OptiPlex-3050:~/Desktop$ javac Server.java
user@user-OptiPlex-3050:~/Desktop$ java Server
Enter File Name:
flower.jpg
Server started and waiting for a request...
Connected with /127.0.0.1
Received &apos;start&apos; request.
Sending file: flower.jpg
File size: 0 MB
File sent successfully
```

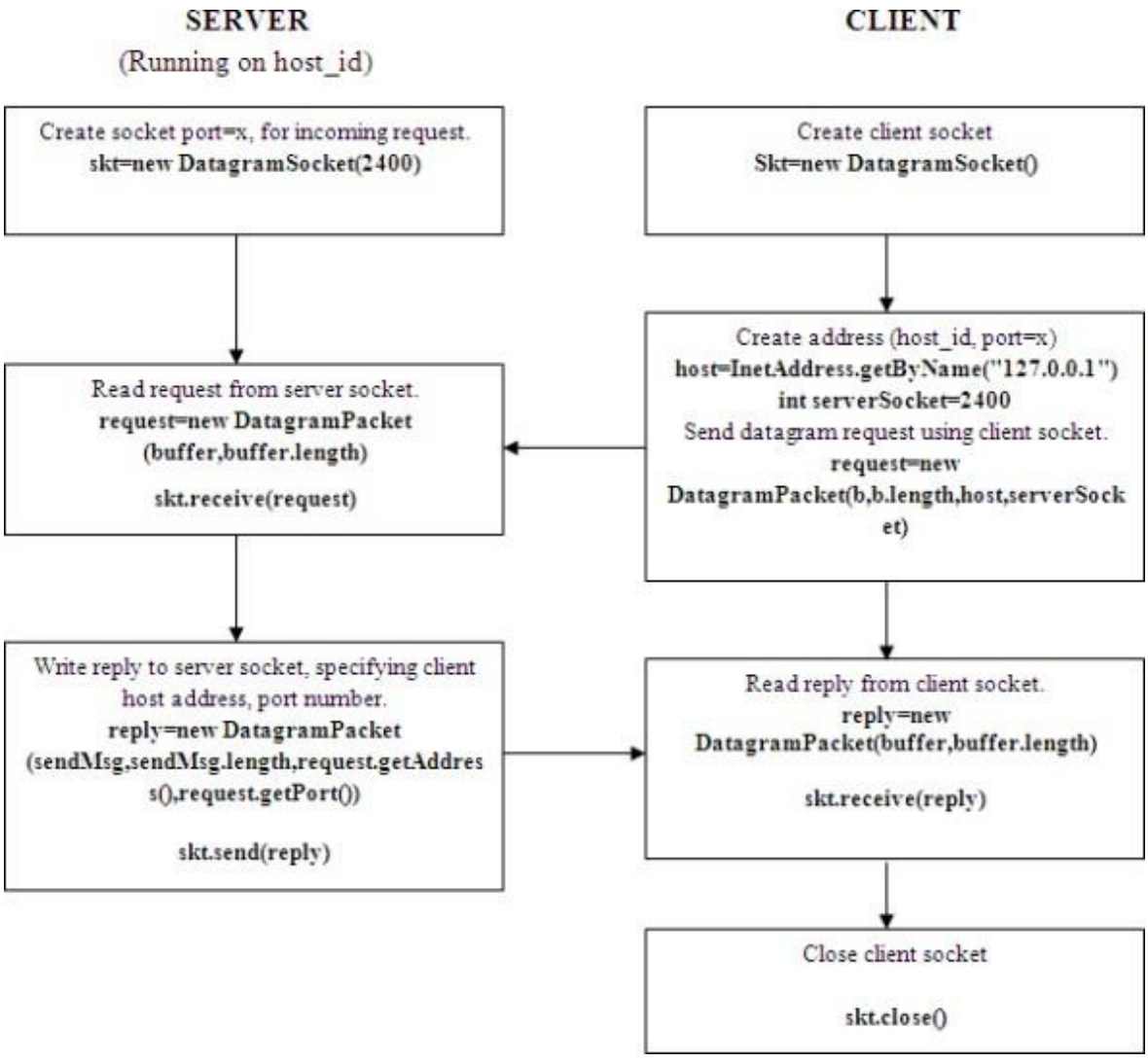
AT CLIENT SIDE

```
user@user-OptiPlex-3050:~/Desktop$ javac Client.java
user@user-OptiPlex-3050:~/Desktop$ java Client
Enter Server Address:
127.0.0.1
Send Get to start...
start
Receving file: flower.jpg
Saving as file: clientflower.jpg
File Size: 0 MB
Receving file..
Completed
```

Program- 8

Write a program on 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.



SourceCode:

UDP CLIENT

```
import java.io.*;
import java.net.*;
public class UDPC
{
public static void main(String[] args)
{
DatagramSocket skt;
try
{
skt=new DatagramSocket();
String msg= "text message ";
byte[] b = msg.getBytes();
InetAddress host=InetAddress.getByName("127.0.0.1");
int serverSocket=6788;
DatagramPacket request =new DatagramPacket (b,b.length,host,serverSocket);
skt.send(request);
byte[] buffer =new byte[1000];
DatagramPacket reply= new DatagramPacket(buffer,buffer.length);
skt.receive(reply);
System.out.println("client received:" +new String(reply.getData()));
skt.close();
}
catch(Exception ex)
{
}
}
}
```

UDP SERVER

```
import java.io.*;
import java.net.*;
public class UDPS
{
public static void main(String[] args)
{
DatagramSocket skt=null;
try
{
skt=new DatagramSocket(6788);
byte[] buffer = new byte[1000];
while(true)
{
DatagramPacket request = new DatagramPacket(buffer,buffer.length);
skt.receive(request);
String[] message = (new String(request.getData())).split(" ");
byte[] sendMsg= (message[1]+ " server processed").getBytes();
DatagramPacket reply = new
DatagramPacket(sendMsg,sendMsg.length,request.getAddress(),request.getPort());
skt.send(reply);
}
}
catch(Exception ex)
{
}
}
}
```

OUTPUT

SERVER SIDE

```
ser@user-OptiPlex-3050:~$ cd Desktop

user@user-OptiPlex-3050:~/Desktop$ javac UDPS.java
user@user-OptiPlex-3050:~/Desktop$ java UDPS
```

CLIENT SIDE

```
user@user-OptiPlex-3050:~/Desktop$ javac UDPC.java

user@user-OptiPlex-3050:~/Desktop$ java UDPC
client received:message server processed
```

Program- 9

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 anumber (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.io.DataInputStream;
import java.io.IOException;
import java.math.BigInteger;
import java.util.Random;
public class RSA
{
private BigInteger p,q,N,phi,e,d;
private int bitlength=1024;
private Random r;
public RSA()
{
r=new Random();
p=BigInteger.probablePrime(bitlength,r);
q=BigInteger.probablePrime(bitlength,r);
System.out.println("Prime number p is"+p);
System.out.println("prime number q is"+q);
N=p.multiply(q);
phi=p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));
e=BigInteger.probablePrime(bitlength/2,r);
while(phi.gcd(e).compareTo(BigInteger.ONE)>0&&e.compareTo(phi)<0)
{
e.add(BigInteger.ONE);
}
System.out.println("Public key is"+e);
d=e.modInverse(phi);
System.out.println("Private key is"+d);
}
public RSA(BigInteger e,BigInteger d,BigInteger N)
{
this.e=e;
```

```
this.d=d;
this.N=N;
}
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()));
byte[] encrypted=rsa.encrypt(testString.getBytes());
byte[] decrypted=rsa.decrypt(encrypted);
System.out.println("Dcrypting Bytes:"+bytesToString(decrypted));
System.out.println("Dcrypted string:"+new String(decrypted));
}
private static String bytesToString(byte[] encrypted)
{
String test=" ";
for(byte b:encrypted)
{
test+=Byte.toString(b);
}
return test;
}
public byte[]encrypt(byte[]message)
{
return(new BigInteger(message)).modPow(e,N).toByteArray();
}
```

```
public byte[]decrypt(byte[]message)
{
return(new BigInteger(message)).modPow(d,N).toByteArray();
}
}
```

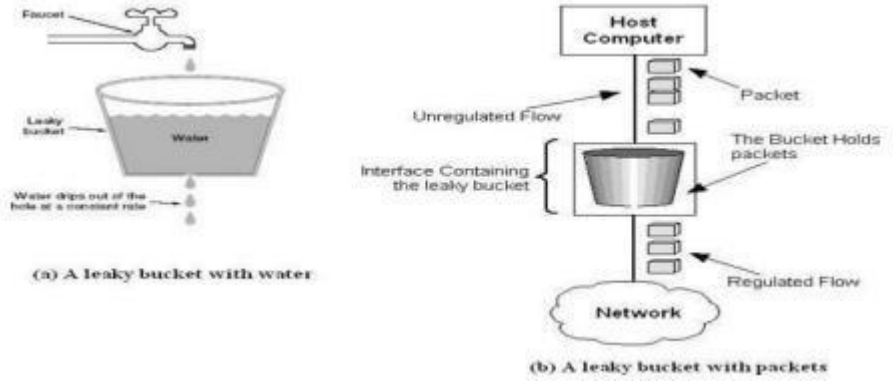
OUTPUT

```
user@user-OptiPlex-3050:~/Desktop$ javac RSA.java
Note: RSA.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
user@user-OptiPlex-3050:~/Desktop$ java RSA
Prime number p
is9434041555966760419639699719435550062061726189236379933318613058155503055902225841264891316693844774939617294283650962209553850054833765439077133114945579551474332728867326799761606710707727123427269947331625843353981838131610867725394852071706607974213087219139102687697893830159529784172945010477751574432 9
prime number q
is1629598763359750702207903579265136270865303408645117941279139515901843403968657441908444545709130901995801135668207776216711680069051286109482368572941249300498548069515141836455735744151938208749235481230328120444329312928487366156541973472639768999105667451493746494619812447605895134166390694042887709897747
Public key
is1253531723193161509753529267918552214239812435914857863149186755566054159490814224380323692838356440909122548541674948384967673130729256425466887634779011 3
Private key
is1344706880345414725001654809915499428800120500205202938088936644254244800976690248370651162937877881427899003449673843431006127738007899220403136870178721884825862219459686943334156924436387393538409510559261651100143375986973916486372833641712855589162249717394066806619491169200573841119916592869630316452377931002782919506672379883291774943110512761359917276098071775301735522806312566850606859872453358910432793165731692975877716335913366358494677352004012674686046317131736280472134341419173492749602169091579591269304457139775415476831072132075121853768901090500673503912679070097939799005120608512595174713083 3
Enter the plain text:
Hello everyone
Encrypting string:Hello everyone
string in bytes: 72101108108111321011181011141211111101 01
Dcrypting Bytes: 72101108108111321011181011141211111101 01
Dcrypted string:Hello everyone
```

Program- 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

```
import java.util.*;
public class leaky
{
static int min(int x,int y)
{
if(x<y)
return x;
else
return y;
}
public static void main(String[] args)
{ int drop=0,mini,nsec,cap,count=0,i,process;
int inp[]=new int[25];
Scanner sc=new Scanner(System.in);
System.out.println("Enter The Bucket Size\n");
cap= sc.nextInt();
System.out.println("Enter The Operation Rate\n");
process= sc.nextInt();
System.out.println("Enter The No. Of Seconds You Want To Stimulate\n");
nsec=sc.nextInt();
for(i=0;i<nsec;i++)
{
System.out.println("Enter The Size Of The Packet Entering At 1 sec:");
inp[i] = sc.nextInt();
}
System.out.println("\nSecond | Packet Recieved | Packet Sent | Packet Left |Packet Dropped|\n");
System.out.println("-----------------------------------------------\n");
for(i=0;i<nsec;i++)
{ count+=inp[i];
if(count>cap)
{ drop=count-cap;
count=cap;
}
System.out.print(i+1);
System.out.print("\t\t"+inp[i]);
mini=min(count,process);
System.out.print("\t\t"+mini);
count=count-mini;
System.out.print("\t\t"+count);
System.out.print("\t\t"+drop);
drop=0;
System.out.println();
```

```
}
for(;count!=0;i++)
{
if(count>cap)
{
drop=count-cap;
count=cap;
}
System.out.print(i+1);
System.out.print("\t\t0");
mini=min(count,process);
System.out.print("\t\t"+mini);
count=count-mini;
System.out.print("\t\t"+count);
System.out.print("\t\t"+drop);
System.out.println();
}
}
}
```

OUTPUT

```
user@user-OptiPlex-3050:~/Desktop$ javac leaky.java
user@user-OptiPlex-3050:~/Desktop$ java leaky
Enter The Bucket Size

5
Enter The Operation Rate

2
Enter The No. Of Seconds You Want To Stimulate

3
Enter The Size Of The Packet Entering At 1 sec:
2
Enter The Size Of The Packet Entering At 1 sec:
3
Enter The Size Of The Packet Entering At 1 sec:
5

Second | Packet Recieved | Packet Sent | Packet Left |Packet Dropped|

-------------------------------------------------------------------------------

1       2       2       0       0
2       3       2       1       0
3       5       2       3       1
4       0       2       1       0
5       0       1       0       0
```

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 filedescriptor?
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 7,4code?
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. Differentiate between Prim's and Kruskal'salgorithm.
44. What are Routing algorithms?
45. How do you classify routing algorithms? Give examples foreach.
46. What are drawbacks in distance vector algorithm?
47. How routers update distances to each of its neighbor?
48. How do you overcome count to infinity problem?
49. What is cryptography?
50. How do you classify cryptographic algorithms?
51. What is public key?
52. What is private key?
53. What are key, ciphertext and plain text?
54. What is simulation?
55. What are advantages of simulation?
56. Differentiate between Simulation and Emulation.
57. What is meant by router?
58. What is meant by bridge?
59. What is meant by switch?
60. What is meant by hub?
61. Differentiate between route, bridge, switch and hub.
62. What is ping and telnet?
63. What isFTP?
64. What is BER?
65. What is meant by congestion window?
66. What is BSS
67. What is incoming throughput and outgoing throughput?
68. What is collision?
69. How do you generate multiple traffics across different sender-receiver pairs?
70. How do you setup Ethernet LAN?

71. What is meant by mobile host?
72. What is meant by NCTUns?
73. What are dispatcher, coordinator and nctuns client?
74. Name few other Network simulators
75. Differentiate between logical and physical address.
76. Which address gets affected if a system moves from one place to another place?
77. What is ICMP? What are uses of ICMP? Name few.
78. Which layer implements security for data?