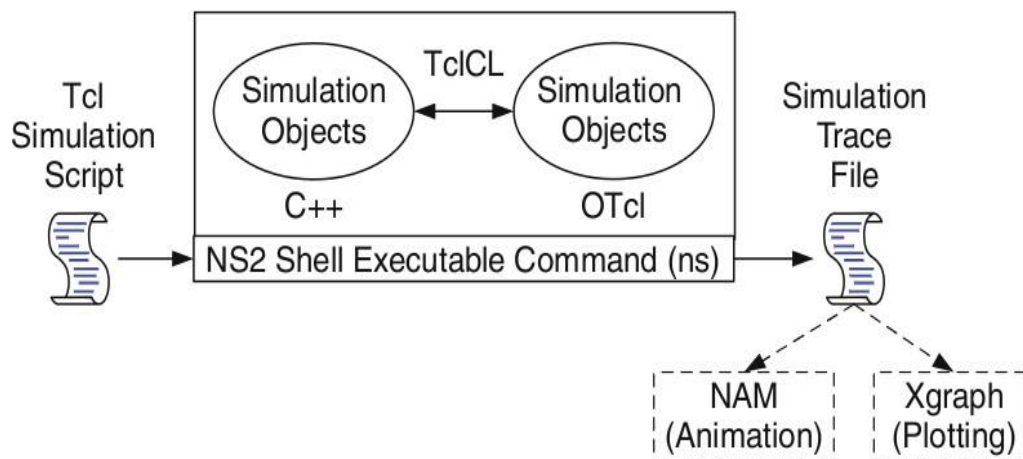


PART-A

Introduction to NS-2

- Widely known as NS2, is simply an event driven simulation tool.
- Useful in studying the dynamic nature of communication networks.
- Simulation of wired as well as wireless network functions and protocols (e.g., routing algorithms, TCP, UDP) can be done using NS2.
- In general, NS2 provides users with a way of specifying such network protocols and simulating their corresponding behaviors.

Basic Architecture of NS2



Tcl scripting

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

Basics of TCL

Syntax: command arg1 arg2 arg3

○ Hello World!

```
puts stdout{Hello, World!}
```

Hello, World!

○ Variables

Command

```
Substitution set a 5 set len [string length
```

```
foobar]
```

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

○ Simple Arithmetic

expr 7.2 / 4

○ Procedures

```
proc Diag {a b} {  
    set c [expr sqrt($a * $a + $b * $b)]  
    return $c }
```

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

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

○ Loops

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

Wired TCL Script Components

Create the event scheduler

Open new files & turn on the tracing

Create the nodes

Setup the links

Configure the traffic type (e.g., TCP, UDP, etc) Set
the time of traffic generation (e.g., CBR, FTP)

Terminate the simulation

NS Simulator Preliminaries.

1. Initialization and termination aspects of the ns simulator.
2. Definition of network nodes, links, queues and topology.
3. Definition of agents and of applications.
4. The nam visualization tool.
5. Tracing and random variables.

Initialization and Termination of TCL Script in NS-2

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 setcommand, 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 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  
  
}
```


it is an instance of the Simulator class, so an object the code[`new Simulator`] is indeed the installation of the class Simulator using the reserved word `new`.

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

#Open the Trace file

```
set tracefile1 [open out.tr w]

$ns trace-all $tracefile1
```

#Open the NAM trace file

```
set namfile [open out.nam w]

$ns namtrace-all $namfile
```

The above creates a trace file called “out.tr” and a nam visualization trace file called “out.nam”. Within the tcl script, these files are not called explicitly by their names, but instead by pointers that are declared above and called “tracefile1” and “namfile” respectively. Remark that they 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 end 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,

will be used to call “**finish**” at time 125.0. Indeed, the **at** method of the simulator allows us to schedule events explicitly.

The simulation can then begin using the command

\$ns run

Definition of a network of links and nodes

The way to define a node is

set n0 [\$ns node]

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

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

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

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

To define a directional link instead of a bi-directional one, we should replace “duplex-link” by “simplex-link”.

In NS, an output queue of a node is implemented as a part of each link whose input is that node. The definition of the link then includes the way to handle overflow at that queue. In our case, if the buffer capacity of the output queue is exceeded then the last packet to arrive is dropped. Many alternative options exist, such as the RED (Random Early 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:

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

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

connection. The command

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

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

#Setup a UDP connection

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

#setup a CBR over UDP connection

```
set cbr [new Application/Traffic/CBR]

$cbr attach-agent $udp

$cbr set packetSize_ 100

$cbr set rate_ 0.01Mb

$cbr set random_ false
```

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

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

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

When we have several flows, we may wish to distinguish them so that we can identify them with different colors in the visualization part. This is done by the command **\$tcp set fid_ 1** that assigns to the TCP connection a flow identification of “1”. We shall later give the flow identification of “2” to the UDP connection.

CBR over UDP

A UDP source and destination is defined in a similar way as in the case of TCP.

Instead of defining the rate in the command **\$cbr set rate_ 0.01Mb**, one can define the time interval between transmission of packets using the command. The packet size can be set to some value using

```
$cbr set interval_ 0.005
```

```
$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:

The scheduler is started when running ns that is through the command \$ns run. The beginning and end of the FTP and CBR application can be done through the following command

\$ns at 0.1 "\$cbr start"

\$ns at 1.0 "\$ftp start"

\$ns at 124.0 "\$ftp stop"

\$ns at 124.5 "\$cbr stop"

Structure of Trace Files

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

| 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 event occurs.
3. Gives the input node of the link at which the event occurs.
4. Gives the output node of the link at which the event occurs.
5. Gives the packet type (eg CBR or TCP)
6. Gives the packet size
7. Some flags
8. This is the flow id (fid) of IPv6 that a user can set for each flow at the input OTcl script one can further use this field for analysis purposes; it is also used when specifying stream color for the NAM display.
9. This is the source address given in the form of "node.port".
10. This is the destination address, given in the same form.
11. This is the network layer protocol's packet sequence number. Even though UDP implementations in a real network do not use sequence number, ns keeps track of UDP packet sequence number for analysis purposes
12. The last field shows the Unique id of the packet.

XGRAPH

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

Syntax:**Xgraph [options] file-name**

Options are listed here

`/-bd <color>` (Border)

This specifies the border color of the xgraph window.

`/-bg <color>` (Background)

This specifies the background color of the xgraph window.

`/-fg<color>` (Foreground)

This specifies the foreground color of the xgraph window.

`/-lf <fontname>` (LabelFont)

All axis labels and grid labels are drawn using this font.

`/-t<string>` (Title Text)

This string is centered at the top of the graph.

`/-x <unit name>` (XunitText)

This is the unit name for the x-axis. Its default is "X".

`/-y <unit name>` (YunitText)

This is the unit name for the y-axis. Its default is "Y".

Awk- An Advanced

Awk is a programmable, pattern-matching, and processing tool available in UNIX. It works equally well with text and numbers.

Awk is not just a command, but a programming language too. In other words, awk utility is a pattern scanning and processing language. It searches one or more files to see if they contain lines that match specified patterns and then perform associated actions, such as writing the line to the standard output or incrementing a counter each time it finds a match.

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 their choice. You can now print a serial number, using the variable kount, and apply it to 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 hold large awk programs in separate files 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 be 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"}' emp.lst

Part-A**Experiment No: 1****THREE NODE POINT TO POINT NETWORK**

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

```
set ns [new Simulator]           # Letter S is capital
set nf [open lab1.nam w]         # open a nam trace file in write mode
$ns namtrace-all $nf           # nf nam filename
set tf [open lab1.tr w]         # tf trace filename
$ns trace-all $tf

proc finish { } {
    global ns nf tf
    $ns flush-trace              # clears trace file contents
    close
    $nfclose
    $tf
    exec nam lab1.nam
    &exit 0
}
set n0 [$ns node]               # creates 3 nodes
set n2 [$ns node]
set n3 [$ns node]

$ns duplex-link $n0 $n2 200Mb 10ms DropTail # establishing links
$ns duplex-link $n2 $n3 1Mb 1000ms DropTail
$ns queue-limit $n0 $n2 10

set udp0 [new Agent/UDP]        # attaching transport layer protocols
$ns attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR] # attaching application layer protocols
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0

set null0 [new Agent/Null]      # creating sink(destination) node
$ns attach-agent $n3 $null0
$ns connect $udp0 $null0

$ns at 0.1 "$cbr0 start"
$ns at 1.0 "finish"
$ns run
```

AWK file: *(Open a new editor using “vi command” and write awk file and save with “.awk” extension)*

#immediately after BEGIN should open braces ‘{

```
BEGIN{ c=0;}
{
```

```
    if($1=="d")
```

```
{    c++;
    printf("%s\t%s\n", $5, $11);
}
}
END{ printf("The number of packets dropped is %d\n", c); }
```

Steps for execution

- Open gedit editor and type program. Program name should have the extension ".tcl"

```
[root@localhost ~]# gedit lab1.tcl
```

- Save the program and close the file.
- Open gedit editor and type **awk** program. Program name should have the extension ".awk"

```
[root@localhost ~]# gedit lab1.awk
```

- Save the program and close the file.
- Run the simulation program

```
[root@localhost ~]# ns lab1.tcl
```

- Here "**ns**" indicates network simulator. We get the topology shown in the snapshot.
- Now press the play button in the simulation window and the simulation will begin.
- After simulation is completed run **awk file** to see the output,

```
[root@localhost ~]# awk -f lab1.awk lab1.tr
```

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

```
[root@localhost ~]# gedit lab1.tr
```

Trace file contains 12 columns:

Event type, Event time, From Node, To Node, Packet Type, Packet Size, Flags (indicated by -----), Flow ID, Source address, Destination address, Sequence ID, Packet ID

Contents of Trace File

Topology

Output

Experiment No: 2
TRANSMISSION OF PING MESSAGE

***Aim:** 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 nf [ open lab2.nam w  
]  
$ns namtrace-all $nf
```

```
set tf [ open lab2.tr w ]  
$ns trace-all $tf
```

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

```
$ns duplex-link $n0 $n4 1005Mb 1ms DropTail  
$ns duplex-link $n1 $n4 50Mb 1ms DropTail  
$ns duplex-link $n2 $n4 2000Mb 1ms DropTail  
$ns duplex-link $n3 $n4 200Mb 1ms DropTail  
$ns duplex-link $n4 $n5 1Mb 1ms DropTail
```

```
set p1 [new Agent/Ping] # letters A and P should be capital  
$ns attach-agent $n0 $p1  
$p1 set packetSize_ 50000  
$p1 set interval_ 0.0001
```

```
set p2 [new Agent/Ping] # letters A and P should be capital  
$ns attach-agent $n1 $p2
```

```
set p3 [new Agent/Ping] # letters A and P should be capital  
$ns attach-agent $n2 $p3  
$p3 set packetSize_ 30000  
$p3 set interval_ 0.00001
```

```
set p4 [new Agent/Ping] # letters A and P should be capital  
$ns attach-agent $n3 $p4
```

```
set p5 [new Agent/Ping] # letters A and P should be capital  
$ns attach-agent $n5 $p5
```

```
$ns queue-limit $n0 $n4 5  
$ns queue-limit $n2 $n4 3
```

```
$ns queue-limit $n4 $n5 2
```



```
Agent/Ping instproc recv {from
rtt} {
$self instvar node_
puts "node [$node_id] received answer from $from with round trip time $rtt msec"
}
# please provide space between $node_ and id. No space between $ and from. No
spacebetween and $ and rtt */
```

```
$ns connect $p1 $p5
$ns connect $p3 $p4
```

```
proc finish {} {
global ns nf tf
$ns flush-
traceclose $nf
close $tf
exec nam lab2.nam
&exit 0
}
```

```
$ns at 0.1 "$p1 send"
$ns at 0.2 "$p1 send"
$ns at 0.3 "$p1 send"
$ns at 0.4 "$p1 send"
$ns at 0.5 "$p1 send"
$ns at 0.6 "$p1 send"
$ns at 0.7 "$p1 send"
$ns at 0.8 "$p1 send"
$ns at 0.9 "$p1 send"
$ns at 1.0 "$p1 send"
```

```
$ns at 0.1 "$p3 send"
$ns at 0.2 "$p3 send"
$ns at 0.3 "$p3 send"
$ns at 0.4 "$p3 send"
$ns at 0.5 "$p3 send"
$ns at 0.6 "$p3 send"
$ns at 0.7 "$p3 send"
$ns at 0.8 "$p3 send"
$ns at 0.9 "$p3 send"
$ns at 1.0 "$p3 send"
```

```
$ns at 2.0 "finish"
$ns run
```

AWK file: (Open a new editor using “gedit command” and write awk file and save with “awk” extension)

```
BEGIN{
drop=0;
}
```

```

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

```

Steps for execution

- Open gedit editor and type program. Program name should have the extension ".tcl"

```
[root@localhost ~]# gedit lab2.tcl
```

- Save the program and close the file.
- Open gedit editor and type **awk** program. Program name should have the extension ".awk"

```
[root@localhost ~]# gedit lab2.awk
```

- Save the program and close the file.
- Run the simulation program

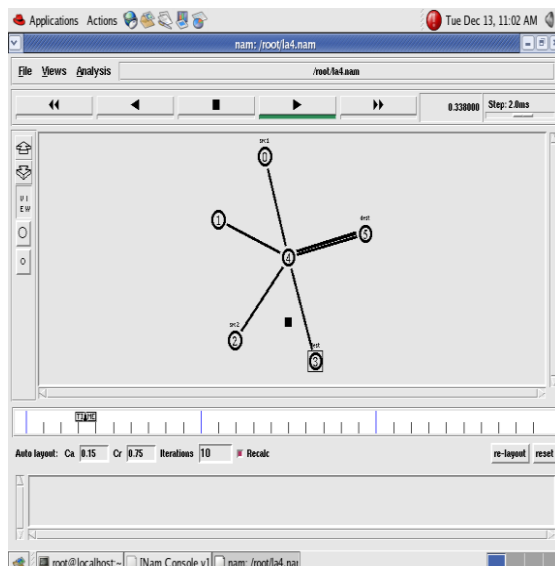
```
[root@localhost~]# ns lab2.tcl
```

- Here "ns" indicates network simulator. We get the topology shown in the snapshot.
- Now press the play button in the simulation window and the simulation will begin.
- After simulation is completed run **awk file** to see the output ,

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

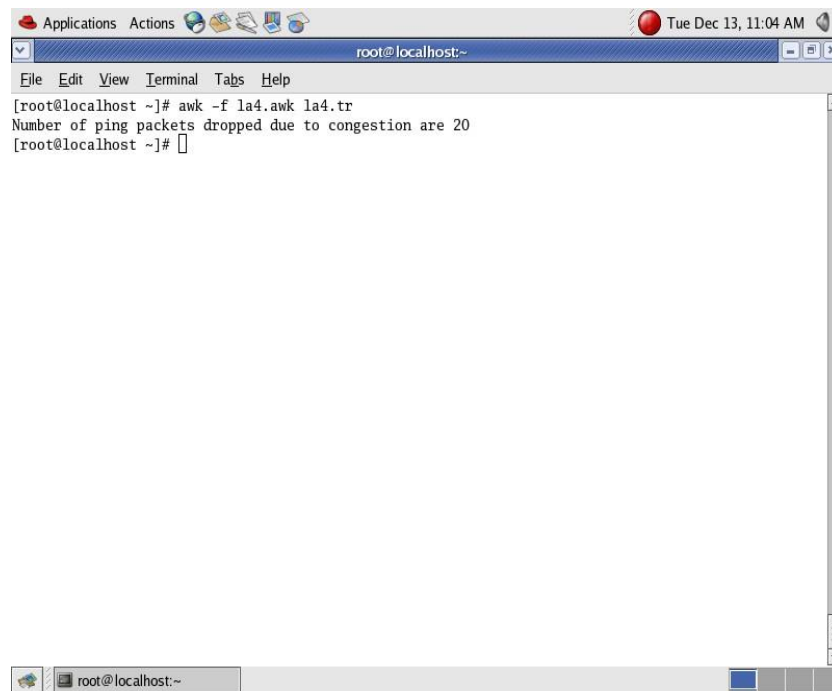
- To see the trace file contents open the file as ,

```
[root@localhost~]# gedit lab2.tr
```



Topology

Output



The screenshot shows a terminal window titled 'root@localhost:~'. The command executed is `awk -f la4.awk la4.tr`. The output is 'Number of ping packets dropped due to congestion are 20'. The terminal window has a menu bar with 'File', 'Edit', 'View', 'Terminal', 'Tabs', and 'Help'. The status bar at the bottom shows 'root@localhost:~'.

```
Applications Actions [Icons] Tue Dec 13, 11:04 AM
root@localhost:~
File Edit View Terminal Tabs Help
[root@localhost ~]# awk -f la4.awk la4.tr
Number of ping packets dropped due to congestion are 20
[root@localhost ~]#
```

Output

Experiment No: 3 ETHERNET LAN USING N-NODES WITH MULTIPLE TRAFFIC

Aim: *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" 100Mb 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_ # must put underscore ( _ ) after cwnd and no space between them
$tcp2 trace cwnd_
```

```
proc finish { } {
global ns nf tf
$ns flush-
tracelose $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 "gedit command" and write awk file and save with ".awk" extension)

cwnd:- means congestion

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

Steps for execution

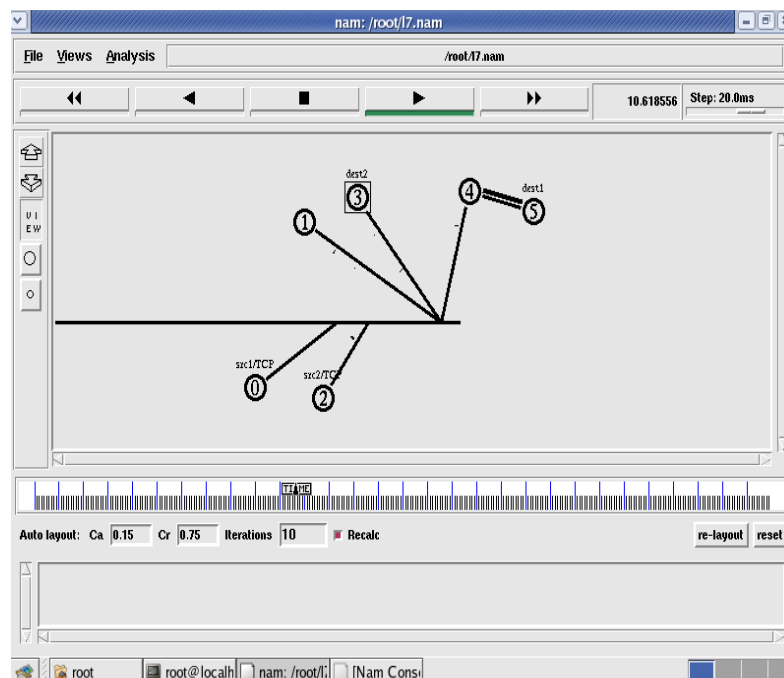
- Open gedit editor and type program. Program name should have the extension ".tcl"

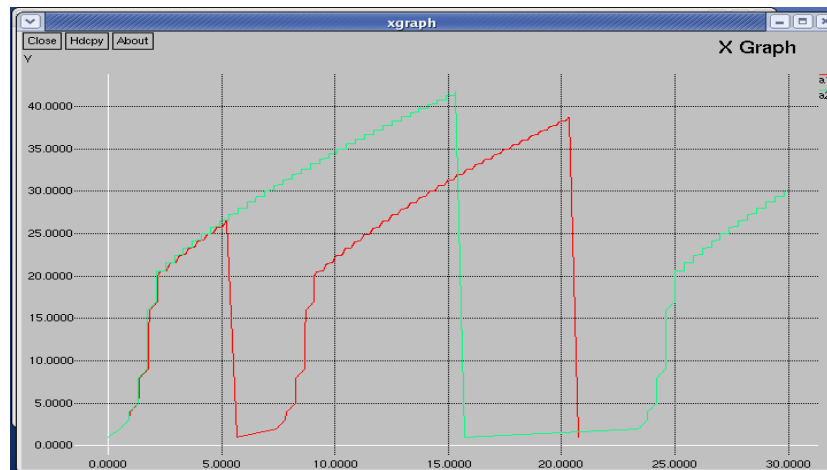

```
[root@localhost ~]# gedit lab3.tcl
```
- Save the program and close the file.
- Open gedit editor and type **awk** program. Program name should have the extension ".awk"


```
[root@localhost ~]# gedit lab3.awk
```
- Save the program and close the file.
- Run the simulation program


```
[root@localhost~]# ns lab3.tcl
```
- Here "**ns**" indicates network simulator. We get the topology shown in the snapshot.
- Now press the play button in the simulation window and the simulation will begin.
- 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 (>)**.
- To see the trace file contents open the file as ,

Topolgy:

Output:
Experiment No: 4
SIMPLE ESS WITH WIRELESS LAN

Aim: *Implement simple ESS and with transmitting nodes in wire-less LAN by simulation and determine the performance with respect to transmission of packets.*

```
set ns [new
Simulator] set tf
[open lab4.tr w]
$ns trace-all $tf
set topo [new Topography]
$topo load_flatgrid 1000
1000set nf [open lab4.nam
w]
$ns namtrace-all-wireless $nf 1000 1000
```

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

```
create-god 3
set n0 [$ns
node] set n1
[$ns node] set
n2 [$ns node]
```

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

\$n0 set X_ 50
\$n0 set Y_ 50
\$n0 set Z_ 0
\$n1 set X_ 100
\$n1 set Y_ 100
\$n1 set Z_ 0
\$n2 set X_ 600
\$n2 set Y_ 600
\$n2 set Z_ 0

\$ns at 0.1 "\$n0 setdest 50 50 15"
\$ns at 0.1 "\$n1 setdest 100 100 25"
\$ns at 0.1 "\$n2 setdest 600 600 25"


```
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
set sink1 [new Agent/TCPSink]
$ns attach-agent $n1 $sink1
$ns connect $tcp0 $sink1
```

```
set tcp1 [new Agent/TCP]
$ns attach-agent $n1 $tcp1
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
set sink2 [new Agent/TCPSink]
$ns attach-agent $n2 $sink2
$ns connect $tcp1 $sink2
```

```
$ns at 5 "$ftp0 start"
$ns at 5 "$ftp1 start"
```

```
$ns at 100 "$n1 setdest 550 550 15"
$ns at 190 "$n1 setdest 70 70
15"proc finish { } {
    global ns nf tf
    $ns flush-trace
    exec nam lab4.nam &
    close $tf
    exit 0
}
$ns at 250 "finish"
$ns run
```

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

```
BEGIN{
    count1=
    0
    count2=
    0
    pack1=0
    pack2=0
    time1=0
    time2=0
}
{
    if($1=="r"&&$3=="_1_"&&$4=="AGT")
    {
        count1++
        pack1=pack1+$
        8time1=$2
    }
    if($1=="r"&&$3=="_2_"&&$4=="AGT")
```

```

    {
        count2++
        pack2=pack2+$
        8time2=$2
    }
}

END{
printf("The Throughput from n0 to n1: %f Mbps \n", ((count1*pack1*8)/(time1*1000000)));
printf("The Throughput from n1 to n2: %f Mbps", ((count2*pack2*8)/(time2*1000000)));
}

```

Steps for execution

- Open gedit editor and type program. Program name should have the extension ".tcl"


```
[root@localhost ~]# gedit lab4.tcl
```
- Save the program and close the file.
- Open gedit editor and type **awk** program. Program name should have the extension ".awk"

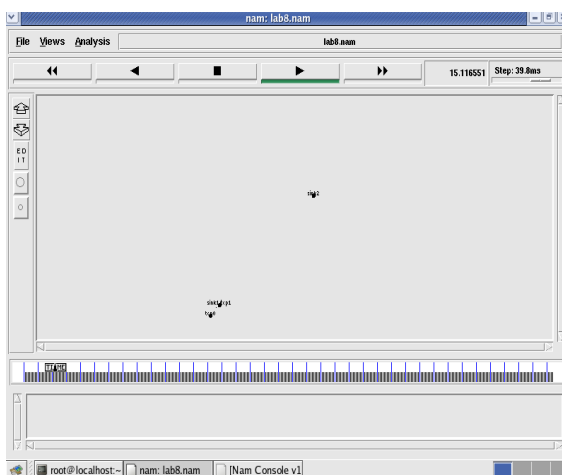

```
[root@localhost ~]# gedit lab4.awk
```
- Save the program and close the file.
- Run the simulation program


```
[root@localhost~]# ns lab4.tcl
```
- Here "**ns**" indicates network simulator. We get the topology shown in the snapshot.
- Now press the play button in the simulation window and the simulation will begin.
- After simulation is completed run **awk** file to see the output,

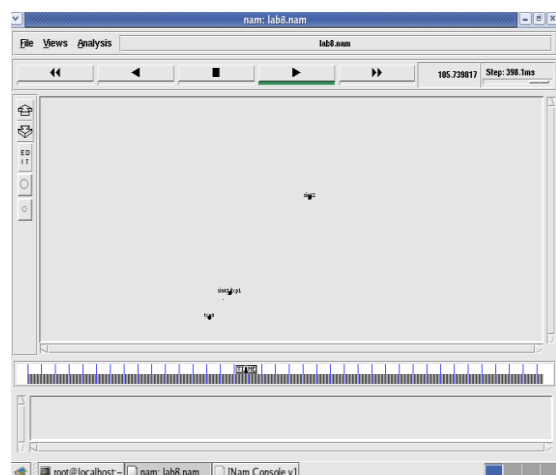

```
[root@localhost~]# awk -f lab4.awk lab4.tr
```
- To see the trace file contents open the file as,


```
[root@localhost~]# gedit lab4.tr
```

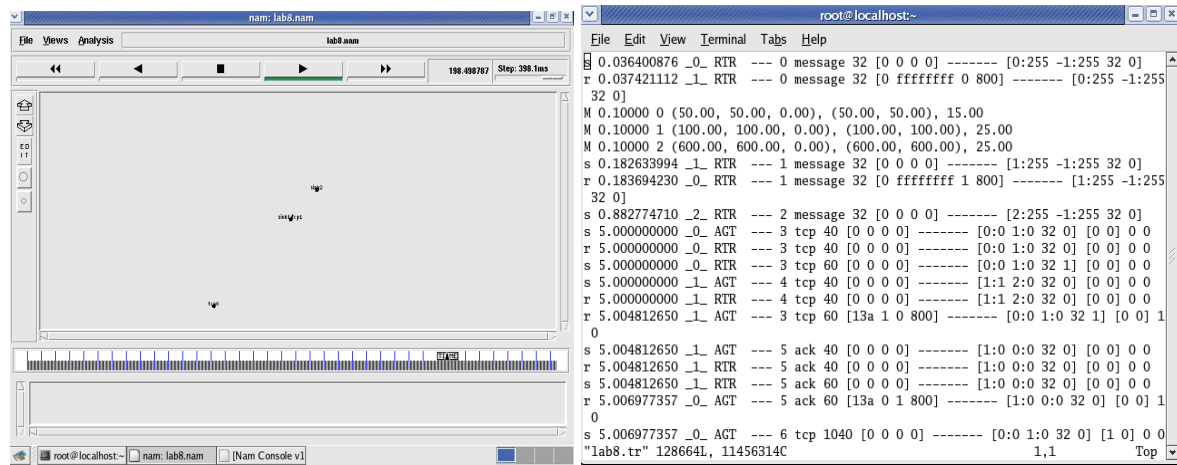
Output:



Node 1 and 2 are communicating



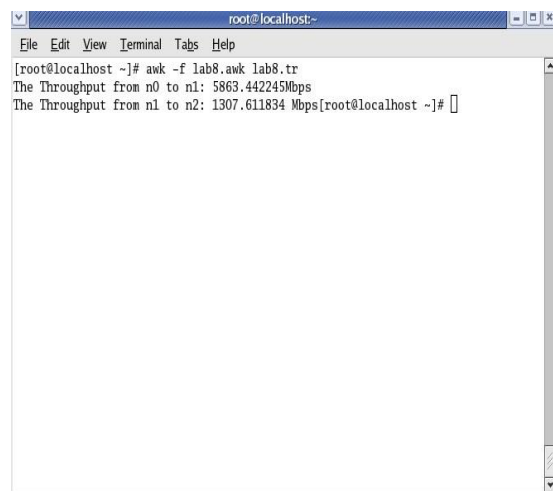
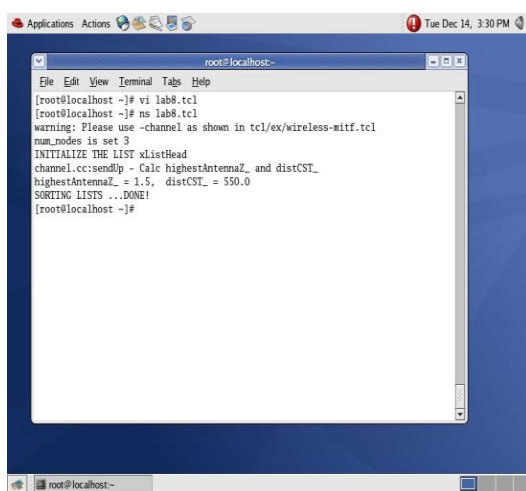
Node 2 is moving towards node 3



Node 2 is coming back from node 3 towards node1

Trace File

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



PART B

PROGRAM-1

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

```
#include<stdio.h>
#include<string.h>

#define N strlen(g)
char t[128],cs[128],g[]="100010000000100010001";
int a,e,c;
void xor()
{
    for(c=1;c<N;c++)
        cs[c]=((cs[c]==g[c])?'0':'1');
}
void crc()
{
    for(e=0;e<N;e++)
        cs[e]=t[e];
    do
    {
        if(cs[0]=='1')
            xor();
        for(c=0;c<N-1;c++)
            cs[c]=cs[c+1];
        cs[c]=t[e++];
    }
    while(e<=a+N-1);
}

void main()
{
    printf("enter data to be sent:");
    scanf("%s",t);
    printf("\n generating polynomial is:%s",g);
    a=strlen(t);
    for(e=a;e<a+N-1;e++)
        t[e]='0';
    printf("\n modified t[u] in:%s",t);
    crc();
    printf("\n checksum is:%s",cs);
    for(e=a;e<a+N-1;e++)
        t[e]=cs[e-a];
    printf("\n final codeword is:%s",t);
    printf("\n do you want to modify the data: 0(yes) 1(no)?");
    scanf("%d",&e);
    if(e==0)
    {
        printf("enter position where data is to be modified:");
        scanf("%d",&e);
        t[e]=(t[e]=='0')?'1':'0';
        printf("errorneous data:%s\n",t);
    }
    crc();
    for(e=0;(e<N-1)&&(cs[e]!='1');e++);
```

```
if(e<N-1)
    printf("error detected");
else
    printf("error not detected");
}
```

Output:

1. enter data to be sent:1011101

generating polynomial is:10001000000100010001
modified t[u] in:1011101000000000000000000000
checksum is:1000101110111011000
final codeword is:10111011000101110111011000
do you want to modify the data: 0(yes) 1(no)?:0
enter position where data is to be modified:3
errorneous data:10101011000101110111011000
error detected

2. enter data to be sent:1011101

generating polynomial is:10001000000100010001
modified t[u] in:1011101000000000000000000000
checksum is:1000101110111011000
final codeword is:10111011000101110111011000
do you want to modify the data: 0(yes) 1(no)?:1
error not detected

PROGRAM-2

Write a C/C++ program for simple RSA Algorithm to Encrypt and Decrypt the Data.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
```

```
char msg[100];
long int p,q,n,t,e,d;
long int pt[100],ct[100];
long int i,j;
```

```
void calce();
void calcd();
void encrypt();
void decrypt();
long int gcd (long int,long int);
long int prime (long int);
```

```
void main()
{
```

```
printf("\n enter first prime number:\n");
scanf("%d", & p );

if(!prime(p))
{
    printf("\n Wrong Input \n");
    exit(0);
}

printf("\n enter another prime number:\n");
scanf("%d", & q);

if(!prime(q)|| p==q)
{
    printf("\n Wrong Input \n");

    exit(0);
}

printf("Enter Message :\n");
scanf("%s", msg);
for(i=0; msg[i]!=NULL;i++)
    pt[i]=msg[i];
pt[i]=NULL;

n=p*q;

t=(p-1) * (q-1);

calce();
calcd();

printf("\n value of p = %d \n",p);
printf("\n value of q = %d \n",q);
printf("\n value of n = %d \n",n);
printf("\n value of t = %d \n",t);
printf("\n value of e = %d \n",e);
printf("\n value of d = %d \n",d);

encrypt();
decrypt();
}

void calce()
{
    for(i=2;i<t;i++)
    {
        if(gcd(i,t)==1)
            e=i;
    }
}
```

```
void calcd()
{
    e=e%t;

    for(i=1; i<t;i++)
    {
        if((e*i)%t==1)
            d=i;
    }
}

void encrypt()
{
    long int k;
    for(i=0; i<strlen(msg);i++)
    {

        k=1;
        for(j=1;j<=e;j++)
        {
            k=(k*pt[i])%n;
        }
        ct[i]=k;

    }
    ct[i]=NULL;

    printf("\n The encrypted message is :\n");

    for(i=0;ct[i]!=NULL;i++)
        printf("%c", ct[i]);
}

void decrypt()
{
    long int k;
    for(i=0;i<strlen(msg);i++)
    {
        k=1;
        for(j=1;j<=d;j++)
        {
            k=(k*ct[i])%n;
        }
        pt[i]=k;
    }
    pt[i] = NULL;

    printf("\n The decrypted message is:");

    for(i=0;pt[i]!=NULL;i++)
        printf("%c",pt[i]);
```

```
}

long int gcd(long int a, long int b)
{
    long int c;
    while(a!=0)
    {
        c=a;
        a=b%a;
        b=c;
    }
    return b;
}

long int prime (long int pr)
{
    int i;
    j=sqrt(pr);

    for(i=2;i<=j;i++)
    {
        if(pr % i ==0)
            return 0;
    }
    return 1;
}
```

OUTPUT

enter first prime number:
11

enter another prime number:
13
Enter Message :
harshitha

value of p = 11

value of q = 13

value of n = 143

value of t = 120

value of e = 119

value of d = 119

The encrypted message is :

sEa OZ s

The decrypted message is: harshitha

PROGRAM -3

Write a C/C++ program for Distance Vector Algorithm to find suitable path for transmission.

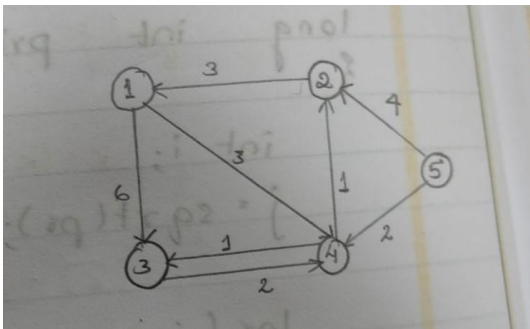
```
#include<stdio.h>
struct node
{
    unsigned dist[20];
    unsigned from[20];
}
rt[10];
void main()
{
    int costmat[20][20];
    int nodes,i,j,k,count=0;

    printf("\n enter the numbers of nodes:");
    scanf("%d",&nodes);
    printf("\n enter the cost matrix:\n");
    for(i=0;i<nodes;i++)
    {
        for(j=0;j<nodes;j++)
        {
            scanf("%d",&costmat[i][j]);
            costmat[i][i]=0;
            rt[i].dist[j]=costmat[i][j];
            rt[i].from[j]=j;
        }
    }
do
{
    count=0;
    for(i=0;i<nodes;i++)
    for(j=0;j<nodes;j++)
    {
        for(k=0;k<nodes;k++)
        {
            if(rt[i].dist[j]>costmat[i][k]+rt[k].dist[j])
            {
                rt[i].dist[j]=rt[i].dist[k]+rt[k].dist[j];
                rt[i].from[j]=k;
                count++;
            }
        }
    }
}
```

```
}  
}  
while(count!=0);  
printf("\n\n SOLUTION:\n\n");  
for(i=0;i<nodes;i++)  
{  
    printf("\n\n for router %d \n",i+1);  
    for(j=0;j<nodes;j++)  
    {  
        printf("\t\n node %d via %d distance %d",j+1,rt[i].from[j]+1,rt[i].dist[j]);  
    }  
}  
printf("\n\n");  
}
```

OUTPUT

enter the numbers of nodes:5



enter the cost matrix:

```
0 99 6 3 99  
3 0 99 99 99  
99 99 0 2 99  
99 1 1 0 99  
99 4 99 2 0
```

SOLUTION:

for router 1

node 1 via 1 distance 0
node 2 via 4 distance 4
node 3 via 4 distance 4
node 4 via 4 distance 3
node 5 via 5 distance 99

for router 2

node 1 via 1 distance 3
node 2 via 2 distance 0
node 3 via 1 distance 7

node 4 via 1 distance 6
node 5 via 5 distance 99

for router 3

node 1 via 4 distance 6
node 2 via 4 distance 3
node 3 via 3 distance 0
node 4 via 4 distance 2
node 5 via 5 distance 99

for router 4

node 1 via 2 distance 4
node 2 via 2 distance 1
node 3 via 3 distance 1
node 4 via 4 distance 0
node 5 via 5 distance 99

for router 5

node 1 via 4 distance 6
node 2 via 4 distance 3
node 3 via 4 distance 3
node 4 via 4 distance 2
node 5 via 5 distance 0

PROGRAM-4

Write a C/C++ program for congestion control using Leaky Bucket Algorithm.

```
#include<iostream>
#include<stdlib.h>
#include<time.h>
#include<unistd.h>

#define bucketSize 512

using namespace std;

void bktInput(int a,int b)
{
    if(a>bucketSize)
    {
        cout<<"\n\t Bucket Overflow";
    } else {
        usleep(500);
        while(a>b)
        {
            cout<<"\n\t"<<b<<" bytes outputed";
            a=a-b;
            usleep(500);
        }
        if(a>0)
```

```
{
    cout<<"\n\tlast "<<a<<" bytes sent\t";
    cout<<"\n\t\tBucket output successful";
}
}
}

int main()
{
    int op,pktsize;
    srand(time(NULL));
    cout<<"Enter output rate: ";
    cin>>op;
    for(int i=1;i<=5;i++)
    {
        usleep(rand() % 1000);
        pktsize = rand() % 1000;
        cout<<"\nPacket no = "<<i<<" Packet size = "<<pktsize;
        bktInput(pktsize,op);
    }
    return 0;
}
```

Output:

Enter output rate: 500

Packet no = 1 Packet size = 304

last 304 bytes sent

Bucket output successful

Packet no = 2 Packet size = 527

Bucket Overflow

Packet no = 3 Packet size = 702

Bucket Overflow

Packet no = 4 Packet size = 634

Bucket Overflow

Packet no = 5 Packet size = 467

last 467 bytes sent

Bucket output successful

PROGRAM-5

Using TCP/IP write a Client-Server program to make-client send the file to server and to make server to send back the contents to the requested file if present.(Client-Server using TCP/IP sockets)

Client-server using TCP/IP sockets

Aim: *Using TCP/IP Sockets, write a client-server program to make client sending the file name and the server to send back the contents of the requested file if present. Implement the above program using as message queues or FIFOs as IPC channels.*

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.

Source Code:**TCP Server**

The server program has three responsibilities which must be fulfilled in the code. First job is to read the file name coming from client. For this, it uses input stream. Second one is to open the file, using some input stream, and read the contents. Third one is, as the reading is going on, to send the contents each line separately.

```
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 waiting for chatting");
        // 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;
        // reading line-by-line from file
        while((str = contentRead.readLine()) != null)
        {
            pwrite.println(str); // sending each line to client
        }
        sock.close();
        sersock.close(); // closing network sockets
        pwrite.close();
        fileRead.close();
        contentRead.close();
    }
}
```

TCP Client:

To read filename as input from keyboard, remember, this file should exist on server. For this, it uses input stream. The file read from keyboard should be sent to the server. For this client uses output stream. The file contents sent by the server, the client should receive and print on the console.

BufferedReader:

The System.in is a byte stream and cannot be chained to BufferedReader as BufferedReader is a character stream. The byte stream System.in is to be converted (wrapped) into a character stream and then passed to BufferedReader constructor. To take input from the keyboard, a [BufferedReader](#) object, keyRead, is created. To send the file name to the server, pwrite of [PrintWriter](#) and to receive the file contents from the server, socketRead of [BufferedReader](#) are created.

```
PrintWriter pwrite = new PrintWriter(ostream, true);
```

```
import java.net.*;
import java.io.*;
public class ContentsClient
{
    public static void main( String args[ ] ) throws Exception
    {
        Socket sock = new Socket( "127.0.0.1", 4000);
        // reading the file name from keyboard. Uses input stream
        System.out.print("Enter the file name");
        BufferedReader keyRead = new BufferedReader(new InputStreamReader(System.in));
        String fname = keyRead.readLine();
        // sending the file name to server. Uses PrintWriter
        OutputStream ostream = sock.getOutputStream( );
        PrintWriter pwrite = new PrintWriter(ostream, true);
        pwrite.println(fname);
        // receiving the contents from server. Uses inputStream
        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:**At server side:**

```
[root@localhost]# javac ContentsServer.java
```

```
[root@localhost]# java ContentsServer
```

Server ready for connection

Connection is successful and waiting for chatting

```
[root@localhost]# javac ContentsClient.java
```

```
[root@localhost]# java ContentsClient
```

Enter the file

nameaa.txt

Welcome to Network Lab

PROGRAM-6

Implement the fifth program using message queue or FIFO as IPC Channels. (Client- Server Communication)

Client-Server Communication

Aim: Write a program on datagram socket for client/server to display the messages on clientside, 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.

Here, the connectionServer.java program creates a ServerSocket object bound to port

3456 and waits for an incoming client connection request. When a client contacts the server program, the `accept()` method is unblocked and returns a `Socket` object for the server to communicate with the particular client that contacted.

The server program then creates a `PrintStream` object through the output stream extracted from this socket and uses it to send a welcome message to the contacting client. The client program runs as follows: The client creates a `Socket` object to connect to the server running at the specified IP address or hostname and at the port number 3456. The client creates a `BufferedReader` object through the input stream extracted from this socket and waits for an incoming line of message from the other end. The `readLine()` method of the `BufferedReader` object blocks the client from proceeding further unless a line of message is received. The purpose of the `flush()` method of the `PrintStream` class is to write any buffered output bytes to the underlying output stream and then flush that stream to send out the bytes. Note that the server program in our example sends a welcome message to an incoming client request and then stops.

Source Code:**UDP Client**

```
import
java.net.*;import
java.io.*;
class connectionClient
{
    public static void main(String[ ] args)
    {
        try

            {
                InetAddress acceptorHost = InetAddress.getByName(args[0]);
                int serverPortNum = Integer.parseInt(args[1]);
                Socket clientSocket = new Socket(acceptorHost, serverPortNum);
                BufferedReader br = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
                System.out.println(br.readLine( ));
                clientSocket.close();
            }
        catch(Exception e)
        { e.printStackTrace( );
        }
    }
}
```

UDP Server

```
import
java.net.*;import
java.io.*;
class connectionServer
{
    public static void main(String[ ] args)
    {
        try
        {
            String message = args[0];
            int serverPortNumber = Integer.parseInt(args[1]);
            ServerSocket connectionSocket = new ServerSocket(serverPortNumber);
            Socket dataSocket = connectionSocket.accept();
            PrintStream socketOutput = new PrintStream(dataSocket.getOutputStream());
            socketOutput.println(message);
            System.out.println("sent response to client");
            socketOutput.flush( );
            dataSocket.close( );
        }
    }
}
```

```
        connectionSocket.close( );
    }
    catch(Exception e)
    { e.printStackTrace( );
    }
}

}
```

Output:

Server Side

[root@localhost]# **Javac ConnectionServer.java**

[root@localhost]# **Java ConnectionServer "Welcome to Computer Network Lab"3956**

Client Side:

[root@localhost]# **Javac ConnectionClient.java**

[root@localhost]# **Java ConnectionClient localhost 3956**
Welcome to Computer Network Lab

