Channabasaveshwara Institute of Technology

(Affiliated to VTU, Belgaum & Approved by AICTE, New Delhi)

(NACC Accredited ISO 9001:2015 Certified Institution)

NH 206 (B.H. Road), Gubbi, Tumkur – 572 216.

Karnataka.

Computer Science & Engineering

V SEMESTER

2025-26(ODD)

COMPUTER NETWORK LAB MANUAL

SHOBHA AGASIBAGIL

Assistant Professor

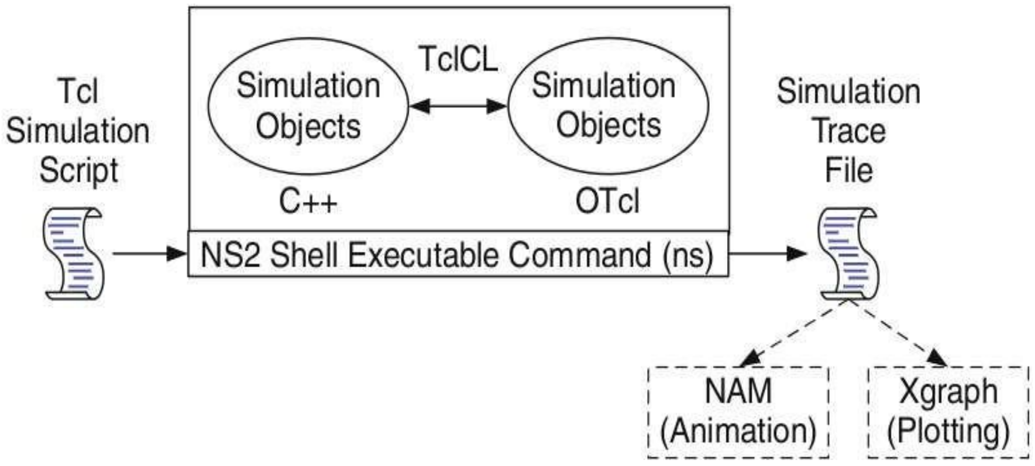
Department of Computer Science and Engineering

CIT GUBBI

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 set command, you can call this variable as you wish, In general people declares it as ns because it is an instance of the Simulator class, so an object the code[new Simulator] is indeed the installation of the class Simulator using the reserved word new.

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

#Open the Trace file

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

$ns trace-all $tracefile1
```

#Open the NAM trace file

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

$ns namtrace-all $namfile
```

The above creates a trace file called "out.tr" and a nam visualization trace file called "out.nam". Within the tcl script, these files are not called explicitly by their names, but instead by pointers that are declared above and called "tracefile1" and "namfile" respectively. Remark that they begins with a # symbol. The second line open the file "out.tr" to be used for writing, declared with the letter "w". The third line uses a simulator method called trace-all that have as parameter the name of the file where the traces will go.

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

The termination of the program is done using a "finish" procedure.

#Define a 'finish' procedure

```
Proc finish { } {

global ns tracefile1 namfile

$ns flush-trace

Close $tracefile1

Close $namfile

Exec nam out.nam &

Exit 0

}
```

The word proc declares a procedure in this case called **finish** and without arguments. The word **global** is used to tell that we are using variables declared outside the procedure. The simulator method "**flush-trace"** will dump the traces on the respective files. The tcl command "**close"** closes the trace files defined before and **exec** executes the nam program for visualization. The command **exit** will ends the application and return the number 0 as status to the system. Zero is the default for a clean exit. Other values can be used to say that is a exit because something fails.

At the end of ns program we should call the procedure "finish" and specify at what time the termination should occur. For example,

```
$ns at 125.0 "finish"
```

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

The simulation can then begin using the command

```
$ns run
```

Definition of a network of links and nodes

The way to define a node is

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

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:

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

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

The command

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

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

#Setup a UDP connection

```
set udp [new Agent/UDP]

$ns attach-agent $n1 $udp

set null [new Agent/Null]

$ns attach-agent $n5 $null

$ns connect $udp $null

$udp set fid_2
```

#setup a CBR over UDP connection

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

$cbr attach-agent $udp

$cbr set packetsize_ 100

$cbr set rate_ 0.01Mb

$cbr set random_ false
```

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

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

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

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

CBR over UDP

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

Instead of defining the rate in the command $cbr set rate_ 0.01Mb, one can define the time interval between transmission of packets using the command.

```
$cbr set interval_ 0.005
```

The packet size can be set to some value using

```
$cbr set packetSize_ <packet size>
```

Scheduling Events

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

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

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

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

$ns at 1.0 " $ftp start"

$ns at 124.0 "$ftp stop"

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

Structure of Trace Files

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

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 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

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
```

Experiment No: 1 **Date:**

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 $nf
        close $tf
        exec nam lab1.nam &
        exit 0
}
set n0 [$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
```

<u>AWK file:</u> *(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")
```

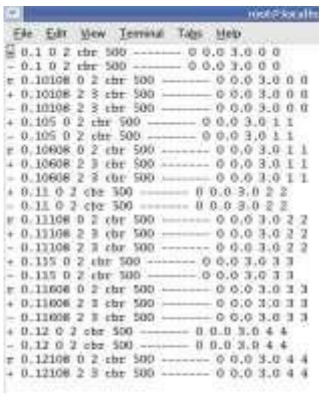
```
 {      c++;
        printf("%s\t%s\n",$5,$11);
  }
 }
END{ printf("The number of packets dropped is %d\n",c); }
```

Steps for execution

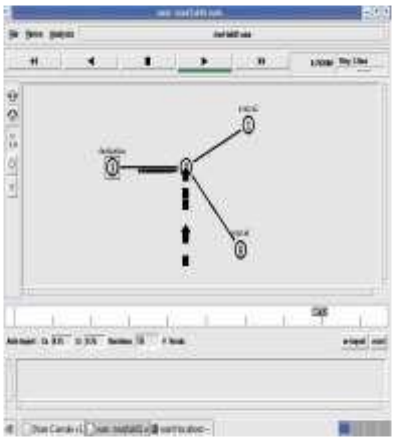
- *Open gedit editor and type program. Program name should have the extension " **.tcl** "*
  [root@localhost ~]# gedit lab1.tcl
- *Save the program and close the file.*
- *Open gedit editor and type **awk** program. Program name should have the extension "**.awk** "*
  [root@localhost ~]# gedit lab1.awk
- *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 begins.*
- *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

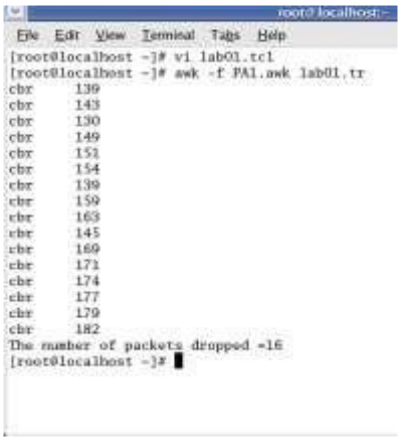
Trace file contains 12 columns:

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



```
+ 0.1 0 2 cbr 500 -------- 0 0.0 3.0 0 0
- 0.1 0 2 cbr 500 -------- 0 0.0 3.0 0 0
r 0.10108 0 2 cbr 500 -------- 0 0.0 3.0 0 0
+ 0.10108 2 3 cbr 500 -------- 0 0.0 3.0 0 0
- 0.10108 2 3 cbr 500 -------- 0 0.0 3.0 0 0
+ 0.105 0 2 cbr 500 -------- 0 0.0 3.0 1 1
- 0.105 0 2 cbr 500 -------- 0 0.0 3.0 1 1
r 0.10608 0 2 cbr 500 -------- 0 0.0 3.0 1 1
+ 0.10608 2 3 cbr 500 -------- 0 0.0 3.0 1 1
- 0.10608 2 3 cbr 500 -------- 0 0.0 3.0 1 1
+ 0.11 0 2 cbr 500 -------- 0 0.0 3.0 2 2
- 0.11 0 2 cbr 500 -------- 0 0.0 3.0 2 2
r 0.11108 0 2 cbr 500 -------- 0 0.0 3.0 2 2
+ 0.11108 2 3 cbr 500 -------- 0 0.0 3.0 2 2
- 0.11108 2 3 cbr 500 -------- 0 0.0 3.0 2 2
+ 0.115 0 2 cbr 500 -------- 0 0.0 3.0 3 3
- 0.115 0 2 cbr 500 -------- 0 0.0 3.0 3 3
r 0.11608 0 2 cbr 500 -------- 0 0.0 3.0 3 3
+ 0.11608 2 3 cbr 500 -------- 0 0.0 3.0 3 3
- 0.11608 2 3 cbr 500 -------- 0 0.0 3.0 3 3
+ 0.12 0 2 cbr 500 -------- 0 0.0 3.0 4 4
- 0.12 0 2 cbr 500 -------- 0 0.0 3.0 4 4
r 0.12108 0 2 cbr 500 -------- 0 0.0 3.0 4 4
+ 0.12108 2 3 cbr 500 -------- 0 0.0 3.0 4 4
```





```
[root@localhost ~]# vi lab01.tcl
[root@localhost ~]# awk -f PA1.awk lab01.tr
cbr     139
cbr     143
cbr     130
cbr     149
cbr     151
cbr     154
cbr     139
cbr     159
cbr     163
cbr     145
cbr     169
cbr     171
cbr     174
cbr     177
cbr     179
cbr     182
The number of packets dropped =16
[root@localhost ~]# 
```

Contents of Trace File Topology Output

Experiment No: 2 **Date:**

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 space
between and $ and rtt */

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

proc finish { } {
global ns nf tf
$ns flush-trace
close $nf
close $tf
exec nam lab2.nam &
exit 0
}
$ns at 0.1 "$p1 send"
$ns at 0.2 "$p1 send"
$ns at 0.3 "$p1 send"
$ns at 0.4 "$p1 send"
$ns at 0.5 "$p1 send"
$ns at 0.6 "$p1 send"
$ns at 0.7 "$p1 send"
$ns at 0.8 "$p1 send"
$ns at 0.9 "$p1 send"
$ns at 1.0 "$p1 send"

$ns at 0.1 "$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);
}
```

<u>Steps for execution</u>

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

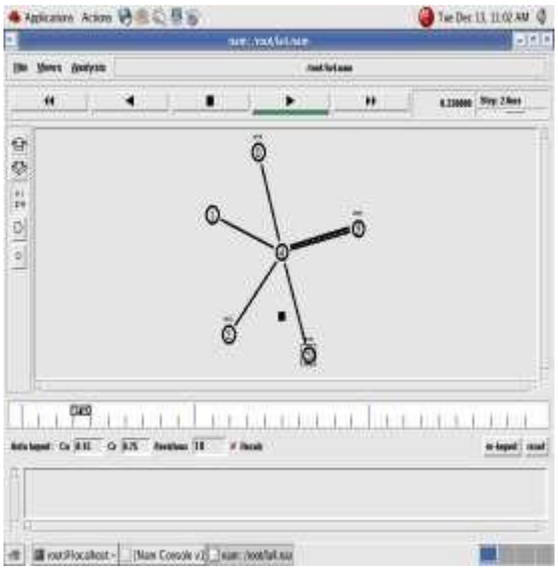
  [root@localhost ~]# gedit lab2.tcl
- *Save the program and close the file.*
- *Open gedit editor and type **awk** program. Program name should have the extension "**.awk** "*

  [root@localhost ~]# gedit lab2.awk
- *Save the program and close the file.*
- *Run the simulation program*

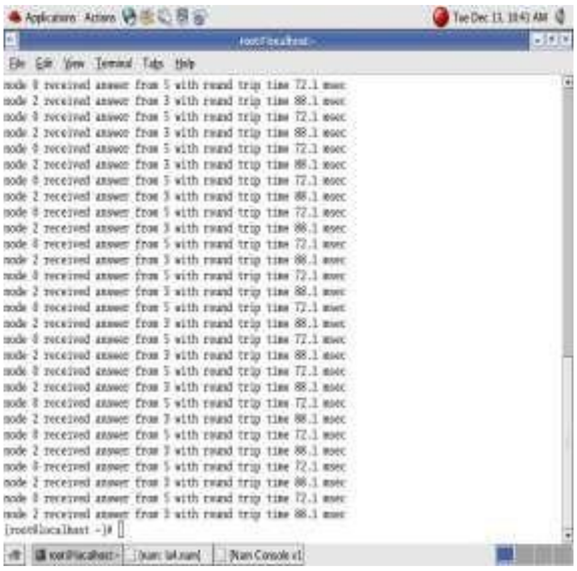
  [root@localhost~]# ns lab2.tcl
- *Here **"ns"** indicates network simulator. We get the topology shown in the snapshot.*
- *Now press the play button in the simulation window and the simulation will begins.*
- *After simulation is completed run **awk file** to see the output ,*

  [root@localhost~]# awk –f lab2.awk lab2.tr
- *To see the trace file contents open the file as ,*

  [root@localhost~]# gedit lab2.tr

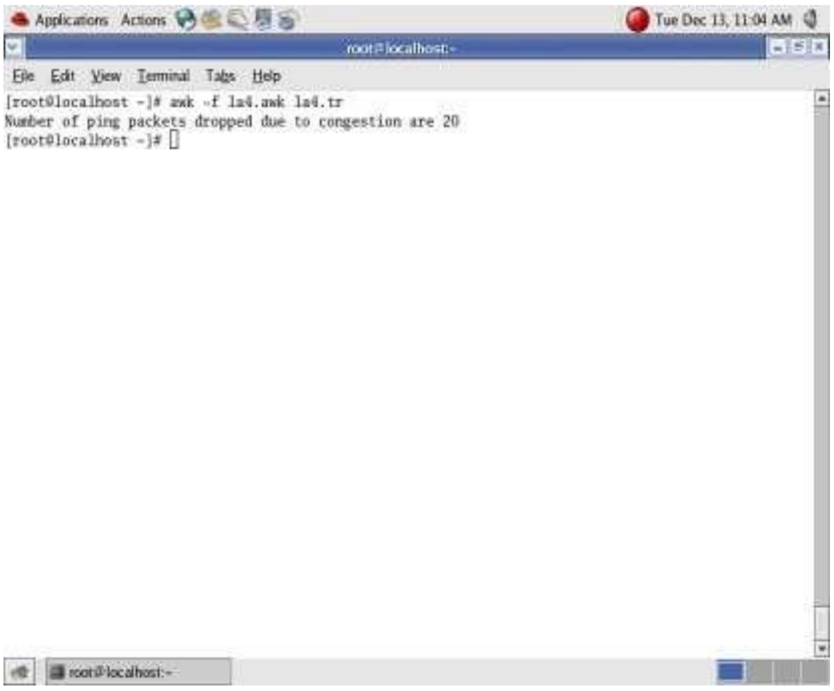


Topology



```
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
node 0 received answer from 5 with round trip time 72.1 msec
node 2 received answer from 3 with round trip time 88.1 msec
[root@localhost ~]#
```

Output



```
[root@localhost ~]# awk -f la4.awk la4.tr
Number of ping packets dropped due to congestion are 20
[root@localhost ~]# 
```

Output

Experiment No: 3 **Date:**

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-trace
close $tf
close $nf
exec nam lab3.nam &
exit 0
}

$ns at 0.1 "$ftp0 start"
$ns at 5 "$ftp0 stop"
$ns at 7 "$ftp0 start"
$ns at 0.2 "$ftp2 start"
$ns at 8 "$ftp2 stop"
$ns at 14 "$ftp0 stop"
$ns at 10 "$ftp2 start"
$ns at 15 "$ftp2 stop"

$ns at 16 "finish"
$ns run
```

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

cwnd:- means congestion window

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

Steps for execution

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

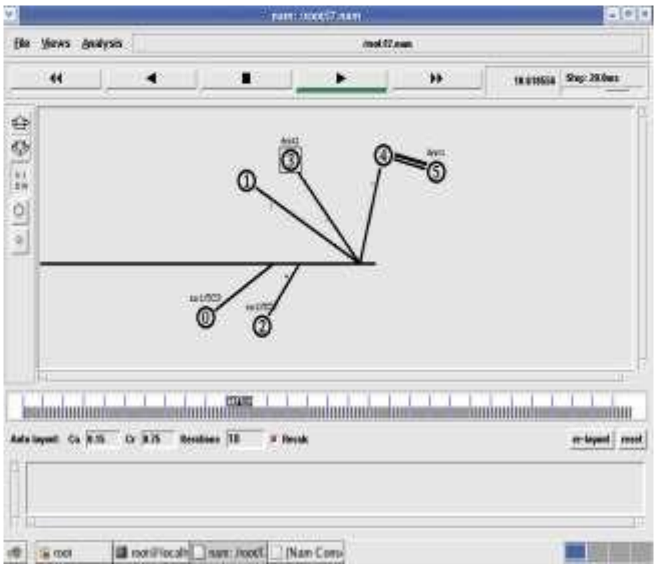
  [root@localhost~]# awk –f lab3.awk file1.tr > a1

  [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: 1 **Date:**

Error Detecting Code Using CRC-CCITT (16-bit)

Aim: *Write a Program for ERROR detecting code using CRC-CCITT (16bit).*

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.

```
                    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
```

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.

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.

Table 1: International Standard CRC Polynomials

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

Error detection with CRC

Consider a message represented by the polynomial M(x)

Consider a *generating polynomial* G(x)

This is used to generate a CRC = C(x) to be appended to M(x).

Note this G(x) is prime.

Steps:

1. Multiply M(x) by highest power in G(x). i.e. Add So much zeros to M(x).
2. Divide the result by G(x). The remainder = C(x).
   Special case: This won't work if bitstring =all zeros. We don't allow such an M(x).But M(x) bitstring = 1 will work, for example. Can divide 1101 into 1000.
3. If: x div y gives remainder c
   that means: x = n y + c
   Hence (x-c) = n y
   (x-c) div y gives remainder 0
   Here (x-c) = (x+c)
   Hence (x+c) div y gives remainder 0
4. Transmit: T(x) = M(x) + C(x)
5. Receiver end: Receive T(x). Divide by G(x), should have remainder 0.

Note if G(x) has order n - highest power is x^n,
then G(x) will cover (n+1) bits
and the *remainder* will cover n bits.
i.e. Add n bits (Zeros) to message.

Some CRC polynomials that are actually used

Some CRC polynomials

- CRC-8:
  x^8+x^2+x+1
  - Used in: 802.16 (along with error *correction*).
- CRC-CCITT:
  $x^{16}+x^{12}+x^5+1$
  - Used in: HDLC, SDLC, PPP default
- IBM-CRC-16 (ANSI):
  $x^{16}+x^{15}+x^2+1$

- 802.3:
  $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x+1$
  - Used in: Ethernet, PPP rootion

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])
                        {
                                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 Reveived 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)
                {
                        System.out.println("ERROR in Received data");
                        return;
                }
                System.out.println("no error");
        }

                }
        }
```

Output:

```
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 Received Data
```

****

Experiment No: 2 **Date:**

Bellman-ford Algorithm

Aim: *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 upd 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

Implementation Algorithm:

1. send my routing table to all my neighbors whenever my link table changes
2. when I get a routing table from a neighbor on port P with link metric M:
    a. add L to each of the neighbor's metrics
    b. for each entry (D, P', M') in the updated neighbor's table:
        i. if I do not have an entry for D, add (D, P, M') to my routing table

    ii. if I have an entry for D with metric M", add (D, P, M') to my routing table if M' < M"

3. if my routing table has changed, send all the new entries to all my neighbors.

Source Code:

```
import java.util.Scanner;
public class BellmanFord
{	private int D[];
	private int num_ver;
	public static final int MAX_VALUE = 999;
	public BellmanFord(int 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 Weighted 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
```

Experiment No: 3 **Date:**

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
        {
        ServerSocket sersock = new ServerSocket(4000);
        System.out.println("Server ready for connection");
        Socket sock = sersock.accept();
        System.out.println("Connection is successful and waiting for chatting");
        InputStream istream = sock.getInputStream( );
        BufferedReader fileRead =new BufferedReader(new InputStreamReader(istream));
        String fname = fileRead.readLine( );
        BufferedReader contentRead = new BufferedReader(new FileReader(fname) );
        OutputStream ostream = sock.getOutputStream( );
        PrintWriter pwrite = new PrintWriter(ostream, true);
```

```
            String str;
            while((str = contentRead.readLine()) !=  null)
             {
                        pwrite.println(str
             }
            sock.close();
            sersock.close();
            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();
        OutputStream ostream = sock.getOutputStream( );
        PrintWriter pwrite = new PrintWriter(ostream, true);
        pwrite.println(fname);
        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
```

At Client Side:

```
[root@localhost]# javac ContentsClient.java
[root@localhost]# java ContentsClient
                Enter the file name
                aa.txt
                Welcome to Network Lab
```

Experiment No: 4 **Date:**

Client-Server Communication

Aim: 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.

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)throws Exception
        {
        try
        {
        InetAddress host = InetAddress.getByName("localhost");
        Socket clientSocket = new Socket(host, 64327);
        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)throws Exception
        {
        try
        {
                ServerSocket sersocket = new ServerSocket(64327);
                Socket socket = sersocket.accept();
                PrintStream pstream = new PrintStream(socket.getOutputStream());
                pstream.println("WELCOME TO COMPUTER NETWORK LAB");
                System.out.println("sent response to client");
                pstream.flush( );
                socket.close( );
                sersocket.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
```

Experiment No: 5 **Date:**

RSA Algorithm to Encrypt and Decrypt the Data

Aim: C Program for Simple RSA Algorithm to encrypt and decrypt the data

The RSA algorithm can be used for both public key encryption and digital signatures. Its security is based on the difficulty of factoring large integers.

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

A very simple example of RSA encryption

This is an extremely simple example using numbers you can work out on a pocket calculator (those of you over the age of 35 can probably even do it by hand on paper).

1. Select primes p = 11, q = 3.
2. n = pq = 11.3 = 33
   phi = (p-1)(q-1) = 10.2 = 20
3. Choose e=3
   Check gcd(e, p-1) = gcd(3, 10) = 1 (i.e. 3 and 10 have no common factors except 1),
   and check gcd(e, q-1) = gcd(3, 2) = 1
   therefore gcd(e, phi) = gcd(e, (p-1)(q-1)) = gcd(3, 20) = 1
4. Compute d such that ed ≡ 1 (mod phi)
   i.e. compute d = $e^{\wedge -1}$ mod phi = $3^{\wedge -1}$ mod 20
   i.e. find a value for d such that phi divides (ed-1)
   i.e. find d such that 20 divides 3d-1.
   Simple testing (d = 1, 2, ...) gives d = 7
   Check: ed-1 = 3.7 - 1 = 20, which is divisible by phi.
5. Public key = (n, e) = (33, 3)
   Private key = (n, d) = (33, 7).

This is actually the smallest possible value for the modulus n for which the RSA algorithm works.

Now say we want to encrypt the message m = 7,

c = m$^{\wedge e}$ mod n = 7$^{\wedge 3}$ mod 33 = 343 mod 33 = 13.

Hence the ciphertext c = 13.

To check decryption we compute

m' = c$^{\wedge d}$ mod n = 13$^{\wedge 7}$ mod 33 = 7.

Note that we don't have to calculate the full value of 13 to the power 7 here. We can make use of the fact that a = bc mod n = (b mod n).(c mod n) mod n so we can break down a potentially large number into its components and combine the results of easier, smaller calculations to calculate the final value.

One way of calculating m' is as follows:-

m' = 13$^{\wedge 7}$ mod 33 = 13$^{\wedge(3+3+1)}$ mod 33 = 13$^{\wedge 3}$.13$^{\wedge 3}$.13 mod 33

= (13$^{\wedge 3}$ mod 33).(13$^{\wedge 3}$ mod 33).(13 mod 33) mod 33

= (2197 mod 33).(2197 mod 33).(13 mod 33) mod 33

= 19.19.13 mod 33 = 4693 mod 33

= 7.

Now if we calculate the cipher text c for all the possible values of m (0 to 32), we get

m	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
c	0	1	8	27	31	26	18	13	17	3	10	11	12	19	5	9	4

m	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
c	29	24	28	14	21	22	23	30	16	20	15	7	2	6	25	32

Note that all 33 values of m (0 to 32) map to a unique code c in the same range in a sort of random manner. In this case we have nine values of m that map to the same value of c - these are known as *unconcealed messages*. m = 0 and 1 will always do this for any N, no matter how large. But in practice, higher values shouldn't be a problem when we use large values for N.

If we wanted to use this system to keep secrets, we could let A=2, B=3, ..., Z=27. (We specifically avoid 0 and 1 here for the reason given above). Thus the plaintext message "HELLOWORLD" would be represented by the set of integers m1, m2, ...

{9,6,13,13,16,24,16,19,13,5}

Using our table above, we obtain ciphertext integers c1, c2, ...

{3,18,19,19,4,30,4,28,19,26}

Note that this example is no more secure than using a simple Caesar substitution cipher, but it serves to illustrate a simple example of the mechanics of RSA encryption.

Remember that calculating m^e mod n is easy, but calculating the inverse c^-e mod n is very difficult, well, for large n's anyway. However, if we can factor n into its prime factors p and q, the solution becomes easy again, even for large n's. Obviously, if we can get hold of the secret exponent d, the solution is easy, too.

Key Generation Algorithm

1. Generate two large random primes, p and q, of approximately equal size such that their product n = pq is of the required bit length, e.g. 1024 bits. [See note 1].
2. Compute n = pq and (φ) phi = (p-1)(q-1).
3. Choose an integer e, 1 < e < phi, such that gcd(e, phi) = 1. [See note 2].
4. Compute the secret exponent d, 1 < d < phi, such that ed ≡ 1 (mod phi). [See note 3].
5. The public key is (n, e) and the private key is (n, d). The values of p, q, and phi should also be kept secret.

- n is known as the *modulus*.
- e is known as the *public exponent* or *encryption exponent*.
- d is known as the *secret exponent* or *decryption exponent*.

Encryption

Sender A does the following:-

1. Obtains the recipient B's public key (n, e).
2. Represents the plaintext message as a positive integer m [see note 4].
3. Computes the ciphertext c = m$^{\wedge e}$ mod n.
4. Sends the ciphertext c to B.

Decryption

Recipient B does the following:-

1. Uses his private key (n, d) to compute m = c$^{\wedge d}$ mod n.
2. Extracts the plaintext from the integer representative m.

Source Code:

```
import java.util.*;
import java.io.*;
public class rsa
{       static int gcd(int m,int n)
        {       while(n!=0)
```

```
                {       int r=m%n;
                        m=n;
                        n=r;
                }
                return m;
        }

        public static void  main(String args[])
        {
                int p=0,q=0,n=0,e=0,d=0,phi=0;
                int nummes[]=new int[100];
                int encrypted[]=new int[100];
                int decrypted[]=new int[100];

                int i=0,j=0,nofelem=0;
                Scanner sc=new Scanner(System.in);
                String message ;
                System.out.println("Enter the Message tobe encrypted:");
                message= sc.nextLine();
                System.out.println("Enter value of p and q\n");
                        p=sc.nextInt();
                q=sc.nextInt();
                n=p*q;
                phi=(p-1)*(q-1);

                for(i=2;i<phi;i++)
                if(gcd(i,phi)==1) break;
                e=i;

                for(i=2;i<phi;i++)
                if((e*i-1)%phi==0)
                        break;
                d=i;

                for(i=0;i<message.length();i++)
                {
                        char c = message.charAt(i);
                int a =(int)c;
                nummes[i]=c-96;

                }
                nofelem=message.length();
                for(i=0;i<nofelem;i++)
                {
                        encrypted[i]=1;
                        for(j=0;j<e;j++)
                        encrypted[i] =(encrypted[i]*nummes[i])%n;
                }
                System.out.println("\n Encrypted message\n");
                for(i=0;i<nofelem;i++)
```

```
                    {
                              System.out.print(encrypted[i]);
                              System.out.print((char)(encrypted[i]+96));
                    }
                    for(i=0;i<nofelem;i++)
                    {         decrypted[i]=1;
                              for(j=0;j<d;j++)
                              decrypted[i]=(decrypted[i]*encrypted[i])%n;
                    }

                    System.out.println("\n Decrypted message\n ");
                    for(i=0;i<nofelem;i++)
                    System.out.print((char)(decrypted[i]+96));
                    return;
          }

          }
```

#***

****RESULT****

```
Enter the text:
hello
Enter the value of P and Q :
5
7
Encrypted Text is: 8 h 10 j 17 q 17 q 15 o
Decrypted Text is: hello
```

Experiment No: 6 **Date:**

Congestion Control Using Leaky Bucket Algorithm

Aim: *C 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)).

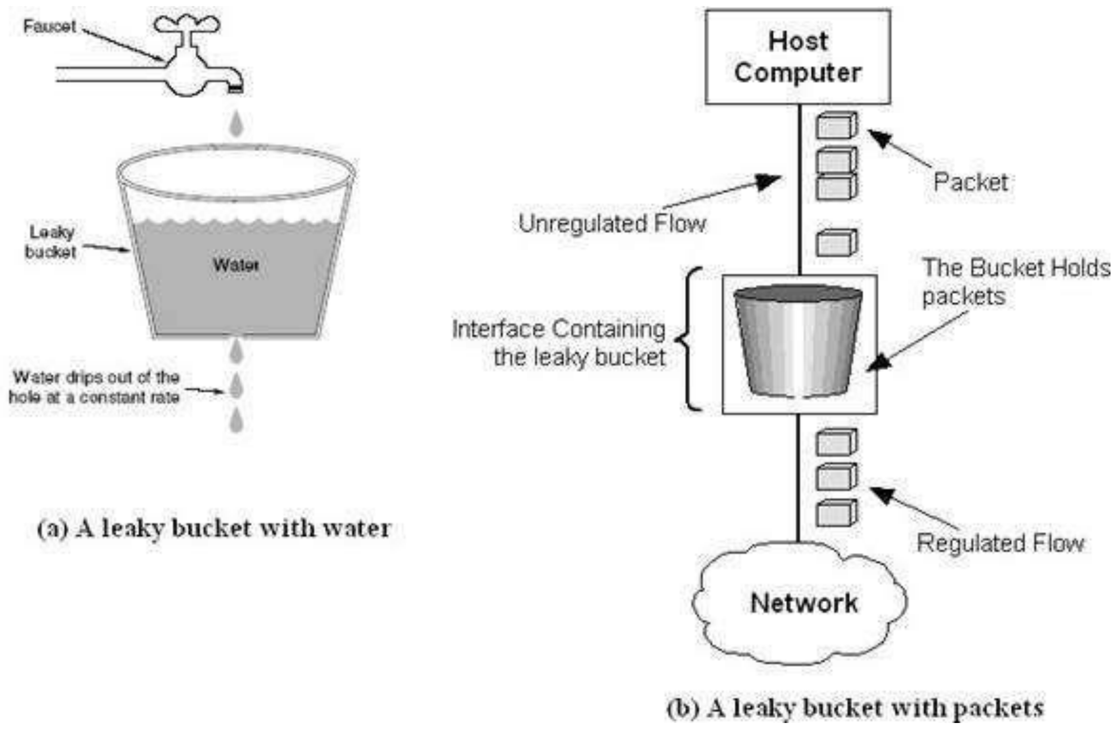

Faucet
Leaky bucket
Water
Water drips out of the hole at a constant rate
(a) A leaky bucket with water
Host Computer
Packet
Unregulated Flow
The Bucket Holds packets
Interface Containing the leaky bucket
Regulated Flow
Network
(b) A leaky bucket with packets

Figure 2.4 - The leaky bucket traffic shaping algorithm

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.

Implementation Algorithm:

```
Steps:
1. Read The Data For Packets
2. Read The Queue Size
3. Divide the Data into Packets
4. Assign the random Propagation delays for each packets to input into the bucket
   (input_packet).
5. wlile((Clock++<5*total_packets)and
         (out_packets< total_paclets))
     a. if (clock == input_packet)
          i. insert into Queue
     b. if (clock % 5 == 0 )
          i. Remove paclet from Queue
6. End
```

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.print("Enter The Size Of The Packet Entering At "+ i+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:

```
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
5
Enter The Size Of The Packet Entering At 1 sec
4
Enter The Size Of The Packet Entering At 1 sec
3
Second|Packet Recieved|Packet Sent|Packet Left|Packet Dropped|
- - - - - ---------------- - - - - - - - - - - - - - - - - - - - - - - - - - -
    1          5             2             3       0
    2          4             2             3       2
    3          3             2             3       1
    4          0             2             1       0
    5          0             1             0       0
```