Karnatak Law Society's
Vishwanathrao Deshpande Institute of Technology
Haliyal – 581329

Computer Networks
[BCS502]
for
Vth Semester B.E.

As prescribed by

VISVESVARAYA TECHNOLOGICAL UNIVERSITY,
BELAGAVI – 590014
(From Academic Year: 2022-23)

Prepared by

Prof. Bhagat G. Inamdar

Department of Computer Science& Engineering

Index

Sl. No.	Content	Page No.
1	Program Outcomes	1
2	Vision , Mission of College and Department, PEO's, PSO's	2
3	Software & Hardware Requirement	6
4	Introduction to NS2	
1	Implement three nodes point – to – point network with duplex links between them. Set the queue size, vary the bandwidth, and find the number of packets dropped.	1
2	Implement transmission of ping messages/trace route over a network topology consisting of 6 nodes and find the number of packets dropped due to congestion.	2
3	Implement an Ethernet LAN using n nodes and set multiple traffic nodes and plot congestion window for different source / destination.	3
4	Develop a program for error detecting code using CRC-CCITT (16- bits).	4
5	Develop a program to implement a sliding window protocol in the data link layer.	5
6	Develop a program to find the shortest path between vertices using the Bellman-Ford and path vector routing algorithm.	6
7	Using TCP/IP sockets, write a client – server program to make the client send the file name and to make the server send back the contents of the requested file if present.	7
8	Develop a program on a datagram socket for client/server to display the messages on client side, typed at the server side.	9
9	Develop a program for a simple RSA algorithm to encrypt and decrypt the data.	10
10	Develop a program for congestion control using a leaky bucket algorithm.	12
Virtual Lab		
1		13
2		14

SIMULATION USING NS-2

Introduction to NS-2:

NS2 is an open-source simulation tool that runs on Linux. It is a discreet event simulator targeted at networking research and provides substantial support for simulation of routing, multicast protocols and IP protocols, such as UDP, TCP, RTP and SRM over wired and wireless (local and satellite) networks.

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 – Tool Command Language

TCL is a very simple programming language. If you have programmed before, you can learn enough to write interesting TCL programs within a few hours. This page provides a quick overview of the main features of TCL. After reading this you'll probably be able to start writing simple TCL scripts on your own; however, we recommend that you consult one of the many available TCL books for more complete information.

Basic syntax

TCL scripts are made up of commands separated by newlines or semicolons. Commands all have the same basic form illustrated by the following example:

```
expr 20 + 10
```

This command computes the sum of 20 and 10 and returns the result, 30. You can try out this example and all the others in this page by typing them to a TCL application such as TCLsh; after a command completes, TCLsh prints its result.

Each TCL command consists of one or more words separated by spaces. In this example there are four words: expr, 20, +, and 10. The first word is the name of a command and the other words are arguments to that command. All TCL commands consist of words, but different commands treat their arguments differently. The expr command treats all of its arguments together as an arithmetic expression, computes the result of that expression, and returns the result as a string. In the expr command the division into words isn't significant: you could just as easily have invoked the same command as

```
expr 20+10
```

However, for most commands the word structure is important, with each word used for a distinct purpose. All TCL commands return results. If a command has no meaningful result then it returns an empty string as its result.

Variables in TCL allows you to store values in variables and use the values later in commands. The set command is used to write and read variables. For example, the following command modifies the variable x to hold the value 32:

```
set x 32
```

The command returns the new value of the variable. You can read the value of a variable by invoking set with only a single argument:

set x

You don't need to declare variables in TCL: a variable is created automatically the first time it is set. TCL variables don't have types: any variable can hold any value. To use the value of a variable in a command, use variable substitution as in the following example:

expr $x*3

When a $ appears in a command, TCL treats the letters and digits following it as a variable name, and substitutes the value of the variable in place of the name. In this example, the actual argument received by the expr command will be 32*3 (assuming that variable x was set as in the previous example). You can use variable substitution in any word of any command, or even multiple times within a word:

set cmd expr set x 11
$cmd $x*$x

Command substitution
You can also use the result of one command in an argument to another command. This is called command substitution:

set a 44
set b [expr $a*4]

When a [appears in a command, TCL treats everything between it and the matching] as a nested TCL command. TCL evaluates the nested command and substitutes its result into the enclosing command in place of the bracketed text. In the example above the second argument of the second set command will be 176.

Quotes and braces

Double-quotes allow you to specify words that contain spaces. For example, consider the following script:

```
set x 24
set y 18
set z "$x + $y is [expr $x + $y]"
```

After these three commands are evaluated variable z will have the value 24 + 18 is 42. Everything between the quotes is passed to the set command as a single word. Note that (a) command and variable substitutions are performed on the text between the quotes, and (b) the quotes themselves are not passed to the command. If the quotes were not present, the set command would have received 6 arguments, which would have caused an error.
Curly braces provide another way of grouping information into words. They are different from quotes in that no substitutions are performed on the text between the curly braces:

set z {$x + $y is [expr $x + $y]}

This command sets variable z to the value "$x + $y is [expr $x + $y]". Control structures
TCL provides a complete set of control structures including commands for conditional execution, looping, and procedures. TCL control structures are just commands that take TCL scripts as arguments. The example below creates a TCL procedure called power, which raises a base to an integer power:

```
proc power {base p} {
   set result 1
   while {$p > 0} {
      set result [expr {$result * $base}]
      set p [expr {$p - 1}]
   }
   return $result
}
```

This script consists of a single command, proc. The proc command takes three arguments: the name of a procedure, a list of argument names, and the body of the procedure, which is a TCL script. Note that everything between the curly brace at the end of the first line and the curly brace on the last line is passed verbatim to proc as a single argument. The proc command creates a new TCL command named power that takes two arguments. You can then invoke power with commands like the following:

```
power 2 6
power 1.15 5
```

When power is invoked, the procedure body is evaluated. While the body is executing it can access its arguments as variables: base will hold the first argument and p will hold the second.
The body of the power procedure contains three TCL commands: set, while, and return.
The while command does most of the work of the procedure. It takes two arguments, an expression ($p > 0) and a body, which is another TCL script. The while command evaluates its expression argument using rules similar to those of the C programming language and if the result is true (nonzero) then it evaluates the body as a TCL script. It repeats this process over and over until eventually the expression evaluates to false (zero). In this case the body of
the while command multiplied the result value by base and then decrements p. When p reaches zero the result contains the desired power of base. The return command causes the procedure to exit with the value of variable result as the procedure's result.

Where do commands come from?

As you have seen, all of the interesting features in TCL are represented by commands. Statements are commands, expressions are evaluated by executing commands, control structures are commands, and procedures are commands.
TCL commands are created in three ways. One group of commands is provided by the TCL interpreter itself. These commands are called builtin commands. They include all of the commands you have seen so far and many more (see below). The builtin commands are present in all TCL applications.
The second group of commands is created using the TCL extension mechanism. TCL provides APIs that allow you to create a new command by writing a command procedure in C or C++ that implements the command. You then register the command procedure with the TCL interpreter by telling TCL the name of the command that the procedure implements. In the future, whenever that

particular name is used for a TCL command, TCL will call your command procedure to execute the command. The builtin commands are also implemented using this same extension mechanism; their command procedures are simply part of the TCL library.

When **TCL** is used inside an application, the application incorporates its key features into TCL using the extension mechanism. Thus the set of available TCL commands varies from application to application. There are also numerous extension packages that can be incorporated into any TCL application. One of the best known extensions is Tk, which provides powerful facilities for building graphical user interfaces. Other extensions provide object-oriented programming, database access, more graphical capabilities, and a variety of other features. One of TCL's greatest advantages for building integration applications is the ease with which it can be extended to incorporate new features or communicate with other resources. The third group of commands consists of procedures created with the proc command, such as the power command created above. Typically, extensions are used for lower-level functions where C programming is convenient, and procedures are used for higher-level functions where it is easier to write in TCL.

Wired TCL Script Components

Create the event scheduler
Open new files & turn on the tracing Create the nodes
Setup the links
Configure the traffic type (e.g., TCP, UDP, etc) Set the time of traffic generation (e.g., CBR, FTP)
Terminate the simulation
NS Simulator Preliminaries.
Initialization and termination aspects of the ns simulator. Definition of network nodes, links, queues and topology. Definition of agents and of applications.
The nam visualization tool.

Tracing and random variables.
Features of NS2
NS2 can be employed in most unix systems and windows. Most of the NS2 code is in
C++. It uses TCL as its scripting language, OTCL adds object orientation to
TCL.NS(version 2) is an object oriented, discrete event driven network simulator that is freely distributed and open source.

- Traffic Models: CBR, VBR, Web etc
- Protocols: TCP, UDP, HTTP, Routing algorithms,MAC etc
- Error Models: Uniform, bursty etc
- Misc: Radio propagation, Mobility models , Energy Models
- Topology Generation tools
- Visualization tools (NAM), Tracing

Structure of NS

- NS is an object oriented discrete event simulator
  Simulator maintains list of events and executes one event after another
  Single thread of control: no locking or race conditions
- Back end is C++ event scheduler
  Protocols mostly
  Fast to run, more control
- Front end is OTCL

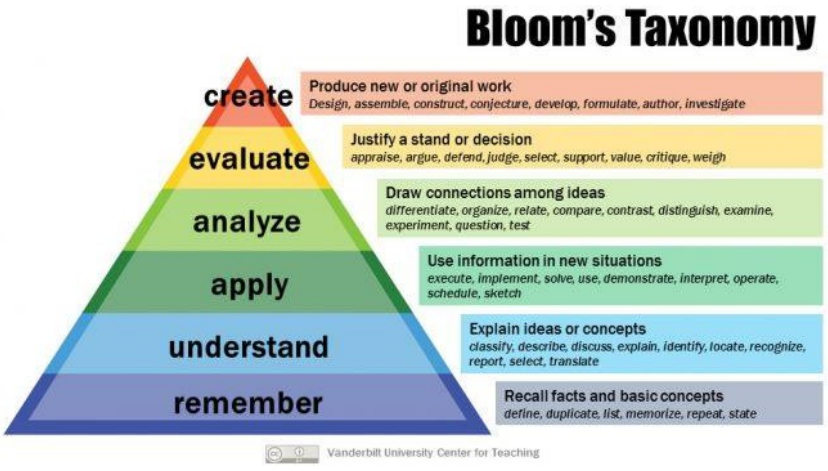
Creating scenarios, extensions to C++ protocols fast to

write and change
Platforms
It can be employed in most unix systems(FreeBSD, Linux, Solaris) and Windows.
Source code
Most of NS2 code is in C++ Scripting
language
It uses TCL as its scripting language OTCL adds object orientation to TCL.

PROGRAM OUTCOMES(POs)

Program Outcomes as defined by NBA (PO) Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.



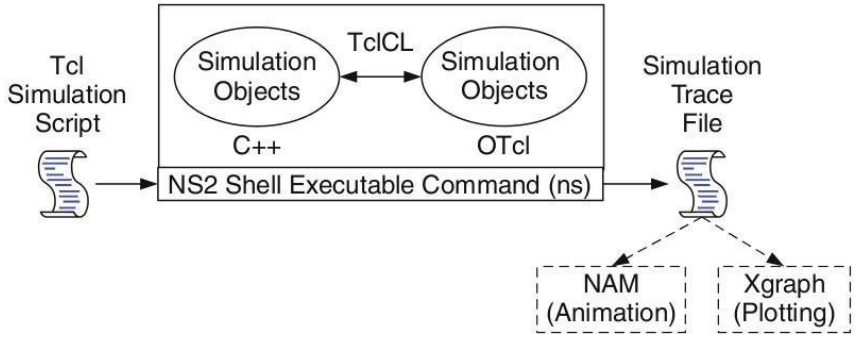
Vision (College)
To grow beyond leaps and bounds as an Institute of par Excellence in the Arena of Technical Education developing Human Resources of High calibre with sound character
Mission (College)
To train the students to Emerge as outstanding skilled Technocrats Imbibed with Professional Ethics and Managerial skills with Commitment to the Society and Nation at Large
Vision (Dept)
 *To grow beyond leaps and bounds as a department of par excellence in the arena of Computer Science and Engineering education in developing human resources meeting global standards.*
Mission (Dept)
To train students with strong theoretical foundation and practical knowledge, to imbibe professional ethics and managerial skills with commitment to the society.. 

Program Educational Objectives (PEO)	
PEO1	To develop an ability to identify and analyze the requirements of Computer Science and Engineering in design and providing novel engineering solutions.
PEO2	To develop abilities to work in team on multidisciplinary projects with effective communication skills, ethical qualities and leadership roles.
PEO3	To develop abilities for successful Computer Science Engineer and achieve higher career goals.
Program Specific Outcomes (PSO)	
PSO 1	To develop ability to model real world problems using appropriate data structure and suitable algorithm in the area of Data Processing, System Engineering, Networking for varying complexity.
PSO 2	To develop an ability to use modern computer languages, environments and platforms in creating innovative career.

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"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 Loop:

While {$i < $n} {
……..
……..
……..
}

Example

```
set x 1

# This is a normal way to write a Tcl while loop.

while {$x < 5} {
    puts "x is $x"
    set x [expr {$x + 1}]
}

puts "exited first loop with X equal to $x\n"
```

For Loop:

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

Example

```
for {set i 0} {$i < 10} {incr i} {
    puts "I inside first loop: $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 begin 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 —finishǁ 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. Inour case, if the buffer capacity of the output queue is exceeded then the last packet to arrive is dropped. Many alternative options exist, such as the RED (Random Early Discard) mechanism, the FQ (Fair Queuing), the DRR (Deficit Round Robin), the stochastic Fair Queuing (SFQ) and the CBQ (which including a priority and a round- robin scheduler).

In ns, an output queue of a node is implemented as a part of each link whose input is that node.

We should also define the buffer capacity of the queue related to each link. An example would be:

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

$ns queue-limit $n0 $n2 20
```

Agents and Applications

We need to define routing (sources, destinations) the agents (protocols) the application that use

them.

FTP over TCP

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

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

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

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

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

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

#Setup a UDP connection

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

#setup a CBR over UDP connection

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

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

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

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

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

CBR over UDP

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

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

```
$cbr set interval_ 0.005
```

The packet size can be set to some value using

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

Scheduling Events

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

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

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

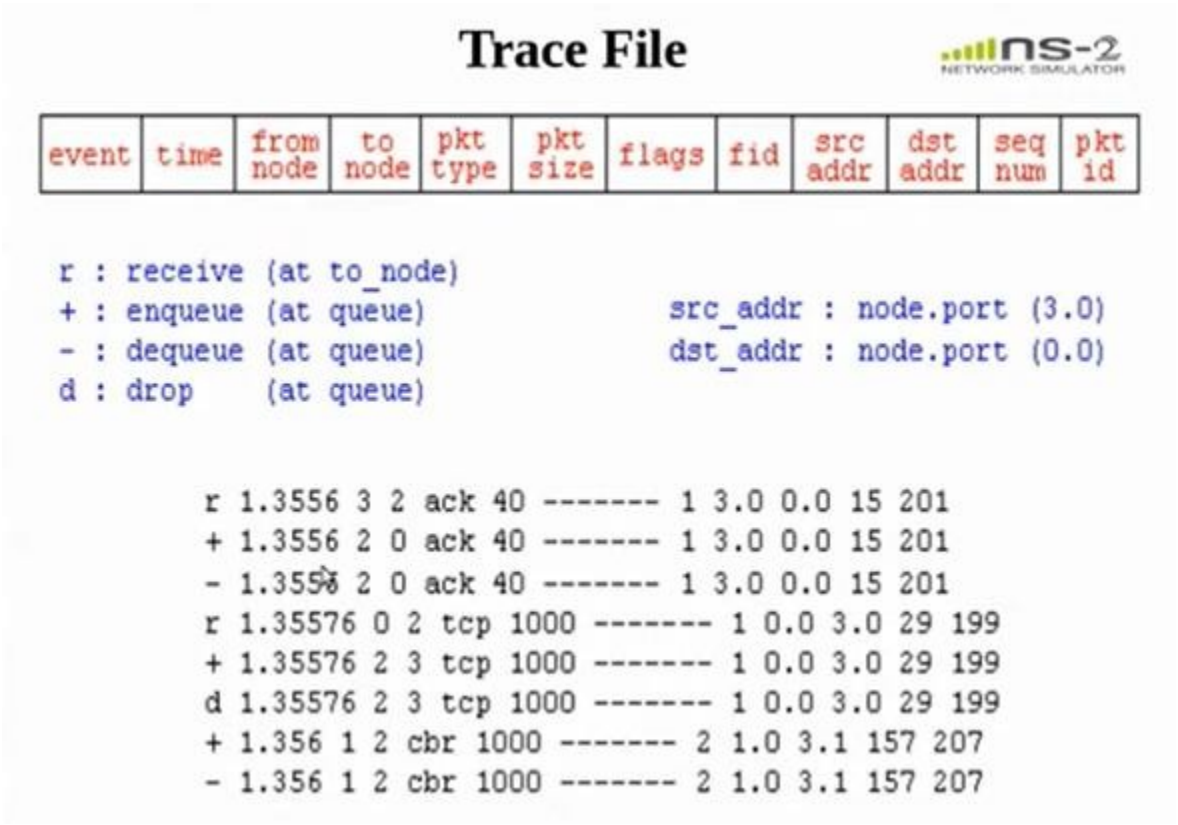
The beginning and end of the FTP and CBR application can be done through the following command

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

$ns at 1.0 " $ftp start"

$ns at 124.0 "$ftp stop"
```

Structure of Trace Files

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

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 {
count =count+1
printf " %3f %20s %-12s %d\n", count,$2,$3,$6 }' empn.lst
```

THE –f OPTION: STORING awk PROGRAMS INA FILE

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

$cat empawk.awk

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

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

THE BEGIN AND END SECTIONS

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

The BEGIN and END sections are optional and take the form

BEGIN {action} END {action}

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

BUILT-IN VARIABLES

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

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

BEGIN {FS= "|"} : This is an alternative to the –F option which does the same thing.

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

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

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

```
$awk 'BEGIN {FS = "|"} NF! =6 {
Print "Record No ", NR, "has", "fields"}' empx.lst
```

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

'$6<4000 {print FILENAME, $0 }: With FILENAME, you can device logic that does different things depending on the file that is processed.

NS2 Installation

- NS2 is a free simulation tool.
- It runs on various platforms including UNIX (or Linux), Windows, and Mac systems.
- NS2 source codes are distributed in two forms: the all-in-one suite and the component- wise.
- 'all-in-one' package provides an "install" script which configures the NS2 environment and creates NS2 executable file using the "make" utility.

NS-2 installation steps in Linux

- Go to **Computer** File System now paste the zip file **"ns-allinone-2.34.tar.gz"** into opt folder.
- Now**unzip** the file by typing the following **command**
  [root@localhost opt] # **tar -xzvf ns-allinone-2.34.tar.gz**
- After the files get extracted, we get ns-allinone-2.34 folder as well as zip file ns-allinone- 2.34.tar.gz
- Now go to ns-allinone-2.33 folder and install it
- [root@localhost opt] # cd ns-allinone-2.34
- [root@localhost ns-allinone-2.33] # ./install
- After the completion of the process paste ns2.sh file to the location /etc/profile.d
- This completes the installation process of "NS-2" simulator.

EXPERIMENT NO: 01 **Date:**

Three nodes point – to – point network with duplex links between them.

AIM: To 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.

EXPLANATION:

There are different types of communications connection in existence between two endpoints., Home networks, and the Internet is the one of the most commonly used examples. Many types of devices are and several different methods are used connecting to these type of network architectures. There are different advantages and disadvantages to this type of network architectures. Connecting computers to these types of networks, we require some networking framework to create the connections. The two different computer network connection types are getting discussed in this page are Point-to-Point Connection and multipoint connection. The main difference between them we can getting discussed with the help of below definition.

Point-to-Point Connection

A **point-to-point connection** is a direct link between two devices such as a computer and a printer. It uses dedicated link between the devices. The entire capacity of the link is used for the transmission between those two devices. Most of today's point-to-point connections are associated with modems and PSTN (Public Switched Telephone Network) communications. In point to point networks, there exist many connections between individual pairs of machines.

To move from sources to destination, a packet (short message) may follow different routes. In networking, the Point-to-Point Protocol (PPP) is a data link protocol commonly used in establishing a direct connection between two networking nodes. It can provide connection authentication, transmission encryption, and compression PPP is used over many types of physical networks including serial cable, phone line, trunk line, cellular telephone, specialized radio links, and fiber optic links such as SONET. PPP is also used over Internet access connections (now marketed as "broadband").

Internet service providers (ISPs) have used PPP for customer dial-up access to the Internet, since IP packets cannot be transmitted over a modem line on their own, without some data link protocol. Two encapsulated forms of PPP, Point-to-Point Protocol over Ethernet (PPPoE) and Point-to-Point Protocol over ATM (PPPoA), are used most commonly by Internet Service Providers (ISPs) to establish a Digital Subscriber Line (DSL) Internet service connection with customers. PPP is commonly used as a data link layer protocol for connection over synchronous and asynchronous circuits, where it has largely superseded the older Serial Line Internet Protocol (SLIP) and telephone company mandated standards (such as Link Access Protocol, Balanced (LAPB) in the X.25 protocol suite). PPP was designed to work with numerous network layer protocols, including Internet Protocol (IP), TRILL, Novell's Internetwork Packet Exchange

(IPX), NBF and AppleTalk.

Multipoint Connection

A multipoint connection is a link between three or more devices. It is also known as Multi-drop configuration. The networks havjng multipoint configuration are called **Broadcast Networks**. In broadcast network, a message or a packet sent by any machine is received by all other machines in a network. The packet contains address field that specifies the receiver. Upon receiving a packet, every machine checks the address field of the packet. If the transmitted packet is for that particular machine, it processes it; otherwise it just ignores the packet.

Broadcast network provides the provision for broadcasting & multicasting. Broadcasting is the process in which a single packet is received and processed by all the machines in the network. It is made possible by using a special code in the address field of the packet. When a packet is sent to a subset of the machines i.e. only too few machines in the network it is known as multicasting. Historically, multipoint connections were used to attach central CPs to distributed dumb terminals. In today's LAN environments, multipoint connections link many network devices in various configurations.

PROGRAM:

Prog1.tcl

```
set ns [new Simulator]
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set f [open prog1.tr w]
$ns trace-all $f
set nf [open prog1.nam w]
$ns namtrace-all $nf
$ns duplex-link $n0 $n1 5Kb 2ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns queue-limit $n1 $n2 5
set udp1 [new Agent/UDP]
$ns attach-agent $n0 $udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp1
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.0005
set null1 [new Agent/Null]
```

```
$ns attach-agent $n2 $null1
$ns connect $udp1 $null1
proc finish { } {
global ns f nf
$ns flush-trace
close $f
close $nf
exec nam prog1.nam &
exec awk -f prog1.awk prog1.tr &
exit 0
}
$ns at 0.1 "$cbr1 start"
$ns at 2.0 "finish"
$ns run
```

Prog1.awk

```
BEGIN {        c=0;     }
{
if($1=="d")
{        c++;     }
}
END {
printf("Number of packets dropped are %d\n",c);
}
```

Steps for execution

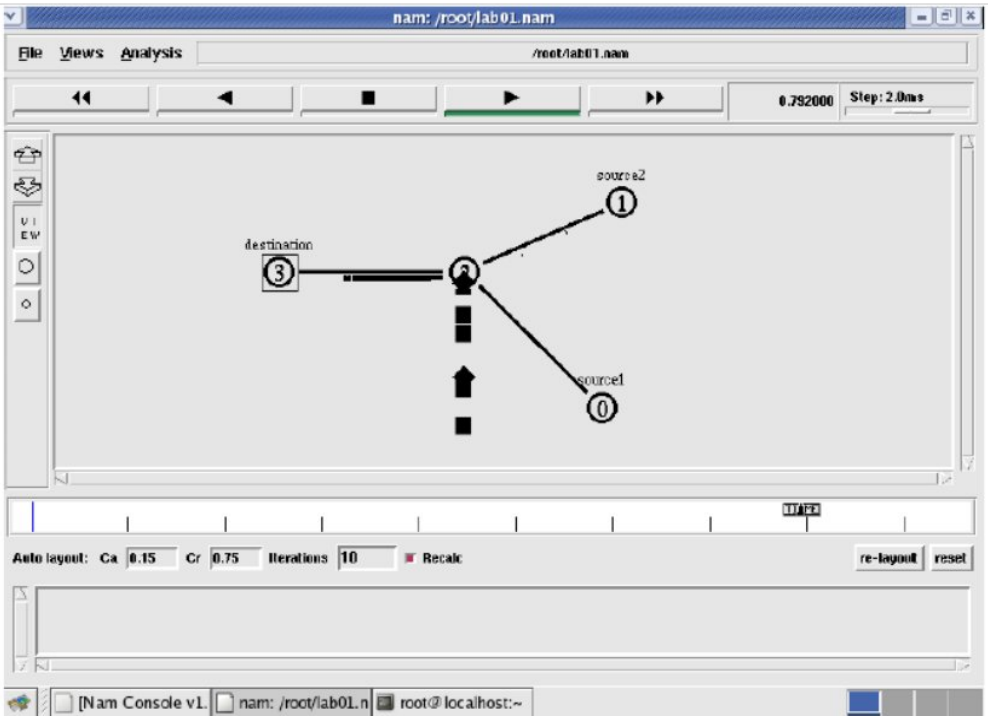
vi lab1.tcl
[root@localhost~]# vi lab1.awk
[root@localhost~]# ns lab1.tcl
[root@localhost~]# awk –f lab1.awk lab1.tr
[root@localhost~]# vi lab1.tr

Note:

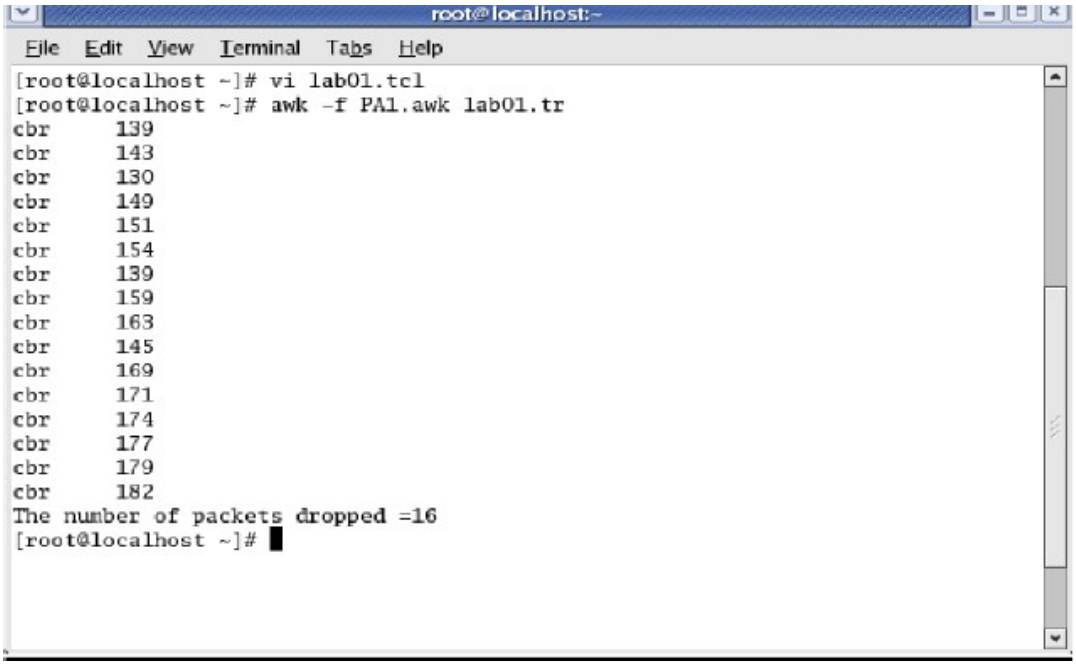
1. Set the queue size fixed from n0 to n2 as 10, n1-n2 to 10 and from n2-n3 as 5. Syntax: To set the queue size
   $ns set queue-limit <from><to><size> Eg:
   $ns set queue-limit $n0 $n2 10
2. Go on varying the bandwidth from 10, 20, 30 and so on.
3. Find the number of packets dropped at the node 2

Trace file contains 12 columns: -

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



OUTPUT



CONCLUSION: At the different bandwidth the dropped rate of the packet varies. If the bandwidth is enormously wide the drop of packet can be stopped.

EXPERIMENT 2 **Date:**

Transmission of ping messages/trace route over a network topology.

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.

EXPLANATION:

Ping is a computer network administration software utility used to test the reachability of a host on an Internet Protocol (IP) network. It is available for virtually all operating systems that have networking capability, including most embedded network administration software.

Ping measures the round-trip time for messages sent from the originating host to a destination computer that are echoed back to the source. The name comes from active sonar terminology that sends a pulse of sound and listens for the echo to detect objects under water.

Ping operates by sending Internet Control Message Protocol (ICMP) echo request packets to the target host and waiting for an ICMP echo reply. The program reports errors, packet loss, and a statistical summary of the results, typically including the minimum, maximum, the mean round-trip times, and standard deviation of the mean.

The command-line options of the ping utility and its output vary between the numerous implementations. Options may include the size of the payload, count of tests, limits for the number of network hops (TTL) that probes traverse, and interval between the requests. Many systems provide a companion utility ping6, for testing on Internet Protocol version 6 (IPv6) networks, which implement ICMPv6.

The following is the output of running ping on Linux for sending five probes to the target host *www.example.com*:

```
$ ping -c 5 www.example.com
PING www.example.com (93.184.216.34): 56 data bytes
64 bytes from 93.184.216.34: icmp_seq=0 ttl=56
time=11.632 ms 64 bytes from 93.184.216.34:
icmp_seq=1 ttl=56 time=11.726 ms 64 bytes from
93.184.216.34: icmp_seq=2 ttl=56 time=10.683
```

ms 64 bytes from 93.184.216.34: icmp_seq=3
ttl=56 time=9.674 ms 64 bytes from
93.184.216.34: icmp_seq=4 ttl=56 time=11.127
ms
--- www.example.com ping statistics ---
5 packets transmitted, 5 packets received, 0.0% packet loss round-trip
min/avg/max/stddev = 9.674/10.968/11.726/0.748 ms

In cases of no response from the target host, most implementations display either nothing or periodically print notifications about timing out. Possible ping results indicating a problem include the following:

- H, !N or !P – host, network or protocol unreachable
- S – source route failed
- F – fragmentation needed
- U or !W – destination network/host unknown
- I – source host is isolated
- A – communication with destination network administratively prohibited
- Z – communication with destination host administratively prohibited
- Q – for this ToS the destination network is unreachable
- T – for this ToS the destination host is unreachable
- X – communication administratively prohibited
- V – host precedence violation
- C – precedence cutoff in effect

In case of error, the target host or an intermediate router sends back an ICMP error message, for example "host unreachable" or "TTL exceeded in transit". In addition, these messages include the first eight bytes of the original message (in this case header of the ICMP echo request, including the quench value), so the ping utility can match responses to originating queries.

ICMP PACKET

IPv4 Datagram				
	Bits 0–7	**Bits 8–15**	**Bits 16–23**	**Bits 24–31**
Header (20 bytes)	Version/IHL	Type of service	Length	
	Identification		*flags* and *offset*	
	Time To Live (TTL)	Protocol	Header Checksum	
	Source IP address			
	Destination IP address			
ICMP Header (8 bytes)	Type of message	Code	Checksum	
	Header Data			
ICMP Payload (*optional*)	Payload Data			

IPv6 Datagram						
	Bits 0–3	Bits 4–7	Bits 8–11	Bits 12–15	Bits 16–23	Bits 24–31
Header (40 bytes)	Version	Traffic Class		Flow Label		
	Payload Length				Next Header	Hop Limit
	Source Address					
	Destination Address					
ICMP6 Header (8 bytes)	Type of message		Code		Checksum	
	Header Data					
ICMP6 Payload (*optional*)	Payload Data					

Generic composition of an ICMP packet:

- IPv4 Header (in blue): *protocol* set to 1 (ICMP) and *Type of Service* set to 0.
- IPv6 Header (in blue): *Next Header* set to 58 (ICMP6)
- ICMP Header (in red): o Type of ICMP message (8 bits)
  - Code (8 bits)
  - Checksum (16 bits), calculated with the ICMP part of the packet (the IP header is not used). It is the 16-bit one's complement of the one's complement sum of the ICMP message starting with the Type field
  - Header Data (32 bits) field, which in this case (ICMP echo request and replies), will be composed of identifier (16 bits) and sequence number (16 bits).
- ICMP Payload: *payload* for the different kind of answers; can be an arbitrary length, left to implementation detail. However, the packet including IP and ICMP headers must be less than the maximum transmission unit of the network or risk being fragmented.

Echo Request:

0 0	**0 1**	**0 2**	**0 3**	**0 4**	**0 5**	**0 6**	**0 7**	**0 8**	**0 9**	**1 0**	**1 1**	**1 2**	**1 3**	**1 4**	**1 5**	**1 6**	**1 7**	**1 8**	**1 9**	**2 0**	**2 1**	**2 2**	**2 3**	**2 4**	**2 5**	**2 6**	**2 7**	**2 8**	**2 9**	**3 0**	**3 1**
Type = 8(IPv4, ICMP) 128(IPv6,ICMP6)								Code = 0								Checksum															
Identifier																Sequence Number															
Payload																															

The Identifier and Sequence Number can be used by the client to match the reply with the request that caused the reply. In practice, most Linux systems use a unique identifier for every ping process, and sequence number is an increasing number within that process. Windows uses a fixed identifier, which varies between Windows versions, and a sequence number that is only reset at boot time.

Echo reply: The*echo reply* is an ICMP message generated in response to an echo request; it is mandatory for all hosts, and must include the exact payload received in the request.

0 0	**0 1**	**0 2**	**0 3**	**0 4**	**0 5**	**0 6**	**0 7**	**0 8**	**0 9**	**1 0**	**1 1**	**1 2**	**1 3**	**1 4**	**1 5**	**1 6**	**1 7**	**1 8**	**1 9**	**2 0**	**2 1**	**2 2**	**2 3**	**2 4**	**2 5**	**2 6**	**2 7**	**2 8**	**2 9**	**3 0**	**3 1**
Type = 0(IPv4,ICMP) 129(IPv6,ICMP6)								Code = 0								Checksum															
Identifier																Sequence Number															
Payload																															

Payload

The payload of the packet is generally filled with ASCII characters, as the output of the tcpdump utility shows in the last 32 bytes of the following example (after the eight-byte ICMP header starting with 0x0800):

16:24:47.966461 IP (tos 0x0, ttl 128, id 15103, offset 0, flags [none],
proto: ICMP (1), length: 60) 192.168.146.22 > 192.168.144.5: ICMP echo request,
id 1, seq 38, length 40
0x0000: 4500 003c 3aff 0000 8001 5c55 c0a8 9216 E..<:.....\U....
0x0010: c0a8 9005 0800 4d35 0001 0026 6162 6364 ......M5...&abcd
0x0020: 6566 6768 696a 6b6c 6d6e 6f70 7172 7374 efghijklmnopqrst
0x0030: 7576 7761 6263 6465 6667 6869 uvwabcdefghi

The payload may include a timestamp indicating the time of transmission and a sequence number, which are not found in this example. This allows ping to compute the round trip time in a stateless manner without needing to record the time of transmission of each packet.

The payload may also include a *magic packet* for the Wake-on-LAN protocol, but the minimum payload in that case is longer than shown. The *Echo Request* typically does not receive any reply if the host was sleeping in hibernation state, but the host still wakes up from sleep state if its interface is configured to accept wakeup requests. If the host is already active and configured to allow replies to incoming ICMP *Echo Request* packets, the returned reply should include the same payload. This may be used to detect that the remote host was effectively woken up, by repeating a new request after some delay to allow the host to resume its network services. If the host was just sleeping in low power active state, a single request wakes up that host just enough to allow its *Echo Reply* service to reply instantly if that service was enabled. The host does not need to completely wake up all devices, and may return to low power mode after a short delay. Such configuration may be used to avoid a host to enter in hibernation state, with much longer wake up delay, after some time passed in low power active mode.

PROGRAM

Prg2.tcl

```
set ns [new simulator]
set nf [open lab2.nam w ]
$ns namtrace-all $nf
set tf [open lab2.tr w ]
$ns trace-all $tf
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
$n4 shape box
$ns duplex-link $n0 $n4 1005Mb 1ms DropTail
$ns duplex-link $n1 $n4 50Mb 1ms DropTail
$ns duplex-link $n2 $n4 2000Mb 1ms DropTail
$ns duplex-link $n3 $n4 200Mb 1ms DropTail
$ns duplex-link $n4 $n5 1Mb 1ms DropTail
set p1 [new Agent/Ping]
$ns attach-agent $n0 $p1
$p1 set packetSize_ 50000
$p1 set interval_ 0.001
set p2 [new Agent/Ping]
$ns attach-agent $n1 $p2
set p3 [new Agent/Ping]
$ns attach-agent $n2 $p3
$p3 set packetSize_ 30000
$p3 set interval_ 0.00001
set p4 [new Agent/Ping]
$ns attach-agent $n3 $p4
set p5 [new Agent/Ping]
$ns attach-agent $n5 $p5
$ns queue-limit $n0 $n4 5
$ns queue-limit $n2 $n4 3
$ns queue-limit $n4 $n5 2
Agent/Ping instproc recv {from rtt} {
$self instvar node_
puts "node [$node_ id] recevived 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 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 &
exec awk -f prg2.awk lab2.tr &
exit 0
}
$ns at 0.1 "$p1 send"
$ns at 0.2 "$p1 send"
$ns at 0.3 "$p1 send"
$ns at 0.4 "$p1 send"
$ns at 0.5 "$p1 send"
$ns at 0.6 "$p1 send"
$ns at 0.7 "$p1 send"
$ns at 0.8 "$p1 send"
$ns at 0.9 "$p1 send"
$ns at 1.0 "$p1 send"
$ns at 1.1 "$p1 send"
$ns at 1.2 "$p1 send"
$ns at 1.3 "$p1 send"
$ns at 1.4 "$p1 send"
$ns at 1.5 "$p1 send"
$ns at 1.6 "$p1 send"
$ns at 1.7 "$p1 send"
$ns at 1.8 "$p1 send"
$ns at 1.9 "$p1 send"
$ns at 2.0 "$p1 send"
$ns at 2.1 "$p1 send"
$ns at 2.2 "$p1 send"
$ns at 2.3 "$p1 send"
$ns at 2.4 "$p1 send"
$ns at 2.5 "$p1 send"
```

```
$ns at 2.6 "$p1 send"
$ns at 2.7 "$p1 send"
$ns at 2.8 "$p1 send"
$ns at 2.9 "$p1 send"
$ns at 0.1 "$p3 send"
$ns at 0.2 "$p3 send"
$ns at 0.3 "$p3 send"
$ns at 0.4 "$p3 send"
$ns at 0.5 "$p3 send"
$ns at 0.6 "$p3 send"
$ns at 0.7 "$p3 send"
$ns at 0.8 "$p3 send"
$ns at 0.9 "$p3 send"
$ns at 1.0 "$p3 send"
$ns at 1.1 "$p3 send"
$ns at 1.2 "$p3 send"
$ns at 1.3 "$p3 send"
$ns at 1.4 "$p3 send"
$ns at 1.5 "$p3 send"
$ns at 1.6 "$p3 send"
$ns at 1.7 "$p3 send"
$ns at 1.8 "$p3 send"
$ns at 1.9 "$p3 send"
$ns at 2.0 "$p3 send"
$ns at 2.1 "$p3 send"
$ns at 2.2 "$p3 send"
$ns at 2.3 "$p3 send"
$ns at 2.4 "$p3 send"
$ns at 2.5 "$p3 send"
$ns at 2.6 "$p3 send"
$ns at 2.7 "$p3 send"
$ns at 2.8 "$p3 send"
$ns at 2.9 "$p3 send"
$ns at 3.0 "finish"
$ns run
```

Prg2.awk

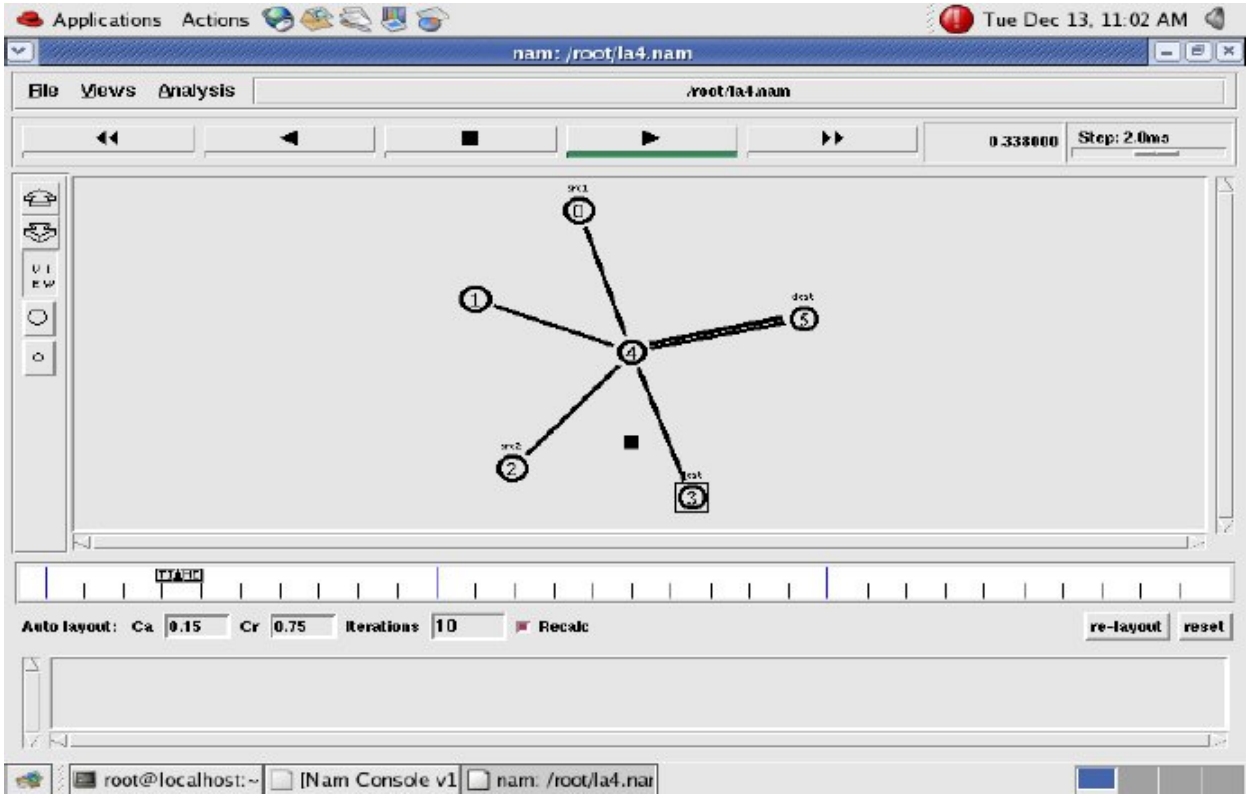
```
BEGIN {
        drop=0;
        }
```

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

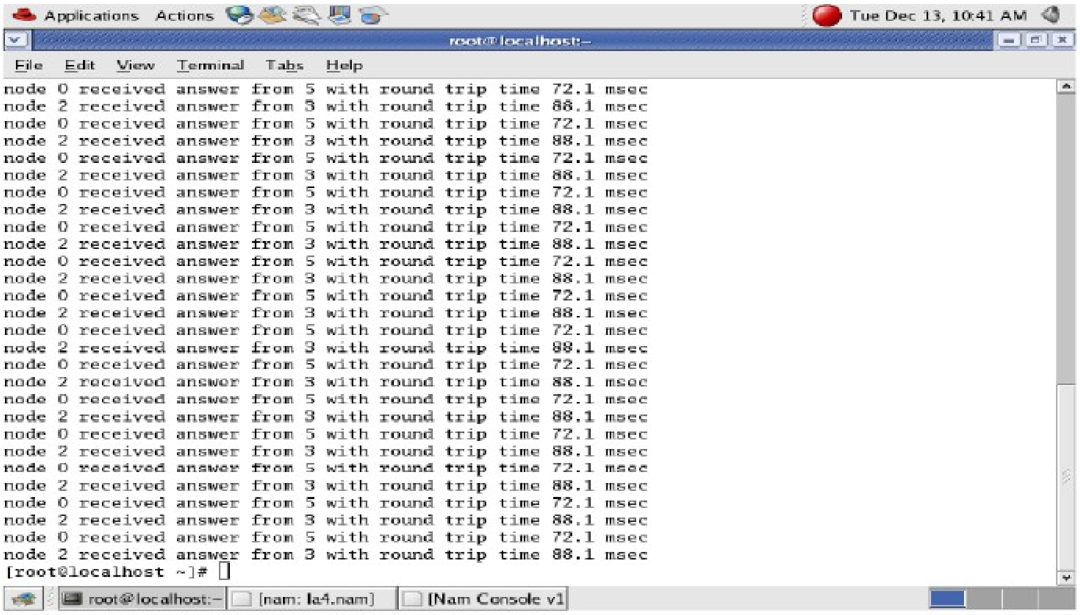
Steps for execution

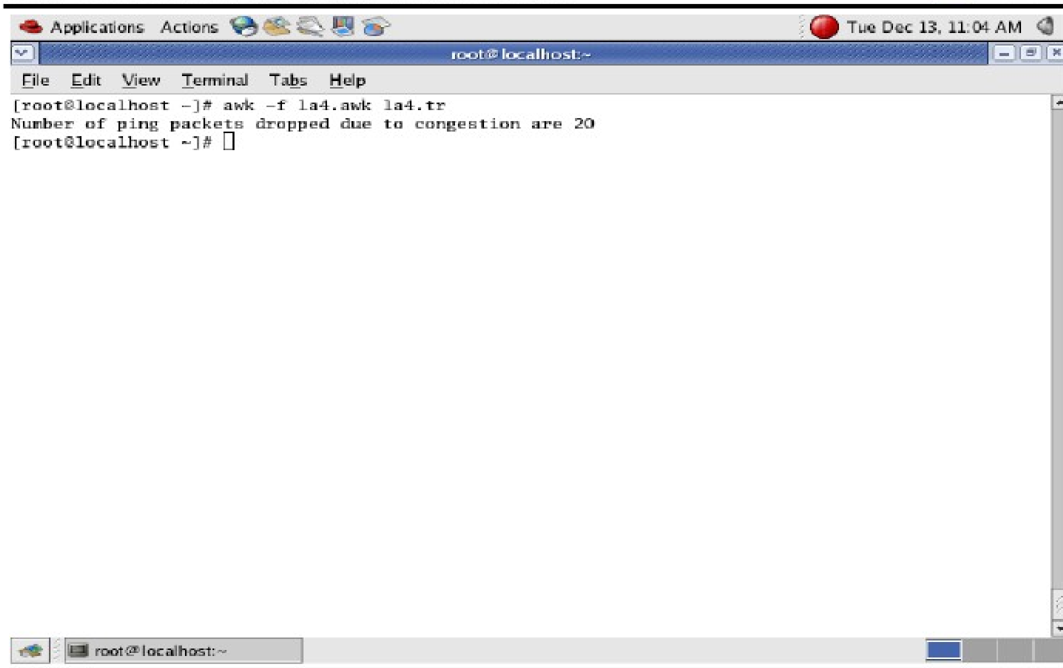
[root@localhost ~]# vi lab2.tcl
[root@localhost ~]# vi lab2.awk
[root@localhost~]# ns lab2.tcl
[root@localhost~]# awk –f lab2.awk lab2.tr
[root@localhost~]# vi lab2.tr

Topology



OUTPUT





Note:

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

CONCLUSION: As the ping messages are negligibly small the drop of such packets are comparatively a very rare occurrence.

EXPERIMENT: 3 **Date:**

ETHERNET LAN DESIGN

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

EXPLANATION:

Ethernet LAN:

Ethernet is the technology that is most commonly used in wired local area networks (LANs). A LAN is a network of computers and other electronic devices that covers a small area such as a room, office, or building. It is used in contrast to a wide area network (WAN), which spans much larger geographical areas. Ethernet is a network protocol that controls how data is transmitted over a LAN. Technically it is referred to as the IEEE 802.3 protocol. The protocol has evolved and improved over time to transfer data at the speed of a gigabit per second.

Many people have used Ethernet technology their whole lives without knowing it. It is most likely that any wired network in your office, at the bank, and at home is an Ethernet LAN. Most desktop and laptop computers come with an integrated Ethernet card inside so they are ready to connect to an Ethernet LAN. **What You Need in an Ethernet LAN**

To set up a wired Ethernet LAN, you need the following:

- **Computers and devices to connect**: Ethernet connects any computer or other electronic devices to its network as long as the device has an Ethernet adapter or network card.
- **Network interface cards in the devices**: A network interface card is either integrated into the motherboard of the computer or installed separately in the device. You also have USB versions of Ethernet cards such as external dongles. An Ethernet card is known as a network card. It has ports where you can connect cables. There may be two ports, one for an RJ-45 jack that connects unshielded twisted pair (UTP) cables and one for a coaxial jack on the network card.
- A **router, hub** or **gateway to connect your devices**: A hub is a device that acts as a connecting point between devices on a network. It consists of several RJ-45 ports to which you plug the cables.
- Cables: UTP cables are commonly used in Ethernet LANs. This is the same type of cable used for landline telephone sets, but fatter, with eight twisted pairs of wires of different colors inside. The end is crimped with an RJ-45 jack, which is a larger version of the RJ-11 jacks that plug into your landline phone. When the Ethernet spans beyond a room to a greater distance, coaxial cable is used. This is the same cable with a round single-core jack you use for a TV.
- **Software to manage the network**: Modern operating systems like recent versions of Windows, Linux and MacOS are more than sufficient to manage Ethernet LANs. Third-party software that gives more features and better control is available.

How Ethernet Works

Ethernet requires technical knowledge in computer science to understand the mechanism behind the Ethernet protocol fully. Here is a simple explanation: When a machine on the network wants to send data to another, it senses the carrier, which is the main wire connecting all the devices. If it is free, meaning no one is sending anything, it sends the data packet on the network, and all other devices check the packet to see whether they are the recipient. The recipient consumes the packet. If there is already a packet on the highway, the device that wants to send holds back for some thousandths of a second to try again until it can send.

Make Lan:

$ns_ make-lan nodelist bw delay LL ifq MAC channel phy

Creates a lan from a set of nodes given by <nodelist>. Bandwidth, delay characteristics along with the link-layer, Interface queue, Mac layer and channel type for the lan also needs to be defined. Default values used are as follows:

<LL> .. LL
<ifq>.. Queue/DropTail
<MAC>.. Mac
<channel>.. Channel and
<phy>.. Phy/WiredPhy

PROGRAM

Prg3.tcl

```
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_
$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
```

Prg3.awk

```
BEGIN {
}
{
        if($6=="cwnd_")
        {
        printf("%f\t%f\t\n",$1,$7);
        }
}
END {
}
```

Steps for execution

[root@localhost ~]# vi lab3.tcl

[root@localhost ~]# vi lab3.awk

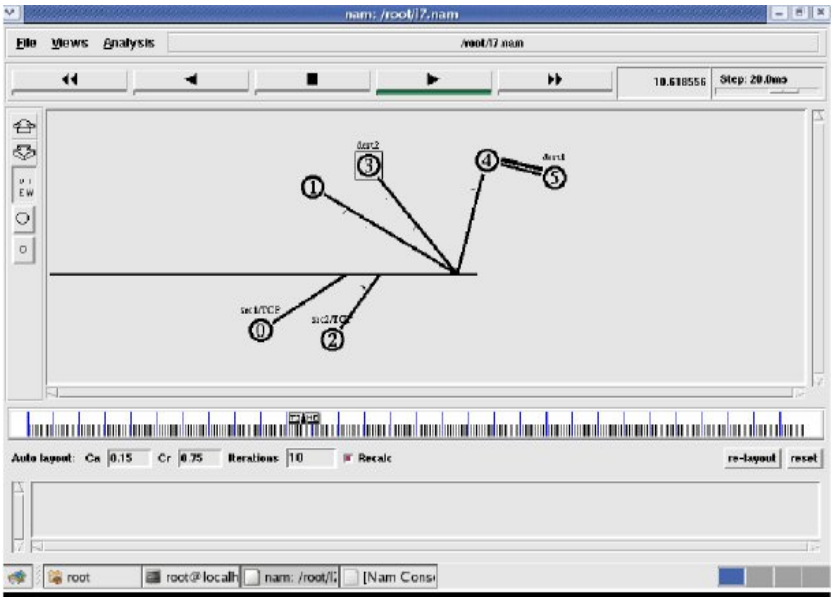
[root@localhost~]# ns lab3.tcl

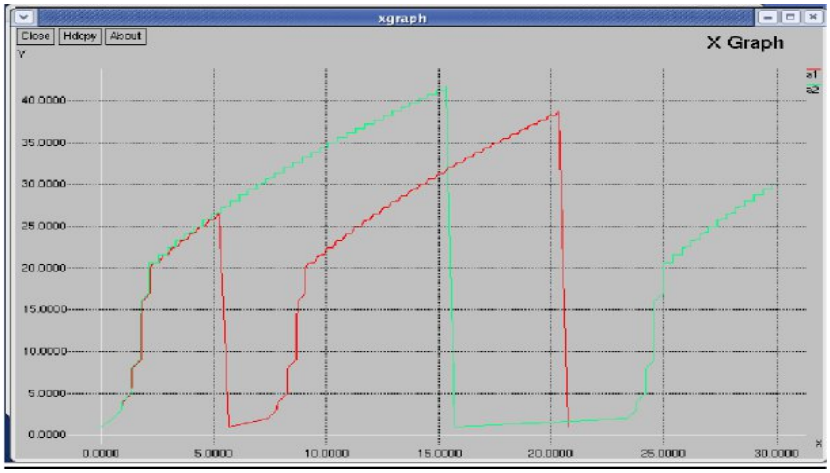
[root@localhost~]# awk –f lab3.awk file1.tr >a1

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

[root@localhost~]# xgraph a1 a2

Topology





OUTPUT

CONCLUSION: The congestion window must be considered for transmission of packets

EXPERIMENT 4: **Date:**

CRC - CCITT

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

EXPLAINATION:

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

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

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.
International Standard CRC Polynomials

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 xn,
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

ration

Crc.java

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class Crc {
public static void main(String args[]) throws IOException
{
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
int[ ] data;
int[ ] div;
int[ ] divisor;
int[ ] rem;
int[ ] crc;
int data_bits, divisor_bits, tot_length;
System.out.println("Enter number of data bits : ");
data_bits=Integer.parseInt(br.readLine());
data=new int[data_bits];
System.out.println("Enter data bits : ");
for(int i=0; i<data_bits; i++)
data[i]=Integer.parseInt(br.readLine());
System.out.println("Enter number of bits in generator polynomial : ");
divisor_bits=Integer.parseInt(br.readLine());
divisor=new int[divisor_bits];
System.out.println("Enter generator polynomial bits : ");
for(int i=0; i<divisor_bits; i++)
{
divisor[i]=Integer.parseInt(br.readLine());
}
System.out.println();
System.out.print("divisor bits are : ");
for(int i=0; i< divisor_bits; i++)
{
System.out.print(divisor[i]);
```

```
}
System.out.println();
tot_length=data_bits+divisor_bits-1;
div=new int[tot_length];
rem=new int[tot_length];
crc=new int[tot_length];
/*------------------ CRC GENERATION-----------------------*/
for(int i=0;i<data.length;i++)
{
div[i]=data[i];
}
System.out.print("Dividend (after appending 0's) are : ");
for(int i=0; i< div.length; i++)
{
System.out.print(div[i]);
}
System.out.println();
for(int j=0; j<div.length; j++)
{
rem[j] = div[j];
}
rem=divide(div, divisor, rem);
for(int i=0;i<div.length;i++) //append dividend and ramainder
{
crc[i]=(div[i]^rem[i]);
}
System.out.println();
System.out.println("CRC code : ");
for(int i=0;i<crc.length;i++)
System.out.print(crc[i]);
System.out.println();
/*-------------------ERROR DETECTION---------------------*/
System.out.println("Enter CRC code of "+tot_length+" bits : ");
for(int i=0; i<crc.length; i++)
{
crc[i]=Integer.parseInt(br.readLine());
}
for(int j=0; j<crc.length; j++)
```

```
{
rem[j] = crc[j];
}
rem=divide(crc, divisor, rem);
for(int i=0; i< rem.length; i++)
{
if(rem[i]!=0)
{
System.out.println("Error"); break;
}
if(i==rem.length-1)
System.out.println("No Error");
}
System.out.println("THANK YOU.... :)");
}
static int[] divide(int div[],int divisor[], int rem[])
{
int cur=0;
while(true)
{
for(int i=0;i<divisor.length;i++)
rem[cur+i]=(rem[cur+i]^divisor[i]);
while(rem[cur]==0 && cur!=rem.length-1)
cur++;
if((rem.length-cur)<divisor.length)
break;
}
return rem;
}
}
```

OUTPUT

CASE 1:

```
Enter number of data bits :
4
Enter data bits :
1011
Enter number of bits in generator polynomial :
17
Enter generator polynomial bits :
10001000000100001
divisor bits are : 10001000000100001
Dividend (after appending 0's) are : 10110000000000000000
CRC code :
10111011000101101011
Enter CRC code of 20 bits :
10111011000101010 11
No Error
THANK YOU.... :)
```

CASE 2:

```
Enter number of data bits :
4
Enter data bits :
1101
Enter number of bits in generator polynomial :
17
Enter generator polynomial bits :
10001000000100001
divisor bits are : 10001000000100001
Dividend (after appending 0's) are : 11010000000000000000
CRC code :
11011101000110101101
Enter CRC code of 20 bits :
11011101001110101101
Error
```

CONCLUSION: CRC can be used for Error Detection but not Correction

EXPERIMENT 5: **Date:**

SLIDING WINDOW PROTOCOL

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

EXPLAINATION:

The **Sliding Window Protocol** is a key method used in the **Data Link Layer** to provide **reliable and efficient data transmission** over a network. It is especially important in environments where acknowledgments (ACKs) may be delayed or lost, and retransmissions are required.

The Sliding Window Protocol controls the flow of data between two devices (sender and receiver) by maintaining a window of frames that can be sent without waiting for acknowledgment. This technique improves bandwidth utilization and supports **full-duplex communication**.

There are two types of sliding window protocols:

1. **Go-Back-N ARQ**
2. **Selective Repeat ARQ**

1. **Sender Side:**
   - Maintains a window of size N (configured by the protocol or network conditions).
   - Can send N frames without waiting for an ACK.
   - When an ACK is received for a frame, the window slides forward.
   - If an ACK is not received within a timeout period, the sender resends the frame(s):
   - In **Go-Back-N**, it resends all frames from the unacknowledged frame onward.
   - In **Selective Repeat**, only the specific unacknowledged frame(s) are resent.
2. **Receiver Side:**
   - Sends ACK for each correctly received frame.
   - In Go-Back-N, it discards out-of-order frames.
   - In Selective Repeat, it buffers out-of-order frames and reorders them later.

Sliding.java

```
import java.util.Scanner;

public class SlidingWindowProtocol {
    static int windowSize;
    static int totalFrames;

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Input total number of frames
        System.out.print("Enter total number of frames to send: ");
        totalFrames = scanner.nextInt();

        // Input window size
        System.out.print("Enter sliding window size: ");
        windowSize = scanner.nextInt();

        int sent = 0;
        while (sent < totalFrames) {
            // Send frames in window
            int i;
            System.out.println("\nSending frames:");
            for (i = 0; i < windowSize && (sent + i) < totalFrames; i++) {
                System.out.println("Sent Frame: " + (sent + i));
            }

            // Simulate ACKs
            for (i = 0; i < windowSize && (sent + i) < totalFrames; i++) {
                System.out.print("Is ACK received for Frame " + (sent + i) + "? (yes/no): ");
                String ack = scanner.next();

                if (ack.equalsIgnoreCase("no")) {
                    System.out.println("Timeout! Resending from Frame " + (sent + i));
                    break;
                }
            }

            if (i == windowSize) {
                sent += windowSize;
```

```
            } else {
                sent += i; // Re-send from the frame which caused timeout
            }
        }



        System.out.println("\nAll frames sent successfully!");
        scanner.close();
    }
}
```

Output

Enter total number of frames to send: 7
Enter sliding window size: 3

Sending frames:
Sent Frame: 0
Sent Frame: 1
Sent Frame: 2
Is ACK received for Frame 0? yes
Is ACK received for Frame 1? yes
Is ACK received for Frame 2? no
Timeout! Resending from Frame 2

Sending frames:
Sent Frame: 2
Sent Frame: 3
Sent Frame: 4
...

EXPERIMENT 6: **Date:**

Bellman-Ford

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

BELLMAN-FORD ALGORITHM

AIM: Write a program to find the shortest path between vertices using bellman-ford algorithm.

EXPLAINATION

Bellman- Ford

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:

- Send my routing table to all my neighbors whenever my link table changes
- When I get a routing table from a neighbor on port P with link metric M: add L to each of the neighbor's metrics
- for each entry (D, P', M') in the updated neighbor's table: if I do not have an entry for D, add (D, P, M') to my routing table.
- if I have an entry for D with metric M'', add (D, P, M') to my routing table if M' < M''
- if my routing table has changed, send all the new entries to all my neighbors.

Bellmanford.java

```
import java.util.Scanner;

public class Bellmanford {
    private int D[];
    private int num_ver;
    public static final int MAX_VALUE = 999;

    public Bellmanford(int num_ver) {
        this.num_ver = num_ver;
        D = new int[num_ver + 1];
    }

    public void BellmanFordEvaluation(int source, int[][] A) {
        // Step 1: Initialize distances
        for (int node = 1; node <= num_ver; node++) {
            D[node] = MAX_VALUE;
        }
        D[source] = 0;

        // Step 2: Relax all edges (V - 1) times
        for (int i = 1; i <= num_ver - 1; i++) {
            for (int sn = 1; sn <= num_ver; sn++) {
                for (int dn = 1; dn <= num_ver; dn++) {
                    if (A[sn][dn] != MAX_VALUE && D[sn] != MAX_VALUE) {
                        if (D[dn] > D[sn] + A[sn][dn]) {
                            D[dn] = D[sn] + A[sn][dn];
                        }
                    }
                }
            }
        }

        // Step 3: Check for negative weight cycle
        for (int sn = 1; sn <= num_ver; sn++) {
            for (int dn = 1; dn <= num_ver; dn++) {
                if (A[sn][dn] != MAX_VALUE && D[sn] != MAX_VALUE) {
                    if (D[dn] > D[sn] + A[sn][dn]) {
                        System.out.println("The graph contains a negative edge cycle.");
                        return;
                    }
                }
            }
        }
```

```
// Step 4: Print result
        for (int vertex = 1; vertex <= num_ver; vertex++) {
            System.out.println("Distance from source " + source + " to vertex " + vertex + " is " + D[vertex]);
        }
    }

    public static void main(String[] args) {
        int num_ver, source;
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter the number of vertices:");
        num_ver = scanner.nextInt();

        int[][] A = new int[num_ver + 1][num_ver + 1];
        System.out.println("Enter the adjacency matrix:");
        for (int sn = 1; sn <= num_ver; sn++) {
            for (int dn = 1; dn <= num_ver; dn++) {
                A[sn][dn] = scanner.nextInt();
                if (sn == dn) {
                    A[sn][dn] = 0;
                } else if (A[sn][dn] == 0) {
                    A[sn][dn] = MAX_VALUE;
                }
            }
        }

        System.out.println("Enter the source vertex:");
        source = scanner.nextInt();

        Bellmanford b = new Bellmanford(num_ver);
        b.BellmanFordEvaluation(source, A);
        scanner.close();
    }
}
```

OUTPUT

```
Enter the number of vertices
4
Enter the adjacency matrix
0 5 0 0
5 0 3 4
0 3 0 2
0 4 2 0
Enter the source vertex
2
Distance of source 2 to 1 is 5
Distance of source 2 to 2 is 0
Distance of source 2 to 3 is 3
Distance of source 2 to 4 is 4
```

CONCLUSION: The shortest path can be found using Bellman for algorithm

EXPERIMENT 7: **Date:**

CLIENT SERVER USING TCP/IP SOCKETS

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

EXPLAINATION

Socket

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

Server.java – TCP File Transfer Server

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

public class Server {
   public static void main(String args[]) throws Exception {
      String filename;
      Scanner sc = new Scanner(System.in);
      System.out.print("Enter File Name: ");
      filename = sc.nextLine();
      sc.close();

      while (true) {
         ServerSocket ss = new ServerSocket(5000); // Server on port 5000
         System.out.println("Waiting for request...");
         Socket s = ss.accept();
         System.out.println("Connected with " + s.getInetAddress().toString());

         DataInputStream din = new DataInputStream(s.getInputStream());
         DataOutputStream dout = new DataOutputStream(s.getOutputStream());
```

```
try {
            String str = din.readUTF();
            System.out.println("Received command: " + str);

            if (!str.equals("stop")) {
               System.out.println("Sending File: " + filename);
               dout.writeUTF(filename);
               dout.flush();

               File f = new File(filename);
               FileInputStream fin = new FileInputStream(f);
               long sz = f.length();
               byte b[] = new byte[1024];
               int read;

               dout.writeUTF(Long.toString(sz)); // Send file size
               dout.flush();
               System.out.println("File Size: " + sz + " bytes");

               while ((read = fin.read(b)) != -1) {
                  dout.write(b, 0, read);
                  dout.flush();
               }

               fin.close();
               System.out.println("File sent successfully.");
            }

            dout.writeUTF("stop");
            System.out.println("Send Complete");
            dout.flush();
         } catch (Exception e) {
            e.printStackTrace();
            System.out.println("An error occurred");
         }

         din.close();
         dout.close();
         s.close();
         ss.close();
      }
   }
```

```
}
```

Client.java – TCP File Transfer Client

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

public class Client {
   public static void main(String args[]) throws Exception {
      Scanner sc = new Scanner(System.in);
      System.out.print("Enter Server Address: ");
      String address = sc.nextLine();

      Socket s = new Socket(address, 5000);
      DataInputStream din = new DataInputStream(s.getInputStream());
      DataOutputStream dout = new DataOutputStream(s.getOutputStream());
      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

      System.out.println("Type 'start' to begin receiving the file...");
      String str = "";

      while (!str.equals("start"))
         str = br.readLine();

      dout.writeUTF(str);
      dout.flush();

      String filename = din.readUTF();
      System.out.println("Receiving file: " + filename);
      filename = "client_" + filename;
      System.out.println("Saving as: " + filename);

      long fileSize = Long.parseLong(din.readUTF());
      System.out.println("File Size: " + fileSize + " bytes");

      byte b[] = new byte[1024];
      FileOutputStream fos = new FileOutputStream(new File(filename), true);
      int bytesRead;

      System.out.println("Receiving file...");
      while ((bytesRead = din.read(b, 0, b.length)) != -1) {
         fos.write(b, 0, bytesRead);
         if (bytesRead < 1024) break;
      }
```

```
        fos.close();
        dout.close();
        s.close();
        System.out.println("File transfer complete.");
    }
}
```

Server Side:

```
Enter File Name: india.jpg
Waiting for request...
Connected with /127.0.0.1
Received command: start
Sending File: india.jpg
File Size: 1399 bytes
File sent successfully.
Send Complete
```

Client Side:

```
Enter Server Address: 127.0.0.1
Type 'start' to begin receiving the file...
start
Receiving file: india.jpg
Saving as: client_india.jpg
File Size: 1399 bytes
Receiving file...
File transfer complete.
```

EXPERIMENT 8: **Date:**

CLIENT SERVER USING DATAGRAM SOCKETS

Aim: A datagram socket AIM: Write a program on datagram socket for client/server to display the messages on client side, typed at the server side.

EXPLAINATION

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

Server.java – UDP Server

```
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.Scanner;

public class Server {
   public static void main(String[] args) throws IOException {
      DatagramSocket serverSocket = new DatagramSocket(9870);
      System.out.println("Server started on socket number 9870");

      byte[] receiveData = new byte[1024];
      byte[] sendData;

      // Receive packet from client
      DatagramPacket receivePacket = new DatagramPacket(receiveData,
receiveData.length);
      serverSocket.receive(receivePacket);

      String clientMessage = new String(receivePacket.getData(), 0,
receivePacket.getLength());
      System.out.println("Client Connected: " + clientMessage);

      // Get client IP and port
      InetAddress clientIPAddress = receivePacket.getAddress();
      int clientPort = receivePacket.getPort();
```

```
            // Prepare response
                Scanner input = new Scanner(System.in);
                System.out.print("Enter the message to be sent: ");
                String message = input.nextLine();
                sendData = message.getBytes();
                input.close();

                // Send packet to client
                DatagramPacket sendPacket = new DatagramPacket(sendData,
        sendData.length, clientIPAddress, clientPort);
                serverSocket.send(sendPacket);

                serverSocket.close();
                System.out.println("Server message sent. Server closed.");
            }
}
```

Client.java – UDP Client

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

public class Client {
   public static void main(String[] args) throws IOException {
      BufferedReader inFromUser = new BufferedReader(new
InputStreamReader(System.in));
      DatagramSocket clientSocket = new DatagramSocket();

      InetAddress IPAddress = InetAddress.getByName("localhost");

      byte[] sendData;
      byte[] receiveData = new byte[1024];

      System.out.print("Enter START to connect to Server: ");
      String sentence = inFromUser.readLine();
      sendData = sentence.getBytes();
```

```
            // Send START message to server
                DatagramPacket sendPacket = new DatagramPacket(sendData,
            sendData.length, IPAddress, 9870);
                clientSocket.send(sendPacket);

                // Receive response from server
                DatagramPacket receivePacket = new DatagramPacket(receiveData,
            receiveData.length);
                clientSocket.receive(receivePacket);

                String modifiedMessage = new String(receivePacket.getData(), 0,
            receivePacket.getLength());
                System.out.println("Message Received from Server: " +
            modifiedMessage);

                clientSocket.close();
            }
}
```

Server Side:

Server started on socket number 9870
Client Connected: START
Enter the message to be sent: Hello, This is 5TH Semester
Server message sent. Server closed.

Client Side:
Enter START to connect to Server: START
Message Received from Server: Hello, This is 5TH Semester

EXPERIMENT: 09 **Date:**

RSA IMPLEMENTATION

AIM: Write a program for simple RSA algorithm to encrypt and decrypt the data.

EXPLAINATION:

RSA

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^-1 mod phi = 3^-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 = me mod n = 7^3 mod 33 = 343 mod 33 = 13. Hence the ciphertext c = 13. To check decryption, we compute m' = cd mod n = 13^7 mod 33 = 7.

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 = me 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 = cd mod n.

2. Extracts the plaintext from the integer representative m.

RSA.java

```
import java.io.IOException;
import java.math.BigInteger;
import java.util.Random;
import java.util.Scanner;

public class RSA {
   private BigInteger p;
   private BigInteger q;
   private BigInteger N;
   private BigInteger phi;
   private BigInteger e;
   private BigInteger d;
   private int bitlength = 1024;
   private Random r;

   public RSA() {
      r = new Random();
      p = BigInteger.probablePrime(bitlength, r);
      q = BigInteger.probablePrime(bitlength, r);
```

```
        N = p.multiply(q);
        phi =
p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));
        e = BigInteger.probablePrime(bitlength / 2, r);

 while (phi.gcd(e).compareTo(BigInteger.ONE) > 0 && e.compareTo(phi) <
0) {
            e = e.add(BigInteger.ONE);
        }

        d = e.modInverse(phi);
    }

    public RSA(BigInteger e, BigInteger d, BigInteger N) {
        this.e = e;
        this.d = d;
        this.N = N;
    }

    public static void main(String[] args) throws IOException {
        RSA rsa = new RSA();
        Scanner in = new Scanner(System.in);

        System.out.print("Enter the plain text: ");
        String teststring = in.nextLine();

        System.out.println("Encrypting String: " + teststring);
        System.out.println("String in Bytes: " +
bytesToString(teststring.getBytes()));

        // Encrypt
        byte[] encrypted = rsa.encrypt(teststring.getBytes());
        System.out.println("Encrypted String = " + bytesToString(encrypted));

        // Decrypt
        byte[] decrypted = rsa.decrypt(encrypted);
        System.out.println("Decrypting Bytes: " + bytesToString(decrypted));
        System.out.println("Decrypted String: " + new String(decrypted));

        in.close();
    }
```

```
    private static String bytesToString(byte[] encrypted) {
        StringBuilder result = new StringBuilder();
        for (byte b : encrypted) {
            result.append(b).append(" ");
        }
        return result.toString();
    }




    // Encrypt message
    public byte[] encrypt(byte[] message) {
        return (new BigInteger(message)).modPow(e, N).toByteArray();
    }

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

Output:

```
Enter the plain text:
SEM 5
Encrypting String: SEM 5
String in Bytes: 83 69 77 32 53
Encrypted String = 0 -103 -105 -67 63 12 55 31 ... (truncated for brevity)
Decrypting Bytes: 83 69 77 32 53
Decrypted String: SEM 5
```

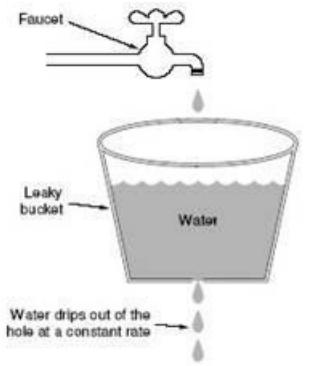
EXPERIMENT: 10 **Date:**

LEAKY BUCKET ALGORITHM IMPLEMENTATION

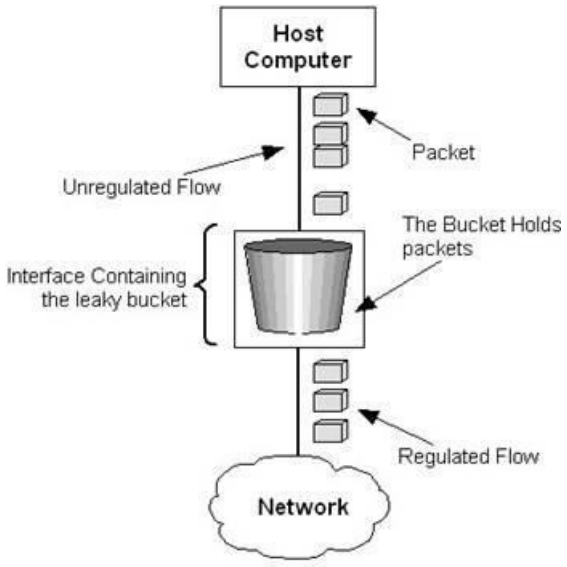
AIM: Write a program for congestion control using leaky bucket algorithm.
EXPLAINATION

Leaky Bucket

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



(a) A leaky bucket with water



(b) A leaky bucket with packets

While leaky bucket eliminates completely bursty traffic by regulating the incoming data flow its main drawback is that it drops packets if the bucket is full. Also, it doesn't take into account the idle process of the sender which means that if the host doesn't transmit data for some time the bucket becomes empty without permitting the transmission of any packet.

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. while((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

Queue.java

```
import java.util.Scanner;
public class Queue {
   int[] q = new int[10]; // Queue of fixed size 10
   int f = 0, r = 0;
   private Scanner in;
   void insert(int n) {
      in = new Scanner(System.in);
      for (int i = 0; i < n; i++) {
         System.out.print("Enter " + i + " element: ");
         int ele = in.nextInt();
         if (r >= 10) {
            System.out.println("Queue is full");
            System.out.println("Lost Packet: " + ele);
            break;
         } else {
            q[r] = ele;
```

```
            r++;
        }
    }
}

void delete() throws InterruptedException {
    if (r == 0) {
        System.out.println("Queue is empty.");
    } else {
        // Simulate leaking with delay
        for (int i = f; i < r; i++) {
            Thread.sleep(100); // simulate time delay in leaking
            System.out.println("Leaked Packet: " + q[i]);
        }
    }
    System.out.println();
}
}
```

Leaky.java

```
import java.util.Scanner;

public class Leaky {
    public static void main(String[] args) throws Exception {
        Queue q = new Queue();
        Scanner src = new Scanner(System.in);

        System.out.println("Enter the number of packets to be sent:");
        int size = src.nextInt();

        q.insert(size); // insert packets into queue
        q.delete();     // leak the packets (simulate transmission)

        src.close();
    }
}
```

Output

Enter the number of packets to be sent: 12
Enter 0 element: 10
Enter 1 element: 20
Enter 2 element: 30
Enter 3 element: 40
Enter 4 element: 50
Enter 5 element: 60
Enter 6 element: 70
Enter 7 element: 80
Enter 8 element: 90
Enter 9 element: 100
Enter 10 element: 110
Queue is full
Lost Packet: 110

Leaked Packet: 10
Leaked Packet: 20
Leaked Packet: 30
Leaked Packet: 40
Leaked Packet: 50
Leaked Packet: 60
Leaked Packet: 70
Leaked Packet: 80
Leaked Packet: 90
Leaked Packet: 100

Virtual Lab Experiments

EXPERIMENT: 01 **Date:**

Public-Key Cryptosystems

AIM: To get familiar with fundamental concepts of ns2.

Algorithm :

RSA Algorithm

Key Generation (at A)

Select two large primes p, q such that p is not equal to q

Compute n = p * q

Compute phi(n) = (p-1) * (q-1)

Select 'e' such that gcd (e, phi(n)) = 1

Compute d = e-1 mod phi(n)

A's Public key is (e, n);

A's Private key is d

Encryption

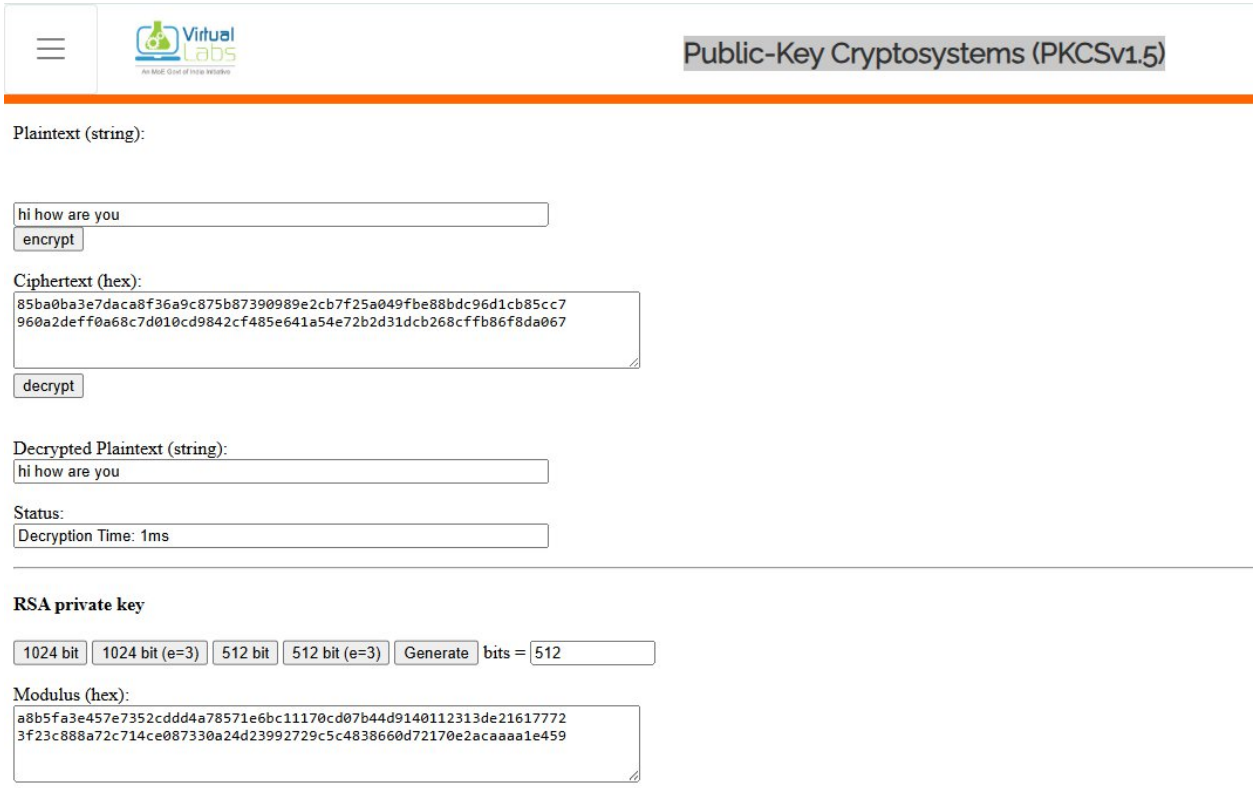
Any party B wishing to send a message M to party A encrypts M using RSA as:

C = Me mod n

Decryption

Party A decrypts 'C', received from party B, using his Private key d

M = Cd mod n


Virtual Labs
Public-Key Cryptosystems (PKCSv1.5)
Plaintext (string):
hi how are you
encrypt
Ciphertext (hex):
85ba0ba3e7daca8f36a9c875b87390989e2cb7f25a049fbe88bdc96d1cb85cc7
960a2deff0a68c7d010cd9842cf485e641a54e72b2d31dcb268cffb86f8da067
decrypt
Decrypted Plaintext (string):
hi how are you
Status:
Decryption Time: 1ms
RSA private key
1024 bit
1024 bit (e=3)
512 bit
512 bit (e=3)
Generate
bits = 512
Modulus (hex):
a8b5fa3e457e7352cddd4a78571e6bc11170cd07b44d9140112313de21617772
3f23c888a72c714ce087330a24d23992729c5c4838660d72170e2acaaaa1e459

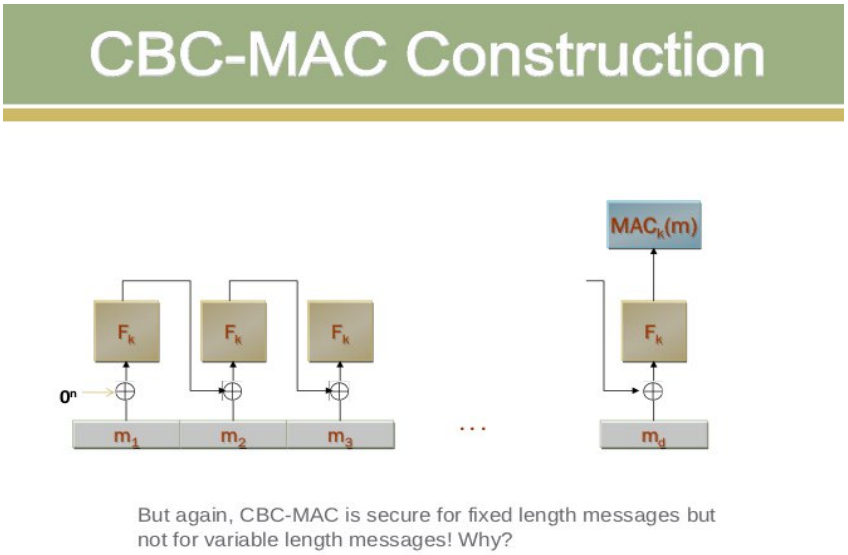
EXPERIMENT: 02 **Date:**

Message Authentication Code (CBC-MAC)

Aim: A Cipher Block Chaining Message Authentication Code (CBC-MAC), is a technique for constructing a message authentication code from a block cipher.

About the experiment:

In the experiment we provide a "dummy" block cipher (that is just a function and has none of the properties of a real block cipher) and ask you to use it to compute the CBC_MAC tag of an arbitrary message.



STEP 1 : Select a plaintext to be encrypted.
STEP 2 : Select 'l' such that l < (length of plaintext)/2.
STEP 3 : Select an Initialization Vector, IV of length l.
STEP 4 : Get the value of F(x) by providing a string of length l in the field titled 'Your text' and clicking on 'Apply Function'.
STEP 5 : Repeat this for as many times as you need to get the encrypted text for the plaintext selected in Step 1. You can change the function if you wish to.
STEP 6 : Once you have learnt to work with the basic CBC-MAC, you can make it secure either by prepending the message length or by using multiple keys (as described in the theory section). You may accordingly work with the second part of the experiment (this is very similar to the first part and hence your answer in not checked)


Virtual Labs
Message Authentication Code (CBC-MAC)
Prepend the message with its length
Plaintext:
11000000001111100101010
Next Plaintext
Your Plaintext:
HI HOW ARE YOU
Key:
11100100
Another Key
length of Initialization Vector (IV), l,
4
where l < (the length of plaintext above)/2
IV:
1100
Next IV
Put your text of size l to get the corresponding value of F_k(text) of size l.
Your text:
11001010
Apply Function
Next Function
Function output:
Final Output: